UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Set-Based Particle Swarm Optimization

by

Joost Langeveld

Submitted in partial fulfillment of the requirements for the degree
Master of Science (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

September 2015

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Publication data:

Joost Langeveld. Set-Based Particle Swarm Optimization. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, September 2015.

Electronic, hyperlinked versions of this dissertation are available online, as Adobe PDF files, at:

`http://cirg.cs.up.ac.za/`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Set-Based Particle Swarm Optimization

by

Joost Langeveld
E-mail: jclangev@gmail.com

Abstract

Particle swarm optimization (PSO) algorithms have been successfully applied to discrete-valued optimization problems. However, in many cases the algorithms have been tailored specifically for the problem at hand. This study proposes a generic set-based particle swarm optimization algorithm, called SBPSO, for use on discrete-valued optimization problems that can be formulated as set-based problems. The performance of the SBPSO is then evaluated on two different discrete optimization problems: the multidimensional knapsack problem (MKP) and the feature selection problem (FSP) from machine learning. In both cases, the SBPSO is compared to three other discrete PSO algorithms from literature. On the MKP, the SBPSO is shown to outperform, with statistical significance, the other algorithms. On the FSP and using a *k*-nearest neighbor classifier, the SBPSO is shown to outperform, with statistical significance, the other algorithms. When a Gaussian Naive Bayes or a J48 decision tree classifier is used, no algorithm can be shown to outperform on the FSP.

Keywords: Particle Swarm Optimization, Swarm Intelligence, Computational Intelligence, Discrete Optimization, Mathematical Sets, Multidimensional Knapsack Problem, Machine Learning, Feature Selection Problem

Supervisor : Prof. Andries P. Engelbrecht
Department : Department of Computer Science
Degree : Master of Science

© University of Pretoria

Pre-face

The document before you is the dissertation of Joost Langeveld for the purpose of obtaining a Master Degree from the University of Pretoria, department of Computer Science. Parts of this work have previously been presented at the 2011 International Conference on Swarm Intelligence, in Cergy, France as "A generic set-based particle swarm optimization algorithm" [79] and published in the journal of Swarm Intelligence as "Set-based particle swarm optimization applied to the multidimensional knapsack problem" [80].

Problem statement

Swarm intelligence research has yielded positive results in a variety of application areas. Particle swarm optimization (PSO) was introduced by Kennedy and Eberhart [62] in 1995, and it forms a key area of swarm intelligence research. Originally used to solve continuous optimization problems, PSO was adapted for discrete optimization problems as well [62]. Later work has seen PSO algorithms built around the concept of mathematical sets [24, 102, 143].

This study claims that a functioning, generic, set-based PSO algorithm does not yet exist. Furthermore, this study proposes a new algorithm and claims that this algorithm *is* a functioning, generic, set-based PSO algorithm. An abstract formulation of the SBPSO's update equations in terms of set-theory will be derived. Then the new algorithm will be applied and evaluated on two set-based optimization problems, namely the multi-dimensional knapsack problem (MKP), and the feature selection problem (FSP). The influence of the algorithms' control parameters on performance will also be investigated.

Set-based optimization problems are discrete or combinatorial optimization problems that *allow for a natural representation using elements, whereby the problem is to find an optimal subset of these elements*. Many combinatorial problems can be defined using sets of elements, but for most problems this is not the natural representation: such a set based representation makes the problem harder to solve. Set-based problems thus form a subset of discrete optimization problems.

The MKP is a discrete optimization problem where the value of a number of items to be included in a knapsack is to be maximized, subject to 0-1 constraints on the items and multiple further "weight" constraints. It was first formulated in 1955 in terms of a capital allocation problem by Lorie and Savage [89].

The FSP arises in machine learning where algorithms are used to classify data [46]. The context for the FSP is a supervised learning problem in which a classifier is used to determine to which discrete class

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

a given instance of data belongs. The FSP itself is defined as the problem to select an optimal subset of features and use only those features to train the classifier for the original classification problem.

Objectives

The main objective of this thesis is to develop and investigate a functioning, generic, set-based PSO algorithm that can solve discrete optimization problems (DOPs). To reach this goal, the following sub-objectives are identified:

- To give an overview of existing discrete and set-based PSO algorithms from literature to show that a functioning, generic, set-based PSO algorithm does not yet exist;
- To determine the basic components that make up a PSO algorithm in order to determine what components should be present to make the new set-based algorithm a PSO algorithm;
- To determine which additional components are required to make the new algorithm a functioning algorithm;
- To define the new algorithm by formulating the PSO update equations in terms of set-theory;
- To ensure the new algorithm is generic and does *not* include any problem domain specific features in the algorithm itself - the only link to the problem domain should lie in the fitness function;
- To test the new algorithm on different DOPs, for which the MKP and the FSP are selected;
- To compare the performance (in terms of quality of the solution found) of the new algorithm against discrete PSO algorithms known from literature which have been applied to the MKP and FSP; and
- To investigate the new algorithm's control parameters to determine which values yield good results.

For clarity, it is important to note which possible objectives are left *outside* of this thesis' scope:

- To find an algorithm that is better at solving the MKP or the FSP than known state-of-the-art algorithms;
- To find the most efficient algorithm in terms of number of iterations or fitness function evaluations, total number of computations (flops) or total time needed to complete; and
- To compare the performance of the new algorithm against non-PSO methods used to solve DOPs.

Contributions

The main contributions of this thesis to the field of swarm intelligence are:

- Developing a functioning, generic, set-based PSO algorithm called SBPSO;
- A first application and investigation of SBPSO on MKP and FSP;

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- Developing a new scoring mechanism for use in the PSO's parameter tuning that can also be used for a sensitivity analysis to determine good control parameter values for SBPSO and the relative importance of each parameter; and
- Performing a thorough comparison between SBPSO and three other discrete PSO algorithms on a large enough set of problems to determine statistically significant difference in quality of solutions found.

Outline

The remainder of this thesis consists of five parts. Part I introduces concepts, algorithms and problems from literature that form the basis and further context for the original research presented in this thesis. Chapter 1 introduces the field of particle swarm optimization (PSO), with a focus on discrete PSO algorithms. Chapter 2 defines and discussed the MKP as a problem for testing discrete optimization algorithms. A well-known set of benchmark problems is listed and approaches to solve the MKP from literature are reviewed. Chapter 3 defines the FSP, another optimization problem. Besides a formal definition, the concepts of classifiers and cross-validation from the domain of machine learning are discussed. Both are important in relation to the FSP and solving the FSP using PSO. This is followed by an overview of the literature on solving the FSP.

In the second part, chapter 4 introduces the new algorithm called the set-based PSO (SBPSO). The concepts of position and velocity as sets are defined, forming the basis of the SBPSO. Also defined are the operators required to manipulate the position and velocity sets as determined by the PSO paradigm. Additional operations are defined outside of the basic PSO paradigm in order to ensure that the algorithm can work. These are then all combined in a description of SBPSO's velocity and position update equations, and the flow of the new algorithm.

Part III describes the experiments in which the SBPSO is applied to the MKP and the FSP. For both problems well-known benchmark problems and three other PSO algorithms to compare against the SBPSO are chosen. The full experimental procedure is outlined in both cases, describing the chosen swarm sizes, swarm topologies, starting and stopping conditions and number of repetitions of the experiments. For the FSP, the chosen classifiers are also listed and the approach used to tune them is described. An extensive tuning process is described and conducted to tune the PSO algorithms using different topologies and on different problem sets: two such sets for the MKP and one for the FSP. The results of the experiments on the tuned algorithms are than summarized and discussed.

Chapter 7 revisits the objectives of this dissertation and determines whether these have been met. An outline of potential future work is also given.

Finally, part V deals mainly with the detailed results of the experiments outlined in part III, split into appendix A for results on the small MKP, appendix B for results on the large MKP, and appendix C for results on the FSP. Furthermore, a description is given of all the classification datasets used in the experiments on the FSP in appendix D. Appendix E details the statistical methods employed in the various comparisons made in this thesis.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Acknowledgements

"It does not matter how slowly you go, so long as you do not stop."

Confucius

First of all, I wish to acknowledge the support and encouragement of Professor Andries Engelbrecht. He gave me the opportunity to enroll for a research Master's degree in Computer Science, even though my previous education had been in the different field of mathematics, and he has patiently corrected my work conceptually and aesthetically. Any remaining errors in this work are fully attributable to my own stubbornness.

I would also like to thank my fellow post-graduate students at the University of Pretoria, who welcomed me during my time on campus and tried to teach me things about Computer Science, superhero movies, computer games, and Afrikaans: Bennie Leonard, Julien Duhain, Nelis Franken, Christoff Enslin, Leo Langenhoven, Will van Heerden, and Koos de Beer. Additional kudos to Bennie for fixing all my Ubuntu and server problems.

A very special thanks must go to my loving wife Marjolein, who supported me in my work and also provided for our whole family during our time in South Africa. She also gave me the best gift while we were living in Pretoria: my son Jonathan Thabiso.

Joost Langeveld
September 2015
Utrecht, the Netherlands

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Contents

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

List of Figures

© University of Pretoria

List of Tables

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

List of Algorithms

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Part I

Background

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 1

Particle swarm optimization

This first chapter introduces the particle swarm optimization (PSO) paradigm. A first objective is to determine what components define a PSO algorithm, the second objective is to determine whether a functioning, generic, set-based PSO algorithm already exists. In the first section of this chapter the continuous PSO is used to present the most relevant parts of the PSO paradigm. Following this, the literature on discrete PSO methods is reviewed with the main focus on binary approaches and PSO algorithms using mathematical sets.

1.1 Introduction

This first chapter describes the general PSO paradigm, starting at its origin in the work of Kennedy and Eberhart [63], in order to fulfill two objectives. The first objective is to determine what components define a PSO algorithm. These same components will be used in the construction of a set-based PSO algorithm in chapter 4. Besides this conceptual overview, the second objective of this chapter is to determine whether a functioning, generic, set-based PSO algorithm already exists. For this a review of the appropriate literature on discrete PSO algorithms is presented.

Section 1.2 outlines the concepts that define PSO, i.e. particles, velocity, and the social and cognitive components of the velocity update. The elements that were added shortly after and now help form the canonical PSO are also introduced, i.e. the inertia weight and velocity clamping. The high-level overview of PSO ends with the introduction of the concept of swarm topologies and three specific swarm topologies are defined for later use, namely, the star, ring, and Von Neumann topologies.

Section 1.3 contains a review of PSO algorithms developed for use on DOPs. This review is divided into three parts: first the binary PSO and its variants are reviewed, followed in by an overview of existing PSO algorithms that are defined using mathematical sets. Finally, other discrete PSO algorithms are mentioned that do not fall in the two previous categories to provide a full picture.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1.2 Continuous Particle Swarm Optimization

This section describes the continuous PSO algorithm from its roots to the currently more used form with velocity clamping and inertia weight. Also, the concept of swarm topology is introduced.

1.2.1 Original Particle Swarm Optimization algorithm

Kennedy and Eberhart [63] proposed an optimization algorithm inspired by bird flocking behavior. The first PSO algorithm was developed to solve optimization problems with continuous-valued parameters. Each particle has a position $\vec{x}_i$ in the search space, and a velocity $\vec{v}_i$ indicating direction and step-size of change in the current position. Each particle keeps track of the quality of the solution to the optimization problem it represents, the best position it has visited in the past, $\vec{y}_i$, and the best position visited in the past by a particle in its neighborhood, denoted $\vec{\hat{y}}_i$.

Let i be a particle in an n-dimensional search space with velocity $\vec{v}_i = (v_i)_{j=1}^n$, position $\vec{x}_i = (x_i)_{j=1}^n$, personal best position $\vec{y}_i = (y_i)_{j=1}^n$, and neighborhood best position $\vec{\hat{y}}_i = (\hat{y}_i)_{j=1}^n$. The original velocity update equation,

$$v_{i,j}(t+1) = v_{i,j}(t) + c_1 r_{1,j}(t)\,[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)\,[\hat{y}_{i,j}(t) - x_{i,j}(t)] \tag{1.1}$$

computes the magnitude of change in the particle's position in each dimension j, where c_1 is the cognitive component weight, c_2 is the social component weight, and $\vec{r}_1$ and $\vec{r}_2$ are n-dimensional random vectors with each $r_{1.j}, r_{2,j} \sim U(0,1)$ drawn independently. The position is updated by adding the updated velocity to the current position:

$$x_{i,j}(t+1) \quad = \quad x_{i,j}(t) + v_{i,j}(t+1) \tag{1.2}$$

1.2.2 Additions to original Particle Swarm Optimization algorithm

To improve the performance of the algorithm and to better control the balance between exploration of new areas of the search space and exploitation of promising areas, various additions have been proposed. A first addition was by Eberhart *et al.* [32], who proposed *velocity clamping* which restricts the velocity to a predetermined maximum in each dimension. After the velocity has been updated, but before the position update, the velocity clamping,

$$v_{i,j}(t+1) \quad = \quad \min\{\max\{v_{i,j}(t+1), V_{\min,j}\}, V_{\max,j}\} \tag{1.3}$$

is applied, where $V_{\min,j}$ and $V_{\max,j}$ with $V_{\min,j} < V_{\max,j}$ denote the minimum and maximum velocity in a single dimension j.

An addition proposed by Shi and Eberhart [126] was a scalar, ω, called the *inertia weight*, which determines the acceleration or deceleration in the current direction. The inertia weight scales the component indicating the particle's current velocity, $v_{i,j}(t)$, in equation (1.1), resulting in an alternative velocity update equation,

$$v_{i,j}(t+1) \quad = \quad \omega\, v_{i,j}(t) + c_1 r_{1,j}(t)\,[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)\,[\hat{y}_{i,j}(t) - x_{i,j}(t)] \tag{1.4}$$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A higher value (usually chosen ≤ 1) for ω causes particles to change direction more slowly and thus to explore further along the currect direction of movement. A lower value for ω causes particles to be more strongly attracted to the personal best and neighbourhood best positions, which leads to better exploitation.

Algorithm 1 describes the flow of the PSO algorithm for a maximization problem with objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. A similar definition is easily obtained for a minimization problem.

Algorithm 1: Continuous PSO for maximization problems

```
Set $N$ equal to the number of particles in the swarm;
for $i = 1, \ldots, N$ do
    Initialize $\vec{x}_i$ uniformly random over the search space ;
    Initialize $\vec{v}_i = \vec{0}$ ;
    Calculate $f(\vec{x}_i)$ ;
    Initialize $f(\vec{y}_i) = -\infty$ ;
    Initialize $f(\vec{\hat{y}}_i) = -\infty$ ;
end
while stopping condition is false do
    for $i = 1, \ldots, N$ do
        // set the personal best position ;
        if $f(\vec{x}_i) > f(\vec{y}_i)$ then
            $\vec{y}_i = \vec{x}_i$;
        end
        // set the neighborhood best position ;
        for all neighbors $l$ of particle $i$ do
            if $(f(\vec{y}_i) > f(\vec{\hat{y}}_l)$ then
                $\vec{\hat{y}}_l = \vec{y}_i$;
            end
        end
    end
    for $i = 1, \ldots, N$ do
        Update $\vec{v}_i$ according to equation (1.4);
        Update $\vec{x}_i$ according to equation (1.2);
        Calculate solution quality $f(\vec{x}_i)$;
    end
end
```

1.2.3 Swarm topologies

One of the strengths of PSO is the flow of information through the swarm due to the interaction of the particles. Particles with a good objective function value attract other particles, hopefully to good areas of the search space. Particles that have found a good solution attract particles for which they are the best neighbor. If two particles i and j are not connected (not in each other's neighborhood), then they can not directly attract each other. If a common neighbor k is attracted to a good solution i and becomes a good solution itself, such that it is the best solution in the neighborhood of j, then j can be said to be

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

indirectly influenced by *i*. For each particle, the social structure, called the swarm topology, determines which particles it can be attracted to.

Kennedy and Eberhart [63] proposed two possible social structures for the particle neighborhoods, and called the two resulting algorithms the global best (*gbest*) PSO and local best (*lbest*) PSO. The gbest PSO uses a *star* topology, while the lbest PSO uses a *ring* topology. The ring topology is a loosely connected topology, while the star topology is one where each particle is directly connected to all other particles in the swarm. A study of the impact of the swarm topology was done by Kennedy and Mendes [64], considering various topologies, including random, star, Von Neumann and ring topologies. Kennedy and Mendes [64] suggested that the *Von Neumann* topology, which has an intermediate level of connectivity, can be a good choice for a particle swarm.

Figure 1.1 illustrates three of the topologies investigated by Kennedy and Mendes [64]: the star topology (figure 1.1(a)), the ring topology with neighborhood size four (figure 1.1(b)), and the Von Neumann topology (figure 1.1(c)). The ring topology with neighborhood size four means that the particles lie in a ring, with each particle connected to its two nearest neighbors on each side, so to four other particles in total.

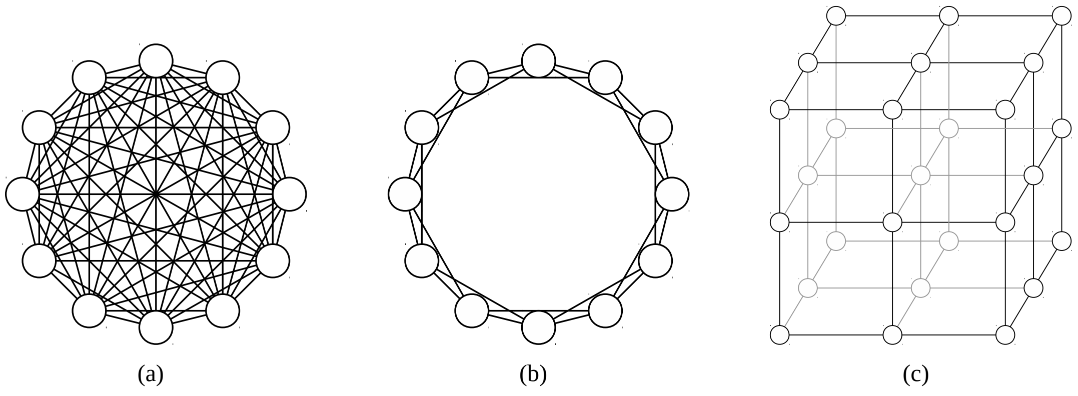


Figure 1.1: Swarm topologies: (a) star topology, (b) ring-4 topology, and (c) Von Neumann topology.

1.3 Discrete Particle Swarm Optimization

This section reviews discrete PSO algorithms developed to solve DOPs. First the binary PSO and its variants are reviewed in section 1.3.1, followed in section 1.3.2 by an overview of existing PSO algorithms that are defined using mathematical sets. Section 1.3.3 mentions other discrete PSO algorithms that do not fall in the two previous categories to provide a full picture.

1.3.1 Binary Particle Swarm Optimization approaches

A number of PSO algorithms that can be applied to DOPs closely follow the binary PSO algorithm first proposed by Kennedy and Eberhart [62]. The variants reviewed here are the (original) binary PSO, the modified binary PSO, the probability binary PSO, and the catfish PSO.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1.3.1.1 Binary Particle Swarm Optimization

Kennedy and Eberhart [62] were the first to define a discrete version of the PSO algorithm, referred to as the binary PSO (BPSO). In this algorithm the particle positions are binary strings, while the velocities exist in continuous space. Velocities are mapped to a scalar value between 0 and 1 using a sigmoidal transformation function, S. This scalar value is interpreted as the probability that the corresponding part of the binary position string is bit 1 or bit 0. The velocity update equation of the BPSO algorithm is the same as equation (1.4). Using the transformation function,

$$S\big(v_{i,j}(t+1)\big) = \frac{1}{1+e^{-v_{i,j}(t+1)}} \tag{1.5}$$

the position update becomes

$$x_{i,j}(t+1) = \begin{cases} 1 & \text{if } r_{3,j} < S\big(v_{i,j}(t+1)\big) \\ 0 & \text{otherwise} \end{cases} \tag{1.6}$$

where $r_{3,j}$ is an independent random variable, uniformly distributed on $(0,1)$. Eberhart *et al.* [30] proposed to use velocity clamping as defined in equation (1.3) in BPSO to prevent saturation of the sigmoid function.

Many variants of the BPSO algorithm have been proposed, e.g., Khanesar *et al.* [66] defined a BPSO that has separate velocity terms depending on whether a bit in the current position vector $\vec{x}$ is 0 or 1, Gao *et al.* [39] removed the randomness from the position update step, and Yang *et al.* [153] proposed the quantum BPSO by introducing the idea of a superposition of states.

1.3.1.2 Modified Binary Particle Swarm Optimization

Shen *et al.* [125] introduced a new PSO algorithm called Modified PSO, which here will be called the modified binary PSO (MBPSO). They investigated quantitative structure-activity relationships (QSAR), the process by which a chemical structure is quantitatively correlated with a well defined process, such as biological activity or chemical reactivity. The new PSO algorithm was applied to the problem of variable selection in multiple linear regression (MLR) and partial least squares (PLS), which in turn are used in many QSAR models.

The velocity update equation of MBPSO is the same as equation (1.4) used for continuous PSO with inertia weight. For the position update in MBPSO, each bit $x_{i,j}(t)$ in the position vector $\vec{x}_i(t)$ is updated according to:

$$x_{i,j}(t+1) = \begin{cases} x_{i,j}(t) & \text{if } 0 \le v_{i,j}(t+1) \le p_{\text{stat}} \\ y_{i,j}(t) & \text{if } p_{\text{stat}} < v_{i,j}(t+1) \le 0.5(1+p_{\text{stat}}) \\ \hat{y}_{i,j}(t) & \text{if } 0.5(1+p_{\text{stat}}) < v_{i,j}(t+1) \le 1 \end{cases} \tag{1.7}$$

where p_{stat} is a parameter in $(0,1)$ called the *static probability*.

Shen *et al.* [125] stated that after the velocity and position updates have been applied, a fraction of particles “are forced to fly randomly not following the two best particles”. This statement has been interpreted as a random re-initialization of both the velocity and the position of a percentage of the swarm

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

at each iteration, similar to Ma *et al.* [91]. The fraction of particles that is re-initialized at each iteration is denoted by p_{reset}.

1.3.1.3 Probability Binary Particle Swarm Optimization

Wang *et al.* [146] proposed a binary variant of PSO called the probability binary PSO (PBPSO) and applied this to the multidimensional knapsack problem. Both the velocity and the position update equations are the same as for PSO in equations (1.4) and (1.2). The continuous position labeled $\vec{x}'$ (which is also called the pseudo-probability) is transformed to a binary position vector $\vec{x}$ using a linear transformation $L(.)$:

$$L\left(x'_{i,j}(t+1)\right) = \frac{x'_{i,j}(t+1) - R_{\min}}{R_{\max} - R_{\min}} \tag{1.8}$$

$$x_{i,j}(t+1) = \begin{cases} 1 & \text{if } r_j < L\left(x'_{i,j}(t+1)\right) \\ 0 & \text{otherwise} \end{cases} \tag{1.9}$$

where r_j is a uniform random variable chosen from $(0,1)$. The parameters $R_{\min}$ and $R_{\max}$ define the linear transformation and are usually chosen such that $R_{\max} > 0$ and $R_{\min} = -R_{\max}$.

The PBPSO algorithm was extended in [96] to also include a mutation operator. For each particle, after the velocity and position of each particle have been updated, the mutation operator is applied. Each bit then has a probability $p_{\text{mut}} \in [0,1]$ of mutating:

$$x_{i,j}(t+1) = \begin{cases} 1 - x_{i,j}(t+1) & \text{if } r_j < p_{\text{mut}} \\ x_{i,j}(t+1) & \text{otherwise} \end{cases} \tag{1.10}$$

where r_j again is a uniform random variable chosen from $(0,1)$. This second version of PBPSO is used in the experiments reported in this thesis.

Note that the mutation operator has a direct effect on the position only for one iteration, as the velocity $\vec{v}$ and the pseudo-probability $\vec{x}'$ are left unchanged by the mutation. At the next iteration, $\vec{x}'$ is used to rebuild the binary position vector $\vec{x}$ according to equation (1.9), and no direct impact of the mutation made during the previous iteration remains in $\vec{x}$. An indirect impact of the mutation is in the update of the velocity $\vec{v}$ according to equation (1.4), which uses the mutated position, and possibly through the personal best position $\vec{y}$, if the mutated position $\vec{x}$ caused the personal best to be updated.

1.3.1.4 Catfish Binary Particle Swarm Optimization

Chuang *et al.* [20] introduced the catfish effect for PSO, with the new method called the catfish PSO (CFPSO). The effect is based on the practice by Norwegian fishermen to introduce catfish predators into tanks in which captured sardines are held on their fishing vessels. This makes the sardines move and keeps them alive before they are delivered to shore. The catfish effect for PSO is that, if the best found fitness in the swarm stays constant for a given number of iterations, catfish particles are introduced at extreme positions of the search space to replace the worst performing swarm particles. Continuous CFPSO has two additional parameters to the normal parameters for the PSO algorithm: the number of iterations the best found fitness needs to stay constant for the catfish effect to be activated, denoted

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

N_{constant}, and the proportion of worst particles in the swarm to be replaced by catfish particles, denoted p_{replace}.

The CFPSO algorithm closely follows the continuous PSO algorithm with velocity update equation (1.4) and position update equation (1.2). During each iteration, after the positions of all particles has been updated, a check is made to see if the global best fitness has stayed constant for N_{constant} iterations. If this is the case, the catfish effect is activated and the particles in the swarm are ordered based on their current fitness. The $p_{\text{replace}}\,N$ worst particles are then removed from the swarm, and an equal number of catfish particles are added to the swarm in their place. Each catfish particle's position is constructed by setting the value for each dimension separately to either the minimum value or maximum value for that dimension randomly. So for a two-dimensional search space labeled (x,y), a catfish particle's position is set to one of the following options: $(\text{min}_x, \text{min}_y), (\text{min}_x, \text{max}_y), (\text{max}_x, \text{min}_y), (\text{max}_x, \text{max}_y)$. Chuang *et al.* [20] did not mention how the velocity of each catfish particle is set. The implementation of CF-PSO that is used in the experiments reported in this thesis set the initial velocity of the catfish particle according to the method used at the swarm's initialization.

Chuang *et al.* [21] adjusted the algorithm and applied it to feature selection. Instead of continuous PSO, the basic algorithm followed is the Binary PSO outlined in section 1.3.1.1. This binary algorithm is called Catfish Binary PSO (CFBPSO). The positions of any catfish particles that are added to the swarm are chosen randomly as either zero ($\{0\}^d$) or one ($\{1\}^d$), where d is the number of dimensions of the search space. Note that this is different from the original CFPSO algorithm such that for CFBPSO particles are reset to one of only two extremes of the search space.

The full Catfish reset algorithm for the CFBPSO is detailed in algorithm 2. The CFPSO and CFBPSO algorithms can be used with any swarm topology, but the Catfish reset algorithm requires that track is kept of the best fitness of the entire swarm.

Algorithm 2: Catfish reset algorithm for Catfish Binary PSO

```
if $f(\vec{y})$ is unchanged for $N_{constant}$ iterations then
    Sort the $N$ particles $x_i$ from best to based on their current fitness $f(\vec{x}_i)$ ;
    for $N \times p_{replace}$ particles $\vec{x}_i$ with the worst fitness in the swarm do
        if $r_i < 0.5$ (where $r_i$ is an uniform random number drawn from $(0,1)$ ) then
            ;
            $\vec{x}_i = \{0\}^d$
        end
        else $\vec{x}_i = \{1\}^d$
    end
end
```

1.3.2 Discrete Particle Swarm Optimization using sets

This section describes and critiques existing discrete PSO algorithms using sets. These existing algorithms are important in light of the stated objective of this thesis to construct a functional, generic set-based PSO algorithm. The goal is to determine whether the existing algorithms already fulfill this objective, and if not, in what manner the existing algorithms are lacking.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The algorithm proposed by Correa *et al.* [24] for attribute selection and the related algorithm by Bock and Hettenhausen [10] for ontology alignment both have set-like characteristics, but both contain problem specific elements. Especially, the concept of a *personal likelihood* that requires each element in a particle position to have its own partial objective function value, prevents these algorithms from being applied to many discrete optimization problems, including the MKP.

Veenhuis [143] proposed a generic, set-based definition of a PSO algorithm. Velocities and positions in this algorithm are both defined as sets. However, the chosen update equations lead the velocities and positions to always increase in size, an effect called *set bloating*. To counter this, a reduction operator with a relatively complex clustering mechanism was introduced. This clustering mechanism requires a function that defines the distance between any two set elements, while a general mathematical set does not support the concept of distance. Veenhuis [143] has chosen a problem specific distance function, meaning that the algorithm is no longer truly generic, and in its current form is not applicable to discrete problems such as the MKP.

Neethling and Engelbrecht [102] proposed the set-based algorithm called SetPSO and applied it to RNA structure prediction. The problem is defined as finding the correct stems (bindings of base pairs) in the RNA structure from the set of all possible stems. Particle positions are defined as sets of stems. In the position update, three probabilities help determine which elements are added and which elements are removed from the position. Although generically applicable, recent work [79] has shown that SetPSO performs less well on the MKP than other PSO methods: SetPSO was outperformed by a large margin by the SBPSO (see chapter 4) and BPSO (see section 1.3.1.1) algorithms. Hence, the SetPSO in its current form can not be considered to be truly functioning on DOPs in general.

Chen *et al.* [17] proposed a generic set-based PSO method called S-PSO that can be used to adjust a continuous PSO algorithm to a discrete one. S-PSO was applied to the TSP and the MKP. The candidate solution represented by a particle position is called a set, but has a fixed size where for each “dimension” of the set an element is chosen from a set of available elements. Thus the position can not be called a true set. Velocity is defined as a *set with possibilities*, which grows in size as the algorithm runs. Positions are rebuilt at each iteration using a constructive process that may include heuristic operators. Wu *et al.* [149] applied a variant of S-PSO based on (continuous) constriction PSO to the problem of cloud computing workflow scheduling.

Khan and Engelbrecht [65] proposed an algorithm called fuzzy PSO (FPSO) to optimize the topology design of distributed local area networks (DLANs). The term fuzzy in FPSO refers to the fuzzy aggregation operator, the *unified And-Or operator*, that is used to aggregate the multiple objectives in the DLAN topology design problem into a single objective function. The particle position is defined as a set of links between nodes in the network. The number of links in the position is exactly $N-1$, where N is the number of nodes in the network. The particle velocity is defined as a set of *link exchange operations*, which removes a single link in the position and replaces it by another. Because the size of the position is fixed, the algorithm is not generally applicable to discrete problems such as the MKP.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1.3.3 Other discrete Particle Swarm Optimization algorithms

This section reviews a number of discrete PSO algorithms that do not follow the binary PSO paradigm and which also are not set-based approaches. The discrete approaches reviewed in this section are the angle modulated PSO, PSOs based on fuzzy logic, rank-based PSO algorithms, and PSO algorithms that redefine the meaning of particle positions, velocities, and arithmetic operators.

1.3.3.1 Angle Modulation Particle Swarm Optimization

Pampara *et al.* [110] developed a different approach to converting the continuous-valued velocity of PSO to a binary string by applying the concept of *angle modulation*. Angle modulation PSO starts with a swarm of particles in a continuous four dimensional space, and uses a continuous PSO algorithm to update the particle velocities and positions. For each particle, the four position components are used as parameters for a trigonometric function, and this function is sampled n times to generate an n-dimensional bit-string. If the function produces a positive value, then bit 1 is recorded, otherwise bit 0 is recorded.

1.3.3.2 Fuzzy Binary Particle Swarm Optimization

Fuzzy logic has also been used to construct discrete PSO algorithms. Where the particle position in binary PSO is a binary vector with a "crisp" separation of bits into 0 and 1, fuzzy binary PSO instead has a position vector with fuzzy bits. It uses a membership function μ to indicate a truth value in $[0,1]$ for the degree to which each fuzzy bit has value 1. The fuzzy PSO algorithm works in continuous space and a separate mechanism called *defuzzification* is used to convert the fuzzy particle position into a binary vector. The first published article on using a fuzzy approach to the discrete PSO is by [127]. Pang *et al.* [111] and Shen *et al.* [124] provided refinements to the fuzzy method and applied it to the traveling salesman problem (TSP). Du *et al.* [28] applied their fuzzy PSO to the shape matching problem, while Abraham *et al.* [1], Liu and Abraham [87], Liu *et al.* [88] applied fuzzy discrete PSO algorithms to job scheduling problems and to the quadratic assignment problem.

1.3.3.3 Rank ordering in discrete Particle Swarm Optimization

The concept of rank ordering has been used to construct discrete PSO algorithms. These algorithms transform a continuous-valued position to a discrete-valued position by determining the relative order (rank) of the continuous values in a particle's position. Tasgetiren *et al.* [135] introduced such a modification to the continuous PSO algorithm and applied it to scheduling problems, exemplified by the single machine total weighted tardiness problem. Solutions for such scheduling problems are sequences or permutations of tasks that indicate the order in which the tasks are performed. A candidate solution is represented as a sequence $S_i = [s_{i,1}, \ldots, s_{i,n}]$ of the numbers $1, \ldots, n$, where each $s_{i,k}$ is unique and denotes one of the n tasks to be scheduled.

The particle velocities and positions are updated according to equations (1.4) and (1.2) respectively. Each position, $\vec{x}_i$, is then translated to a sequence S_i using the *smallest position value* (SPV) rule. The SPV rule takes the position component, $x_{i,j}$, with the smallest value in $\vec{x}_i$, and sets $s_{i,1}$ equal to j. Then

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

it takes the next smallest position component, $x_{i,k}$, and sets $s_{i,2} = k$. This process continues until the sequence S_i has been filled.

Similar algorithms have been proposed by Pang *et al.* [112], who used the *greater value priority* to transform the continuous-valued position $\vec{x}_i$ to a sequence S_i and applied the resulting PSO algorithm to the TSP. Liu *et al.* [86] used an almost identical approach called *rank order value* and applied this method to the flow shop scheduling problem (FSSP).

1.3.3.4 Redefined Particle Swarm Optimization operators

Clerc [23] formulated a discrete PSO algorithm by redefining the particles, velocities and operators used in PSO. A general mathematical specification is given as well as an implementation that is then applied to the TSP. A particle position is defined as a sequence of $N+1$ arcs between nodes, where N is the number of nodes in the TSP. A velocity is defined as a list of *exchange operations* (i, j), where nodes i and j in a position are swapped. Special operations are also defined for subtraction of two positions, the addition of two velocities, and the multiplication of a scalar and a velocity. These new operators are then used in a formulation of the velocity update equation in the discrete PSO that is very similar to equation (1.4) used in continuous PSO.

Wang *et al.* [145], Zhang *et al.* [156], and Zhong *et al.* [158] proposed similar approaches to modifying the PSO operators and each applied the resulting PSO to the TSP. García *et al.* [40] applied an adapted PSO algorithm to the response time variability problem, where the particle velocity is defined as an ordered list of transformations called *movements*. Benameur *et al.* [7] proposed a similar discrete PSO and applied it to the frequency assignment problem. Chandrasekaran *et al.* [16] applied a discrete PSO with redefined operators to the FSSP, where the velocity is a set of transpositions with ordering values. The transpositions contained in the velocity are applied to the position in the order of high to low ordering values.

1.4 Conclusions

A first objective outlined for this chapter was to determine the components that define a PSO algorithm, so that these components can later be used to construct a set-based PSO algorithm in chapter 4. To do this, the canonical version of the continuous PSO algorithm was described and its constituent parts considered in turn. The components of a PSO algorithm are identified as:

- A swarm of particles which each have a position and a velocity, whereby the position is updated by adding the velocity to the current position. The algorithm's power comes form the simple but effective way the velocity is updated.
- The velocity update equation contains three components that together determine how a particle's velocity changes:
  - a cognitive component that describes the attraction of the particle to the best position in the search space found by that particle previously,

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- a social component that describes the attraction of the particle to the best position in the search space found by any particle in its neighborhood, and
- an inertia component that causes the velocity to retain part of the direction it currently has.

- The impact of the swarm topology is highlighted and a trio of possible topologies were introduced.

Together with a description of the algorithm's flow, this provides the framework to construct a set-based algorithm in chapter 4 that is also a PSO algorithm.

The second objective of this chapter was to determine whether a functioning, generic, set-based PSO algorithm already exists. For this a review of the appropriate literature was presented. It can be concluded that such an algorithm is not yet available, as the reviewed algorithms all lack at least one of the attributes of (i) functioning such that its use leads to good results in solving DOPs, (ii) being generically applicable to all DOP and thus not contain any problem specific features like heuristics or operators specifically defined for the problem domain, or (iii) not being truly set based:

- The SetPSO proposed by Neethling and Engelbrecht [102] was shown in [79] to perform badly on the MKP and hence it does not fulfill the criterion of being an algorithm that is truly functioning on this DOP.
- The algorithms proposed by Correa *et al.* [24], Bock and Hettenhausen [10] and Veenhuis [143] are not generic but each contains problem specific elements.
- In the algorithms proposed by Chen *et al.* [17], Wu *et al.* [149], and Khan and Engelbrecht [65] the candidate solution is represented by a particle position with a fixed size and which thus can not be called a true set.

A functioning, generic set-based PSO algorithm is thus shown to not yet exist. Hence the generally applicable, functioning set-based PSO algorithm that will be detailed in chapter 4 is a real contribution to the domain of discrete PSO algorithms.

The remaining two chapters in part I formally introduce two well-known DOPs: the multidimensional knapsack problem in chapter 2 and the feature selection problem in chapter 3. These two problems will be used in part III to prove that the new algorithm proposed in chapter 4 is able to solve such DOPs and to compare its performance against existing discrete PSO algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 2

Multidimensional knapsack problem

This chapter describes the multidimensional knapsack problem (MKP), an optimization problem that has been used to test the performance of discrete optimization algorithms. This is one of the two sets of problems that will be used in this thesis to test such algorithms, the other being the feature selection problem that is described in chapter 3. This chapter has two objectives, the first of which is to formally define the MKP and list its main characteristics. The second and more important objective is to give a brief overview of the various methods that have been employed to solve the MKP, with a focus on approaches using PSO, and thereby convince the reader that the MKP is a valid problem on which to test discrete PSO algorithms. The sections of this chapter follow these two objectives, while also introducing a commonly used set of benchmark problems and the variables that help describe these problems.

2.1 Introduction

The previous chapter dealt with the PSO algorithm and laid a strong emphasis on PSO variants that have been used to solve DOPs. This gave a background against which a new discrete PSO algorithm will be introduced in chapter 4. In order to determine the usefulness of this new algorithm and whether it forms a relevant contribution to the field of swarm intelligence, the algorithm will need to be tested on actual DOPs. This chapter introduces one such problem, the MKP. First mentioned in 1955 by Lorie and Savage [89] in the form of the problem of capital budgeting, the MKP has become the catch-all name for any zero-one integer problem with non-negative coefficients, as such problems can all be re-formulated as MKPs.

The first and relatively simple objective of this chapter is to introduce the MKP and list its main characteristics. By exhibiting the exact definition of the problem, all further analysis is set on a solid and unambiguous base. Besides the exact equations that define the problem, some further features are sought that can be useful in categorizing MKPs. These categories not only deal with whether a particular set of problem is easy or hard to solve, but can also be used to gain insight on what parameter values for a PSO algorithm can be expected to work well on a specific problem.

The second and more important objective of this chapter is to give a brief overview of the various methods that have been employed to solve the MKP, with a focus on approaches using PSO. This

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

overview is meant to convince the reader of the non-trivial nature of the problem and show that a wide range of approaches including PSO have been employed in solving the MKP. So convinced, the MKP can be seen as a valid choice test bed of the new PSO algorithm introduced in chapter 4 and other discrete PSO algorithms it will be compared to. A well known test-bed ensures that no simple exploits exist, and proves that the MKP is sufficiently hard to test the quality of the optimization algorithms. The introduction of work done on MKPs using discrete PSO algorithms will also help to determine later in part III which PSO algorithms are included in the comparison to the newly developed set-based algorithm.

This chapter begins with a formal definition of the MKP in section 2.2, followed in section 2.3 with a description of a number of benchmark problems that recent studies into the MKP frequently use. The introduction of these benchmark problems allows to naturally mention some of the features of MKP that can help determine whether a particular MKP is simple or hard to solve. Then a review is made of previous work on solving the MKP in section 2.4, with a more detailed view on approaches to solve the MKP using PSO outlined in section 2.4.2. With the specific view to help determine which PSO algorithms are to be included in a comparison of PSO algorithms on the MKP in chapter 5, section 2.4.2.3 reviews and analyzes in detail the work of Wang *et al.* [146] who have already made such a comparison.

2.2 Definition of the Multidimensional Knapsack Problem

The MKP, also called the multidimensional zero-one knapsack or rucksack problem, is a discrete optimization problem. The aim of the problem is to maximize the total value of all items to be put in a knapsack, i.e.

$$\max \sum_{i=1}^{n} v_i x_i \tag{2.1}$$

subject to the zero-one constraints

$$x_i \in \{0,1\},\ \forall i \in \{1,\ldots,n\} \tag{2.2}$$

and weight constraints

$$\sum_{i=1}^{n} w_{i,j} x_i \leq C_j,\ \forall j \in \{1,\ldots,m\} \tag{2.3}$$

There are n items in total, each with value v_i. The binary variable x_i indicates whether the item i is present in the knapsack or not. These in-or-out constraints make the problem a *discrete* optimization problem and increase its complexity above, for example, relaxations of the problem where the x_i are only limited to lie within the range $[0,1]$.

The problem also contains m weight constraints. Items have a separate and distinct weight for each weight constraint, such that the weight item i has with regards to weight constraint j can be denoted $w_{i,j}$. The weight constraint itself is defined such that $\sum_i w_{i,j}\, x_i$, the total weight for constraint j, may not exceed the capacity C_j. In the remainder of this thesis, all mention of the MKP's constraints refer to the *weight* constraints, as the zero-one constraints are considered part of the definition of the MKP as a class of discrete optimization problems.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A well-formulated multidimensional knapsack problem also adheres to the value constraints,

$$v_i > 0, \quad \forall i \in \{1, \dots, n\} \tag{2.4}$$

and constraints on the total weight

$$w_{i,j} \leq C_j < \sum_{i=1}^{n} w_{i,j}, \quad \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\} \tag{2.5}$$

The value constraint means that each item has a positive value and thus ensures that no items can be simply disregarded as having no contribution to the value to be maximized. The constraints on the total weight have two implications. The first implication is that each item can indeed be included in the knapsack. This means that for each weight constraint in equation (2.3) the capacity, C_j, exceeds the j-weight of each item i denoted by $w_{i,j}$. The second implication is that each weight constraint indeed constrains the problem and can not be ignored, by virtue of the fact that the capacity C_j is less than the sum of j-weights of all items.

The MKP has been proven to be an NP-complete optimization problem [43]. This means that no method of solving the problem can exist that is guaranteed to find the solution within polynomial time $\zeta\, x^N$ for some finite order N where x is the number of items in the MKP and ζ is a constant. Note furthermore that any zero-one integer problem with non-negative coefficients can be formulated as a MKP. The first mention of such problems was with regards to capital budgeting [89].

2.3 Benchmark problems

Recent studies into the MKP frequently use the same benchmark problems to compare the performance of algorithms. Two sets of problems are distinguished, which are here denoted as the large MKP and the small MKP. The large MKPs were constructed and introduced by Chu and Beasley [18]. The small MKPs form a collection of 55 problems that have been mentioned in literature prior to the paper by Chu and Beasley [18]. Both sets of problems are available on-line at the Operations Research Library (ORLib) at `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`.

For these small MKPs, the number of items n ranges from 6 to 90 and the number of constraints m ranges from 2 to 30. The optimal solution is known for all small MKPs. The small MKPs are listed in table 5.1 in section 5.2.1.1. Note that for each problem, the problem name reflects the filename from the ORLib source the problem comes from, plus a number indicating which problem from that file it refers to. For example, “mknap2-3” is the third problem found in the file *mknap2.txt*. Table 5.1 also includes the number of times n and the number of constraints m for each of the 55 small MKPs.

Chu and Beasley [18] have generated the large MKPs randomly. Note that for each such problem generated, the number of items, n, the number of constraints, m, and another variable called tightness ratio, r, which is defined below, were determined by the problem specification before the weights and capacity were generated randomly. The latter began by randomly choosing the weights $w_{i,j}$ and values v_i. The capacity constraint variables C_j in equation (2.3) were determined by the random weights and

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the chosen tightness ratio r, according to the formula

$$C_j \quad = \quad r\sum_{i=1}^{m} w_{i,j}, \quad \forall j \in \{1,\ldots,m\} \tag{2.6}$$

The large MKP consists of a collection of 270 MKPs with number of items $n = 100, 250$, or 500, number of constraints $m = 5, 10$, or 30, and tightness ratio $0.25, 0.50$, or 0.75. The three choices for each of the three parameters, n, m, and r, yield 27 different problem specifications. For each problem specification, 10 problem instances are included in the problem set.

The large MKPs are labeled according to the filename they are found in combined with the number of the problem in that file. There are nine files named “mknapcb1” to “mknapcb9” which each contains 30 problems with the same parameters n and m. The 30 problems are 10 random instances each with tightness ratio $r = 0.25$ for problems 1 to 10, $r = 0.50$ for problems 11 to 20 and $r = 0.75$ for problems 21 to 30. An example label is thus “mknapcb3-24”, which is the 24th problem found in file “mknapcb3”.

In general, these three problem parameters, n, m, and r, have the following effects on the MKP search space:

- a larger number of items, n, increases the search space and hence makes the problem of finding the optimum harder,
- a larger number of constraints, m, makes the feasible part of the search space smaller, but no simple relationship exists between the size of the feasible search space and the difficulty in solving the MKP, and
- a larger tightness ratio, r, means that the weight constraints are *less* restrictive and that the feasible part of the search space becomes larger.

The optimal solution is known for some but not all large MKPs. In order to be able to compare the quality of solutions across problems for the large MKPs, Chu and Beasley [18] obtained an upper bound for the objective function value by solving the linear programming (LP) relaxation of the large MKPs. The LP relaxation of the problem changes the zero-one constraint in equation (2.2) on x_i from an integer constraint to a continuous constraints:

$$x_i \in [0,1],\ \forall i \in \{1,\ldots,n\} \tag{2.7}$$

thereby making the problem easier to solve and no longer NP-hard. The LP relaxed version of the MKP can be efficiently solved using standard LP solvers [18]. The bounds found by solving the relaxed problem are available at the ORLib website.

2.4 Literature on solving the Multidimensional Knapsack Problem

This section describes some of the main approaches from literature used to solve the MKP. The descriptions are split between approaches involving PSO and other methods, where the latter are mentioned first. For methods used to solve the MKP using PSO, a further distinction is made between approaches that use repair operators and those using penalty functions.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

2.4.1 Approaches not involving Particle Swarm Optimization

An overview of exact methods and heuristics used to solve the MKP can be found in [18, 67] and a recent update in [115]. This section mentions work that uses exact approaches, primal heuristics and approximations, bound based heuristics, and work on worst-case analysis, before moving on to population based methods.

Early exact approaches to solve the MKP used branch-and-bound methods combined with various relaxations of the integer constraints of the MKP, including linear programming (LP), Lagrangian, surrogate and composite relaxations [42]. Dynamic programming and iterative schemes for LP were developed, but only limited success was reported [104, 132].

As exact methods are impractical for larger problems given the NP-hardness of the MKP, heuristic methods were then employed. Early heuristic approaches used primal heuristics starting with an empty solution and building this up using greedy methods with different functions to weigh the utility of each item to be added [90]. Kellerer [61] proposed an approximation scheme based on a generalized greedy algorithm and proved it could approximate a solution for the MKP in linear time.

Bound based heuristics use an upper bound on the optimal solution to the MKP. Similar relaxation techniques as used in the exact approaches were utilized: Lagrangian, LP, surrogate, and composite relaxations. Chu and Beasley [18] and Khuri *et al.* [67] employed a LP relaxation technique. Vasquez and Hao [141] implemented a tabu search heuristic which they combined with LP to form an efficient way to solve the MKP.

Another approach used is to investigate the worst-case or probabilistic performance of different models on a given MKP. Averbakh [4] used this approach to develop a fast statistically efficient approximate algorithm with linear running time complexity for problems with random coefficients.

Population based optimization algorithms have also been applied to the MKP including genetic algorithms (GA) [18, 67], ant colony optimization (ACO) [73], cuckoo search [82], and PSO.

The current state of the art consists of hybrid algorithms that include MKP specific heuristics. Smaller MKP benchmark problem are consistently and perfectly solved be all of these algorithms. There is, however, no single algorithm that performs best on all of the larger benchmark problems, like those introduced by Chu and Beasley [18]. The algorithms that are considered state-of-the-art for the MKP are:

- The tabu-search algorithms embedding effective preprocessing by Vasquez and Hao [141] and Vasquez and Vimont [142],
- a heuristic combining two specific constraint propagations by Vimont *et al.* [144],
- an algorithm that solves the MKP by solving a series of smaller sub-problems generated by subsequent linear programming relaxations of the original MKP by Hanafi and Wilbaut [51], and
- an exact method based on a multi-level search strategy by Boussier *et al.* [13].

The next section deals with PSO based approaches to solving the MKP. A number of issues that are relevant for any population based approach to solving the MKP are addressed in this next section, dealing with the use of repair operators and penalty functions.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

2.4.2 Approaches involving Particle Swarm Optimization

PSO has proven to be a popular choice of population based approaches to the MKP, primarily due to its simplicity. When applying PSO or other population based methods to the MKP, it is possible that particles stray into infeasible parts of the solution space: the candidate solution represented by a particle does not satisfy all the constraints posed by the MKP. Two different mechanisms have been used to address this problem: repair operators and penalty functions. The overview of previous work on the MKP using PSO presented below is grouped according to these two mechanisms.

2.4.2.1 Repair operators

A first method used to deal with a swarm of particles in a PSO algorithm leaving the feasible part of the MKP's solution space is to use repair operators. These operators ensure that once a particle leaves the feasible part of the solution space it is immediately "repaired" and converted to a feasible solution.

Kong and Tian [72] used the binary PSO from section 1.3.1.1, and constructed two variants: PSO-R which included a heuristic repair operator to avoid infeasible solutions, and PSO-P which applied a penalty function. The repair operator process consists of first determining the so-called pseudo-utility of each item before the PSO algorithm itself is run. The pseudo-utility draws closely on information specific to the MKP domain. Then, if during the run of the PSO-R algorithm a particle leaves the space of feasible solutions, it is moved back into feasible space in two phases. In the first phase labeled "DROP", the items that are included in the particle's position are listed in order of increasing pseudo-utility and removed one-by-one until no constraints are violated. Then in the second "ADD" phase, items are investigated in order of *decreasing* pseudo-utility and added to the particle's position as long as no constraints are violated. The two algorithms PSO-P and PSO-R were compared using the quality of the solution found on 7 small MKPs and 10 large MKPs. PSO-R outperformed the PSO-P algorithm on all but one simple problem, on which both algorithms performed equally well. The PSO-P algorithm is described below in section 2.4.2.2. Kong and Tian [72] noted that the repair operator plays a critical role in quickly finding good solutions.

Labed *et al.* [78] developed the Modified Hybrid Particle Swarm Optimization (MHPSO) algorithm which combines PSO with two different particle repair operators and a crossover operator from GAs. The first repair operator called the Particle Repair Algorithm (PRA) randomly choses items to remove from an infeasible candidate solution until it no longer violates any constraint. A second operator called Check and Repair Operator calculates a measure called profit density to determine which item to remove from an infeasible solution. Which repair operator is used in a specific iteration of the algorithm is not made clear.

The algorithm was tested on 25 small and 10 large MKPs from the benchmark problems mentioned in sections 2.3 and 5.2.1. It is not possible to determine how well the algorithm performed, as no details were given for the experimental set-up, nor was the algorithm compared to any other known algorithm.

Gherboudj *et al.* [44] proposed the New Hybrid Binary PSO algorithm (NHBPSO) which combined PSO with a crossover operator from GAs and a repair operator equal to the PRA operator used in [78]. This NHBPSO method is compared to a gbest BPSO algorithm with an unspecified penalty function.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Details on chosen parameters, swarm size and number of iterations are missing. Gherboudj *et al.* [44] ran 30 independent runs of both algorithms on 25 small MKPs and 10 large MKPs, with the NHBPSO showing the best average result on all 35 problems. Although no such analysis is included in the paper, this indicates statistically significant outperformance.

The work by Chen *et al.* [17] was already mentioned in section 1.3.2 as having introduced a set-based PSO algorithm. They listed two variants that were applied to the MKP, i.e. the PSO algorithms labeled S-CLPSO-V1 and S-CLPSO-V2, where CLPSO refers to the comprehensive learning PSO algorithm introduced by Liang *et al.* [85]. The first variant, S-CLPSO-V1, only selects feasible elements in the position updating procedure so that the feasibility of the solutions are always guaranteed. The second variant paired the S-CLPSO algorithm with the MKP repair operator introduced by Chu and Beasley [18], which is similar to the “DROP” and “ADD” operators described above for the work by Kong and Tian [72]. S-CLPSO-V2 compared favorably to the PSO-P and PSO-R algorithms from Kong and Tian [72] mentioned above, and was also shown to achieve similar quality of solutions as two ACOs and one evolutionary algorithm. The S-CLPSO algorithms use domain specific information to either only construct feasible solutions or use a repair operator to achieve feasible solutions for the MKP.

The PSO algorithms using repair operators described in this section have been shown to help PSO achieve a higher efficiency in solving the MKP than algorithms without repair operators. The repair operator incorporates knowledge from the domain of knapsack problems to avoid searching known sub-optimal regions of the solution space. Results seem to indicate that the appropriate repair operator is of bigger influence on the success of the PSO algorithm than the PSO algorithm itself. However, no specific study was found that focused on the exact contribution of a repair operator *vis-a-vis* the PSO algorithm. A benefit of the dominance of the repair operator is that less effort needs to be expended on choosing the best PSO algorithm or finding the best parameters for that algorithm. These domain-specific operators, however, make it harder or impossible to apply the algorithms to different problem domains.

2.4.2.2 Penalty functions

A second method to deal with particles that leave the feasible part of the MKP’s solution space is to allow the particles to remain in the infeasible part of the solution space, but to apply a penalty to the fitness of the particle. Different kinds of penalty functions have been proposed, ranging from penalties proportional to the amount with which a constraint is breached to functions that apply an infinite penalty to infeasible solutions. The work by Olsen [107] is an early overview of the main penalty functions that are in use.

Hembecker *et al.* [52] used the binary PSO combined with a penalty function to steer the search towards solutions that satisfy the MKP’s constraints. The penalty was set proportional to the total amount of excess in the knapsacks. The problem set to which the algorithm was applied consisted of 10 small MKPs with the number of items, n, ranging from 28 to 105 and the number of constraints, m, equal to 2 or 30. Only 300 iterations were performed and the algorithm was run 100 times on each of the 10 problems. For only one problem the optimum was found in at least one of the 100 runs, with the author suggesting that more iterations would be required to achieve better results. Thus the study by Hembecker *et al.* [52] does not give an indication of whether the penalty function approach is effective.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

As discussed in section 2.4.2.1, Kong and Tian [72] used the binary PSO on the MKP and constructed two variants: PSO-R which included a heuristic repair operator, and PSO-P which applied a penalty function. The penalty function used in PSO-P applies a penalty that increases linearly with the amount that constraints are violated. The formula for the objective function f including this penalty is

$$f = \sum_{i=1}^{n} v_i x_i - poslin\Big(M_i \Big(\sum_{i=1}^{n} w_{i,j} x_i - C_j\Big)\Big) \tag{2.8}$$

where the penalty parameters M_i are large scalars and the function $poslin$ is defined as

$$poslin(s) = \begin{cases} s & \text{if } s > 0, \\ 0 & \text{if } s \leq 0. \end{cases} \tag{2.9}$$

The two algorithms, PSO-P and PSO-R, were compared using the quality of the solutions found for seven small MKPs and 10 large MKPs. PSO-R outperformed the PSO-P algorithm on all but one simple problem, on which both algorithms performed equally well. The results showed that a penalty function approach applying PSO on the MKP can work, but also that in this case problem domain specific repair operators performed better than a problem domain specific penalty function.

The PBPSO algorithm introduced by Wang *et al.* [146] was discussed in section 1.3.1.3. In their study Wang *et al.* [146] applied PBPSO to the MKP and the performance of the PBPSO algorithm was compared to that of the BPSO (see section 1.3.1.1) and MBPSO (see section 1.3.1.2) algorithms. Wang *et al.* [146] mentioned that a penalty function was used and in the article they referred to the work by Olsen [107]. Unfortunately, the exact penalty function used is not mentioned. The authors concluded that the PBPSO algorithm exhibited better optimization performance in terms of convergence speed and global search ability than BPSO and MBPSO. Only 10 small MKPs were used and no statistical test was performed by Wang *et al.* [146]. Because the aim of the study reviewed corresponds closely to some of the objectives of this thesis, it is important to determine the statistical validity of this conclusion. It proved possible to test for statistical significance of the results reported in the study and the outcome of this statistical analysis is listed in section 2.4.2.3. The PBPSO algorithm could only be shown to outperform MBPSO but not the BPSO. The work by Wang *et al.* [146] shows that it is possible to compare discrete PSO algorithms, but that it is also clear that a larger number of problems is required in order to be able to draw statistically significant conclusions about relative performance. It is also noted that the control parameters of MBPSO and BPSO were not tuned, but values known from earlier studies were used, while for PBPSO only a minimal investigation was conducted on the values of R_{min} and R_{max}. It is unclear whether better tuned versions of the three algorithms would show the same relative performance.

Deep and Bansal [26] proposed a new method called Socio-Cognitive Particle Swarm Optimization (SCPSO) and applied it to the MKP, comparing the new algorithm's performance to that of BPSO. The objective function including an unspecified penalty function approach was used for handling capacity constraints. A star topology was used with a fixed swarm size of 40 particles. No tuning was performed for either algorithm. Both algorithms were applied to the seven small MKPs from the file *mknap1.txt*. Thirty independent runs were performed for each problem and the number of function evaluations was

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

limited in each run to 1000 times the number of items *n*. This resulted in a maximum number of iterations ranging from 150 for mknap1-1 to 1250 iterations for mknap1-7. For details on the specifications of these seven problems, see table 5.1. Deep and Bansal [26] showed that for four out of seven problems SCPSO outperformed BPSO. For the first three problems, both algorithms were able to solve the problems, but the paper did not include a statistical analysis. Due to the limited number of problems and the fact that both algorithms achieved a perfect success in solving the first three problems, no statistically significant outperformance could be determined using the method outlined in appendix E. Because only small MKPs were considered, it is as yet unclear whether the SCPSO algorithm works well on larger MKPs.

Recapping the work done on using PSO with penalty functions on the MKP, one first observes that the work of Olsen [107] showed that many variants of penalty functions exist. These variants range from a simple penalty function that applies a value of minus infinity to infeasible solutions, to those that try to incorporate more information about the problem domain to better steer the search. The latter approach more closely links the PSO algorithm's objective function to the domain of the MKP, and does not allow such linked PSO algorithms to be easily applied in the same form to DOPs from a different domain. This use of domain specific information is, however, weaker than that used by the repair operators reviewed in section 2.4.2.1. The work by Kong and Tian [72] and Gherboudj *et al.* [44] indicates that PSO algorithms using repair operators can achieve better solutions than PSO algorithms using penalty functions.

2.4.2.3 PSO comparisons on MKP by Wang *et al.*

As mentioned previously in section 2.4.2.2, Wang *et al.* [146] compared the BPSO, MBPSO, and PBPSO algorithms on a number of MKPs and concluded that the PBPSO exhibited better optimization performance in terms of convergence speed and global search ability than BPSO and MBPSO. This section contains a statistical analysis of the results reported in the study by Wang *et al.* [146]. The BPSO, MBPSO, and PBPSO algorithms are described in sections 1.3.1.1, 1.3.1.2, and 1.3.1.3 respectively.

This section uses two different measures of the quality of results achieved by the different algorithms. Firstly, the average error denotes the percentage deviation from the known optimum for a given instance. Secondly, the success rate denotes the percentage of independent runs that found the optimum for a given instance.

All average error and success rate numbers in this section are copied or computed directly from [146]. Table 2.1 summarizes the average errors and success rates of three PSO algorithms on 10 small MKPs. The number of problems solved perfectly in all 20 runs of the algorithm is listed as "# perfect", while "# failure" indicates the number of problems for which the algorithm was not able to find the optimum in any of the runs.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 2.1: Summary of results from Wang *et al.* on 10 small MKPs. Bold face indicates statistically significant outperformance of one or more algorithms.

		BPSO	(rank)	MBPSO	(rank)	PBPSO	(rank)
avg error		**1.09 %**	**(2.20)**	1.59 %	(2.40)	**0.80 %**	(**1.40)**
stdev error		1.40 %		2.64 %		1.02 %	
average SR		41.0 %	(2.30)	36.5 %	(2.40)	**49.0 %**	(**1.30)**
stdev SR		44.9 %		45.0 %		41.6 %	
# perfect		3	(2)	3	(2)	3	(2)
# failure		3	(2.5)	3	(2.5)	0	(1)
P-value	avg error	3.68 %		1.27 %			
significant	avg error	Best				Best	
P-value	SR	1.27 %		0.70 %			
significant	SR					Best	

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The detailed results underlying table 2.1 are found in table 2.2 for the average error per problem and in table 2.3 for the success rate per problem.

Table 2.2: Average error results from Wang *et al.* on 10 small MKPs. Results shown are the average of 20 independent runs of the algorithms. Bold face indicates statistically significant outperformance of one or more algorithms. Note that Wang *et al.* used the acronym KBPSO to denote what is called BPSO here.

Problem	Name	Group	*n*	*m*	avg error	(rank)	avg error	(rank)	avg error	(rank)
mknap1-1	Pet1	Simple	6	10	0.00 %	(2)	0.00 %	(2)	0.00 %	(2)
mknap1-2	Pet2	Simple	10	10	0.00 %	(2)	0.00 %	(2)	0.00 %	(2)
mknap1-3	Pet3	Simple	15	10	0.00 %	(2)	0.00 %	(2)	0.00 %	(2)
mknap2-43	Pb4	Simple	29	2	3.46 %	(3)	2.89 %	(2)	2.16 %	(1)
mknap2-44	Pb5	Simple	20	10	0.37 %	(2)	1.31 %	(3)	0.21 %	(1)
mknap2-45	Pb6	Simple	40	30	3.74 %	(2)	8.69 %	(3)	2.98 %	(1)
mknap2-1	Sent1	Complex	60	30	1.48 %	(3)	1.01 %	(2)	0.98 %	(1)
mknap2-2	Sent2	Complex	60	30	0.60 %	(2)	0.69 %	(3)	0.58 %	(1)
mknap2-22	Weish12	Complex	50	5	0.69 %	(3)	0.35 %	(2)	0.11 %	(1)
mknap2-30	Weish20	Complex	70	5	0.59 %	(1)	0.97 %	(3)	0.93 %	(2)
average					**1.09 %**	**(2.20)**	1.59 %	(2.40)	**0.80 %**	**(1.40)**
# perfect					3	(2)	3	(2)	3	(2)
# failure					3	(2.5)	3	(2.5)	0	(1)

Table 2.3: Success rate results from Wang *et al.* on 10 small MKPs. Results shown are the combined success rate over 20 independent runs of the algorithms. Bold face indicates statistically significant outperformance of one or more algorithms. Note that Wang *et al.* used the acronym KBPSO to denote what is called BPSO here.

Problem	Name	Group	*n*	*m*	SR	(rank)	SR	(rank)	SR	(rank)
mknap1-1	Pet1	Simple	6	10	100 %	(2)	100 %	(2)	100 %	(2)
mknap1-2	Pet2	Simple	10	10	100 %	(2)	100 %	(2)	100 %	(2)
mknap1-3	Pet3	Simple	15	10	100 %	(2)	100 %	(2)	100 %	(2)
mknap2-43	Pb4	Simple	29	2	20 %	(2)	15 %	(3)	40 %	(1)
mknap2-44	Pb5	Simple	20	10	65 %	(2)	5 %	(3)	75 %	(1)
mknap2-45	Pb6	Simple	40	30	10 %	(2.5)	10 %	(2.5)	15 %	(1)
mknap2-1	Sent1	Complex	60	30	0 %	(2.5)	0 %	(2.5)	5 %	(1)
mknap2-2	Sent2	Complex	60	30	0 %	(2.5)	0 %	(2.5)	5 %	(1)
mknap2-22	Weish12	Complex	50	5	15 %	(3)	35 %	(2)	45 %	(1)
mknap2-30	Weish20	Complex	70	5	0 %	(2.5)	0 %	(2.5)	5 %	(1)
average					41 %	(2.2)	37 %	(2.4)	**49 %**	**(1.4)**
# perfect					3	(2)	3	(2)	3	(2)
# failure					3	(2.5)	3	(2.5)	0	(1)

The specifications of the three algorithms and the experimental procedure followed by Wang *et al.* [146] are repeated here. For the first six problems, together called the “simple group” of problems, the

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

swarm size was set to 30 particles and a maximum of 3000 iterations of the algorithms were performed. For the final four problems, together called the “complex group” of problems, the swarm size was set to 60 particles and a maximum of 4000 iterations of the algorithms were performed. In all cases 20 independent runs of the algorithm were performed. For MBPSO the parameter a was set equal to 0.5 and kept constant. For BPSO and PBPSO the common parameters were kept equal and set to $\omega = 0.8$, $c_1 = 2.0$ and $c_2 = 2.0$. No velocity clamping was used in the BPSO. For PBPSO the remaining parameters R_{min} and R_{max} were set to -50 and 50 respectively.

The newly introduced PBPSO algorithm performed best measured in average error (0.80% versus 1.09% for BPSO and 1.59% for MBPSO) as well as success rate (49.0% versus 41.0% for BPSO and 36.5% for MBPSO). The BPSO and MBPSO algorithms showed a somewhat similar performance, with BPSO achieving a slightly better average rank. Due to the limited number of problems in the problem set, however, statistically significant outperformance is not clear cut. Using a significance level of $\alpha = 5\%$, only based on the success rate is PBPSO shown to outperform the other two algorithms. Based on the average rank of the average error numbers, PBPSO is shown to outperform MBPSO, but then no significant difference in performance is seen compared to BPSO. The statistical procedure used to make these comparisons is outlined in appendix E.

Note that, although the appearance of the tables in this section is similar to that used when reporting on the experiments performed for the purpose of this thesis in chapter 5, the results for each algorithm and problem reported in this section are different and not directly comparable to the results from chapter 5: Wang *et al.* [146] used much less extensive tuning for the algorithms; the number of iterations, the swarm size and the number of independent runs of the algorithm all differ from the choices made for the experiments reported in this thesis.

To repeat the conclusion from the previous section, the work by Wang *et al.* [146] showed that it is possible to compare discrete PSO algorithms, but that a larger number of problems is required in order to be able to draw statistically significant conclusions about relative performance. Also, the control parameters for MBPSO and BPSO were not tuned, but values known from earlier studies were used, while for PBPSO only a minimal investigation was conducted on the values of R_{min} and R_{max}. It is unclear whether better tuned versions of the three algorithms would show the same relative performance.

2.5 Conclusions

The first objective of this chapter was to introduce the MKP, list its main characteristics, and identify further features that can be useful in categorizing MKPs. A formal mathematical definition of the MKP as a maximization optimization problem was given, with the variables n for the number of items, m for the number of constraints and the weights $w_{i,j}$ for item i relative to constraint j.

By reviewing a well-known set of benchmark problems in section 2.3, more insight was gained on the impact of the variables n and m on the hardness of a MKP. Another feature considered to be relevant for the randomly generated problems in the large benchmark problems was the tightness ratio r. The insights pointed to the conclusion that different algorithms, or even different tunings of the same algorithm, can perform better on a set of problems with a certain combination of n, m, and r, while it will perform less

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

well on sets of MKPs with different specifications of n, m, and r.

The second and more important objective of this chapter was to give a brief overview of the various methods that have been employed to solve the MKP, with a focus on approaches using PSO. A brief overview was given of exact methods and some recent work using population based methods other than PSO. Then a more detailed review was made of work from literature where the PSO was applied to the MKP. A defining characteristic in these approaches was how the algorithms reviewed dealt with the problem of swarm particles straying into infeasible parts of the solution space where the candidate solution represented by a particle does not satisfy all the constraints posed by the MKP. Two different mechanisms have been used to address this problem: repair operators and penalty functions.

The best solutions for the MKP have been achieved by algorithms that explicitly use features from the problem domain. Some of the most successful of these have been population based methods using repair operators. Some authors, for example Kong and Tian [72], have combined such repair operators with the PSO, while others used different population based methods, like Chu and Beasley [18] who used a GA. The conclusion is that the repair operator makes these methods more successful than other population based approaches, but there is no clear indication that PSO underperforms or outperforms other population based approaches in this setup.

Approaches to solve the MKP using PSO and penalty functions were shown to be effective, though these yielded poorer results than PSO algorithms using a repair operator. A benefit of the penalty function approach is that the amount of problem domain specific knowledge that is used can be proportioned. Thus, it is possible to chose a penalty function that is generic and can be employed on problem domains other than the MKP without the need for redesign. Most articles reviewed that use penalty functions, however, do not specify the exact penalty function used. Therefore it is not known if using a generic penalty function with the minimum number of domain specific features is an effective way to solve the MKP.

The review of methods to solve the MKP using PSO should be considered sufficient to prove the non-trivial nature of the problem: first, new discrete PSO algorithms are introduced and first tested on the MKP, indicating that it is an important set of problems to use for testing new algorithms. Second, note that for some of the benchmark problems that have been used in research on the MKP since 1998, it is still not clear if the optimal solution has been found. Thus, it seems fair to conclude that the MKP is a valid test bed of the new PSO algorithm to be introduced in chapter 4 and other discrete PSO algorithms it will be compared to.

The next chapter will introduce a second DOP, namely the feature selection problem. Besides a formal definition and cursory investigation of salient aspects of the problem, the main objective of chapter 3 will be the same as this chapter's: review the literature on methods used to solve the DOP to see if it can form a test bed for the new PSO algorithm to be introduced in chapter 4 and other discrete PSO algorithms it will be compared to. For the MKP itself, the experiments conducted and reported in this thesis in order to test PSO algorithms are detailed in chapter 5.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 3

Feature selection problem

This chapter describes the feature selection problem (FSP), an optimization problem from the domain of machine learning. This is the second of two sets of problems that will be used in this thesis to test the SBPSO algorithm, the other being the MKP that was described in the previous chapter.

This chapter has two objectives, the first of which is to formally define the FSP as well as the underlying classification problem and list the FSP's main characteristics. The second objective is to give a brief overview of the various methods that have been employed to solve the FSP, with a focus on approaches using PSO. This overview is meant to convince the reader that the FSP is a valid problem on which to test discrete PSO algorithms like the SBPSO.

This chapter formally defines the classification problem and the FSP, followed by a brief overview of the literature on solving the FSP. The final section also introduces a set of benchmark datasets from the UCI Machine Learning Repository that help define a set of benchmark FSP's.

3.1 Introduction

The FSP is a problem in the domain of machine learning where an algorithm is used to label or classify data. The goal in the FSP is to select the optimum subset of variables that will result in the best accuracy for the underlying classification problem. This chapter aims to formally introduce the FSP, its relation with the underlying classification problem, and the place of both within the wider field of machine learning. In the course of this brief review, important concepts like classifiers, classification accuracy, and cross validation will be introduced. These concepts will be important in the construction of the experiments for the FSP in chapter 6. In those experiments four PSO algorithms will each be used to select a subset of features which in turn will act as input for the underlying classification problem. Three different classifiers will use the input features to determine the class labels, i.e. the *k*-nearest neighbor classier, decision trees, and the Gaussian Naive Bayes classier. This chapter will also introduce these three classifiers in some detail.

The second objective of this chapter is to convince the reader that the FSP is a valid test problem on which to compare the SBPSO algorithm to other PSO algorithms. In order to achieve this goal a review is made of the literature on the FSP and ways that have been employed to solve the problem. The

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

main approaches will be touched upon, with specific focus on those methods that use population based methods such as PSO.

Thus it will be made clear that the manner in which PSO algorithms will be used to solve the FSP in chapter 6 has been used successfully in other studies.

The MKP was introduced in chapter 2, where it was argued that the MKP formed a good problem in which to test the SBPSO. This chapter will attempt to do the same for the FSP. The idea is that testing the SBPSO on two different DOPs will lead to better evidence on whether the SBPSO has merit as a generic algorithm, and not applicable to just one problem. For this argument of wider scope to carry weight, the MKP and the FSP need to be sufficiently different. It is important to remember from chapter 2 that the MKP is a completely deterministic problem that has a clear way to determine what candidate solution is best: the solution that leads to the highest value of items in the knapsack. Although the optimal solution need not be known, it is always possible to compare two candidate solutions and determine which is better. By clearly defining the FSP and the underlying classification problem, it will be shown that the FSP differs fundamentally from the MKP. Hence, the FSP forms a test bed for the SBPSO that is truly different from the MKP.

This introductory section is followed by three main parts and a conclusion that links back to the objectives outlined above. The first main section sketches the broader field of machine learning and the place that classification has in this field. It also includes an explanation of the concept of classifiers, performance measures and the use of cross validation, as well as some notes on data preprocessing. The second main part formally defines the FSP within the framework of the first section. The third part is a review of the literature on various approaches used to solve the FSP. A special focus will be on methods that use PSO. Contained in this review is an introduction to the UCI Machine Learning Repository, which houses a collection of problems which serve as a benchmark for testing machine learning algorithms.

3.2 The classification problem in machine learning

This section first defines the field of machine learning and gives a very broad overview of the three main ways in which algorithms learn from experience, i.e. unsupervised, supervised and reinforcement learning. Following this, the classification problem that underlies the FSP is formally defined.

3.2.1 Machine learning

Machine learning was defined by Simon [129] as “any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population”. Mitchell [97] has a more formal approach and defines machine learning to mean that a computer program or algorithm *learns* from experience E with respect to some class of tasks T and a performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. For the classification problem the performance measure, P, is usually taken to be some measure of the accuracy achieved by the algorithm in classifying a set of known examples. Performance measures are discussed in more detail in section 3.2.5.

Some examples of learning problems that are well-defined such that machine learning can be applied

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

include: learn to win at a board game (checkers, chess), learn to recognize speech or other patterns, or learn to drive an autonomous vehicle.

One way to segment the field of machine learning is along the lines of what kind of experience is used to learn: is the algorithm given no feedback, limited feedback, or explicit feedback on what the correct outcome is in a given learning situation? This results in three main branches of machine learning: unsupervised learning, reinforcement learning, and supervised learning.

In unsupervised learning the aim is to find hidden structure in unlabeled data [29]. Since the examples given to the algorithm are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning and reinforcement learning. Unsupervised learning approaches include clustering, hidden Markov models, and feature extraction techniques such as principal component analysis.

Reinforcement learning was inspired by behaviorist psychology, and is concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. According to Mitchell [97] reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goal. It differs from standard supervised learning in that data is not labeled, nor are sub-optimal actions explicitly corrected. Further, there is a focus on on-line performance. Real world problems on which these techniques are applied include robot control or real-time anomaly detection. Algorithms in the field of reinforcement learning are often closely related to those from dynamic programming. An example algorithm is the Q algorithm [147].

Supervised learning is defined by Mohri *et al.* [99] as learning which aims to infer a function from a set of labeled training examples. Each example is a pair consisting of an input object (typically a vector) and a desired output value (the class). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used to map examples not in the training set. The algorithm thus tries to generalize from the training data to unseen situations. Two large branches of supervised learning are classification and regression. Classification is the subject of section 3.2. Regression or regression analysis is a statistical process for estimating the relationships between a dependent variable and one or more independent variables. Regression analysis is widely used for prediction and forecasting, for example in the world of finance by Li *et al.* [84] and Claessens *et al.* [22]. An example regression technique is linear regression or least squares.

3.2.2 Classification problem

The classification problem in machine learning is the problem of automatically labeling data: an algorithm is used to perform the task of determining in which *class* from a finite set a given *instance* of input data is to be classified. An example of such classification tasks is to determine the outcome of a game of tic-tac-toe from the end-state of Xs and Os on the board. The possible classes are "game won by X", "game won by O", or "draw". In this case the output class for a given set of inputs is fully determined and the tasks is for the algorithm to learn this target concept [94]. A different example classification task is to use a number of measurements for a patient regarding temperature, blood pressure, and the levels of various blood chemicals, to determine whether or not the patient suffers from a specific disease. In this

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

case the relationship between input and output is not known: a human doctor uses the measurements of the patient to diagnose whether he/she suffers from the disease in question, but this human diagnosis is not infallible and a direct link between the measurements and the diagnosis in usually not known.

Within the taxonomy of machine learning from the previous section, the classification problem is a supervised learning problem because the experience, E, which the algorithm uses to improve its performance consists of known examples: a set of inputs to be classified for which the correct output classes are known. Formally, these examples form a training set of data instances (I,c) with input vector I and the correct or desired output class c. The input vector $I = \{i_k\}_k = 1^n$ contains n features or attributes, which each can have different values. In this thesis the terms "feature" is used in most places, but sometimes convention calls for the use of "attribute" instead; for example in the field of decision trees discussed in section 3.2.3.3. There is no difference in meaning implied between the two terms. Attribute values are considered to be finite. Possible value types are numeric (a continuous value), and nominal or discrete (a fixed and finite number of possibilities). The discrete values can just be a finite number of choices without structure (for example the colors red, yellow, and blue) or contain an internal hierarchy or structure (for example the outcomes of a hotel review "excellent", "good", "average", "bad", and "appalling"). The output class c is one from a known and finite set of possible classes. Using the elements involved in the definition of machine learning by Mitchell [97] from the previous section, classification is thus the class of tasks to be performed, T. The other elements are

- the algorithm that does the learning and which, after being trained, performs the classification task. This algorithm is called a *classifier* and is the subject of section 3.2.3;
- the performance measure P used in learning, which will be discussed in section 3.2.5; and
- the experience E on which to train the classifier. If the task contains of a fixed set of training instances, this is included in the problem. But in many cases only a dataset of labeled instances is given. In this case it is important how the dataset is divided into a training set from which to learn and a second set on which to measure the performance of the trained classifier. The concept of *cross-validation* deals with this division of a dataset, and is the subject of section 3.2.5.2, and follows after the discussion on performance measures to which cross-validation is closely linked.

3.2.3 Classifiers

This sections briefly further elaborates on the the concept of a classifier used to solve the classification problem, including detail on of three classifiers that are used in the experiments on the FSP in this thesis, i.e. the Gaussian Naive Bayes classifier in subsection 3.2.3.2, the decision tree classifier in subsection 3.2.3.3, and the k-Nearest Neighbor classifier in subsection 3.2.3.4.

3.2.3.1 Definition of a classifier

A classification problem involves a learning algorithm trained by experience, producing an induced model or function. This induced model is used to classify new instances for which the output class is not known. The induced model is called a *classifier*. Strictly speaking, a difference thus exists between

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the (abstract) learning algorithm and the induced model, which is an instance of the learning algorithm trained on a specific dataset. This difference is dropped in the remainder of this thesis, however, and the term classifier is used for both algorithm and induced model.

The first phase in which the classifier learns from the labeled instances is called the *training phase*. The learning algorithm needs to generalize from the training data to new unseen data as best as possible. The trained classifier can then be used to classify previously unseen instances in the *classification phase*.

Kotsiantis [76] mentioned that classifiers were developed from different types of learning approaches:

- Logic based learning has yielded decision trees and rule based learners.
- Statistical approaches include Gaussian Naive Bayes (GNB) and Bayesian networks.
- The best-known example of instance based learning is the k-Nearest Neighbor (k-NN).
- Perceptron based learning comes in the form of artificial neural networks (ANN) and radial basis function (RBF) networks.
- Support vector machines (SVM) form their own separate class.

The classifier may have no parameters (for example GNB), one (e.g. k-NN) or more parameters (e.g. decision trees, SVM). For some classifiers further choices need to be made, e.g. a choice of distance measure (metric) for the k-NN classifier or a choice of kernel function to transform the feature space for SVM. These parameters and other metrics greatly determine the accuracy of the resulting classifier. For the classifiers described in the next sections, particular attention is given to these parameters and metrics.

3.2.3.2 Gaussian Naive Bayes classifier

The name Gaussian Naive Bayes classifier [60] indicates that three concepts are combined to form the classifier. The term **Bayes** is related to Bayes' Theorem [6] which states how the mathematical probabilities of two events A and B relate to the *conditional* probabilities of A given B (what is the probability of A, given that B is known) denoted as $\mathbb{P}(A|B)$ and the conditional probability of B given A denoted as $\mathbb{P}(B|A)$. In its simplest form this relationship is given as

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}. \tag{3.1}$$

The term **Gaussian** indicates that the classifier assumes the values for each feature to be distributed according to a Gaussian or normal probability distribution. A Gaussian distribution is fully determined by two values: the mean value μ and the standard deviation σ^2. The probability density function which describes the probability distribution is

$$\phi(x,\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}}, \tag{3.2}$$

such that

$$\mathbb{P}(X < x) = \int_{-\infty}^{x} \phi(t)dt. \tag{3.3}$$

The term **Naive** involves the mathematical concept of conditional independence. To introduce this

© University of Pretoria

concept, first note that two events A and B are considered *independent* if the following holds:

$$\mathbb{P}(A,B) = \mathbb{P}(A)\mathbb{P}(B) \tag{3.4}$$

where $\mathbb{P}(A,B)$ denotes the joint probability of A and B. In words, this can be interpreted as that the value of A does *not* depend on the value of B and vice-versa. Two events A and B are considered *conditionally independent given a third event* C, if the following holds:

$$\mathbb{P}(A|B,C) = \mathbb{P}(A|C). \tag{3.5}$$

In words, this can be interpreted as given that C is known, the value of A does not depend on the value of B. Or less rigorously, the only link between A and B is via the third event, C. The concept Naive in the GNB classifier refers to the (strong) assumption that the values of features F_i and F_j are pair-wise conditionally independent (for $i \neq j$) given the value of the class label C.

The GNB classifier calculates the probability that an instance belongs to a class C, given the values of the features F_i. Assuming the data contains only two features, this probability is denoted as $\mathbb{P}(C|F_1,F_2)$. The GNB classifier tries to find that C which maximizes $\mathbb{P}(C|F_1,F_2)$.

Since the feature values F_1 and F_2 are given, $\mathbb{P}(F_1,F_2)$ is a constant. Hence, maximizing $\mathbb{P}(C|F_1,F_2)$ will yield the same result as maximizing $\mathbb{P}(C|F_1,F_2)\ /\ \mathbb{P}(F_1,F_2)$. Maximizing the latter term has some mathematical benefits, as is made clear below.

Bayes' Theorem tells us that

$$\mathbb{P}(C|F_1,F_2) = \frac{\mathbb{P}(C)\mathbb{P}(F_1,F_2|C)}{\mathbb{P}(F_1,F_2)} \tag{3.6}$$

Using the assumption that F_1 and F_2 are conditionally independent given C, it is possible to use the chain rule of probability[1] to write

$$\begin{aligned}\mathbb{P}(C|F_1,F_2)\ /\ \mathbb{P}(F_1,F_2) &= \mathbb{P}(C)\mathbb{P}(F_1,F_2|C)\\ &= \mathbb{P}(C)\mathbb{P}(F_1|C)\mathbb{P}(F_2|F_1,C)\\ &= \mathbb{P}(C)\mathbb{P}(F_1|C)\mathbb{P}(F_2|C). \end{aligned} \tag{3.7}$$

In order to calculate $\mathbb{P}(C|F_1,F_2)/\ \mathbb{P}(F_1,F_2)$ it is thus sufficient to calculate the simpler probabilities $\mathbb{P}(C)$, $\mathbb{P}(F_1|C)$, and $\mathbb{P}(F_2|C)$. The general case of n features can be written as

$$\mathbb{P}'(C|F_1,\ldots,F_n) = \mathbb{P}(C)\Pi_{i=1}^{n}\mathbb{P}(F_i|C) \tag{3.8}$$

3.2.3.2.1 Training phase: The training phase for the GNB classifier consists of determining the values for all probabilities $\mathbb{P}(C=c)$ and $\mathbb{P}(F_i|C=c)$ for features F_i and c ranging over all possible values of the class label C. The probabilities $\mathbb{P}(C=c)$ are called the *class prior* and need to be estimated for each value of c based on the training data. The Gaussian assumption means that $\mathbb{P}(F_i|C=c)$ is assumed to be normally distributed for all features F_i and class value c. To determine this distribution, the parameters μ_i and σ_i^2 need to be estimated.

First the class prior probabilities $\mathbb{P}(C=c)$ are set equal to the *maximum likelihood estimate* based on

[1]The chain rule of probability is $\mathbb{P}(A,B) = \mathbb{P}(B)\mathbb{P}(A|B)$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the class count in the training set:

$$\mathbb{P}(C=c) = \frac{\text{\#instances with class label } c}{\text{\#instances } c} \tag{3.9}$$

Note that this means that only class labels that exist in the training set have a probability greater than zero and that the GNB classifier will never assign to an instance a class label that does not exist in the training set. A mechanism to overcome this shortcoming is to apply Laplace smoothing. The class prior probabilities in that case are calculated as

$$\mathbb{P}(C=c) = \frac{\text{\#instances with class label } c + \alpha}{\text{\#instances } c + \alpha \text{ \#classes}} \tag{3.10}$$

with α an integer greater than zero. In simple terms, this can be understood as assuming that for each class an additional α instances are present in the training set. The most commonly used value for α is one. A higher value of α means that the class priors are more smoothed, or pushed towards the uniform distribution where each class has the same class prior probability.

The Gaussian parameters are also set equal to their maximum likelihood estimates, which are just the class mean and standard deviation in the training set. Denote by $\text{Train}(c)$ the set of instances from the training set that have class label c and let $F_i(t|c)$ be the value of feature F_i for instance $t \in \text{Train}(c)$. Then the maximum likelihood estimates for parameters $\mu_i(c)$ and $\sigma_i^2(c)$ of F_i given $C=c$ are

$$\mu_i(c) = \sum_{t\in\text{Train}(c)} \frac{F_i(t)}{|\text{Train}(c)|} \tag{3.11}$$

$$\sigma_i^2(c) = \sum_{t\in\text{Train}(c)} \frac{[F_i(t) - mu_i(c)]^2}{|\text{Train}(c)|} \tag{3.12}$$

$$\tag{3.13}$$

3.2.3.2.2 Classification phase: Classification using the GNB classifier requires finding the class label c that maximizes the conditional probability $\mathbb{P}(C=c|F_1,\dots,F_n)$ for the unlabeled instance t with feature values $(f_1,\dots,f_n)$. Written as a mathematical equation, this is equivalent to equation (3.8):

$$\text{GNB-classification}(t) = \underset{c}{\text{argmax}}\ \mathbb{P}(C=c)\prod_{i=1}^{n}\mathbb{P}(F_i=f_i|C=c) \tag{3.14}$$

Should an instance have two or more classes which have the same conditional probability, the tie is broken randomly.

In short, the GNB classifier makes the (strong) assumptions that

1. all the features in the data have values that are normally distributed given the class label C, and
2. if the class value C is known, all features are pair-wise conditionally independent.

GNB then uses maximum likelihood estimates to determine the class prior $\mathbb{P}(C)$ and the Gaussian parameters μ_i and σ_i^2 of the conditional distributions of the features F_i given C. Classification is finding that c which maximizes the conditional probability $\mathbb{P}(C=c|F_1,\dots,F_n)$ for the feature values given in the unlabeled instance.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Even though in reality the assumptions of normality and conditional independence are often broken, the GNB classifier has proven to be quite robust [120]. One of its strengths is that it handles large feature spaces quite well, as the conditional independence assumption means that training consists only in calculating $|C|$ averages and standard deviations without considering any of the many covariances.

3.2.3.3 Decision tree classifier

A decision tree classifier uses a tree as the basis of a model, with the leaves of the tree representing the class labels, and higher nodes in the tree encode decisions based on the values of the attributes. Classification of an instance with a decision tree involves starting at the root of the tree and flowing down the branches of the tree, where the decision node determines which branch is chosen based on the value of a single attribute. This process continues until a leaf node is reached which contains the classification for the instance.

A first mention of the decision tree concept is the concept learning system from Hunt *et al.* [54], but the field of decision trees is considered to have really started with the work of Quinlan [117] who developed the ID3 algorithm. This was later expanded upon and improved into the C4.5 algorithm [118] and its latest version C5.0.

Training a decision tree consists of building the whole tree structure with leaves and decision nodes. This training is done using a decision tree induction algorithm. Following the work by Kotsiantis [76], a decision tree induction algorithm (here shown for C4.5 using the information gain) can be summarized by the recursive algorithm 3 which returns the root of the final decision tree.

Algorithm 3: Decision Tree Induction Algorithm

```
DecisionTree(Set $S$):
begin
    begin Check base cases:
        if all instances in the data have the same class label then
            return a leaf node with the unique class label ;
        end
        if the instances in $S$ contains 0 attributes then
            return a leaf node with the most common class label in $S$ ;
        end
    end
    begin Recursive step:
        forall the attributes $A$ in $S$ do calculate $IG(S,A)$ ;
        $A_{\text{best}} := \text{argmax}_A\{IG(S,A)\}$ (ties broken randomly) ;
        forall the values $a_i$ of $A_{best}$ do
            Set $S_i := \{ s' \in S : A_{\text{best}} = a_i$, attribute $A_{\text{best}}$ removed from $s'$ $\}$ ;
            Node $d_i$ := DecisionTree($S_i$) ;
        end
        return node $d$ which splits based on $A_{\text{best}}$ into children $d_i$ ;
    end
end
```

The *Check base cases* step at the start of algorithm 3 deals with those sets of instances where a

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

decision tree with a single leaf node is the best classification. In the *Recursive step* a node is created that splits data based on the m values of A_{best} and has m child-nodes. These child nodes d_i are each made by calling the `DecisionTree` algorithm on the subset S_i which contains all instances in S that have $A_{\text{best}} = a_i$, but with the attribute A itself removed from all the instances. So if the set S contains instances with n attributes, then all S_i contain instances with $n-1$ attributes.

The most important step in the induction algorithm is constructing the decision nodes that lead to splits in the dataset based on a single attribute. At each step an attribute needs to be selected that, for some metric, leads to the best split in the data. In other words: which attribute gives the "best" split of the instances in that branch of the tree. The C4.5 algorithm uses *information gain* as the metric to be maximized. To define information gain denote the following:

- S is a set of labeled instances with n different class labels, $c_1, \ldots, c_n$;
- A is an attribute of the instances in S, with m discrete values, $a_1, \ldots, a_m$;
- $S_{A=a_i}$ is the subset of S with all instances where attribute A has value a_i;
- $f_S(cond)$ is the fraction of instances in S for which $cond$ is true, e.g. $f_S(A = a_i) = |S_{A=a_i}| \,/\, |S|$.

The *information entropy* of the set S is defined by its class distribution according to

$$E(S) := -\sum_{j=1}^{n} f_S(C = c_j) \log_2 f_S(C = c_j) \tag{3.15}$$

The information entropy of the set S on attribute A is defined equivalently based on the distribution of the values a_i of A:

$$E(S,A) := -\sum_{j=1}^{m} f_S(A = a_i) \log_2 f_S(A = a_i) \tag{3.16}$$

Combining the two information entropies above, the information gain $IG(S,A)$ indicates the decrease in information entropy for set S by splitting the set based on the values of attribute A:

$$IG(S,A) := E(S) - E(S,A) \tag{3.17}$$

At each step in the decision tree induction algorithm the attribute A_{best} is selected which maximizes $IG(S,A)$ over all attributes A present in the instances in S.

Note that the description of information entropy $E(S,A)$ and algorithm 3 all assume that the attributes A have a finite set of discrete values, leading to decision nodes $(A = a_i)$. In the induction algorithm C4.5, Quinlan [119] added the functionality to also deal with continuous attributes. These decision nodes take the form of $(A \le t)$ for some threshold t. The algorithm checks different values of the threshold. If S contains N instances, each with a continuous value $a_{\text{cont},i}$, then the thresholds to check are the $N-1$ midpoints, $t_k = \frac{1}{2}(a_{\text{cont},(k)} - a_{\text{cont},(k+1)})$, between the (ranked) values and the algorithm selects the threshold that maximizes the information gain for the attribute A.

After the main loop of the algorithm has created a decision tree, a further pruning step can be made. In the pruning step, the algorithm attempts to remove those branches from the tree that may cause overfitting. This is done because a smaller tree requires less memory and leads to quicker classification. For

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

more details about the pruning of decision trees, see for example the work by Esposito *et al.* [33] and Oliver and Hand [106].

3.2.3.4 *k*-Nearest Neighbor classifier

The *k*-nearest neighbor classifier is one of the oldest classifiers, with roots going back to the work of Fix and Hodges [35]. It is a popular choice of classifier because it is simple and easily implemented. It classifies a new instance by calculating the distance from the new instance to each of the labeled instances in the training set using a pre-defined distance metric. Then the *k* instances from the training set that have the shortest distance to the instance to be classified are selected - its *k* nearest neighbors. Ties in the shortest distance are broken randomly. The classification of the unlabeled instance is by majority rule for the classes present in the *k* nearest neighbors, again with ties broken randomly. As such, no distinct training phase can be identified for this classifier, as calculating the distances to the instance to be classified can only be done in the classification phase.

Different distance metrics can be used to determine the distance between two instances. For instances with continuous valued attributes, the Euclidean distance metric is often used. This metric is defined as

$$d_{\text{Euclidean}}(i_{\text{cont}}, j_{\text{cont}}) \quad := \quad \sqrt{\sum_{n=1}^{N} (i_{\text{cont},n} - j_{\text{cont},n})^2} \tag{3.18}$$

where i_{cont} and j_{cont} are two instances with N continuous valued attributes. For instances with nominal or categorical attributes, the Hamming distance is popular, which counts the number of attributes for which the instances have different values. This metric is defined as

$$d_{\text{Hamming}}(i_{\text{nom}}, j_{\text{nom}}) \quad := \quad \sum_{n=1}^{N} \mathbb{1}\{i_{\text{nom},n} \neq j_{\text{nom},n}\} \tag{3.19}$$

where i_{nom} and j_{nom} are two instances with N nominal attributes. The $\mathbb{1}$ denotes the indicator function that equals 1 if the condition following is true, and 0 otherwise. Other metrics that can be used are Minkowsky, Manhattan, Chebysev, Camberra, or Kendall's rank correlation [76].

The parameter *k* of how many nearest neighbors to include in the class voting needs to be determined beforehand. The value 1 is popular, taking only the single nearest neighbor into account in determining the classification. It benefits from a slightly faster implementation in which only the nearest neighbor needs to be found.

Incorporating feature selection into the *k*-Nearest Neighbor is simple, as only those attributes or chosen in the feature selection process are taken into account when the distances between instances are calculated. For example, consider the case of the Euclidean distance metric for two instances with n attributes, where the attributes with indices $a_1, \ldots, a_m$ are selected. Then the distance between two instances i and j is calculated as

$$d_{\text{Euclidean}}[a_1, \ldots, a_m](i_{\text{cont}}, j_{\text{cont}}) := \sqrt{\sum_{k=1}^{m} (i_{\text{cont},a_k} - j_{\text{cont},a_k})^2}, \tag{3.20}$$

where the two instances are treated as having only the m selected attributes.

A downside of using the *k*-nearest neighbor classifier is that it is computationally expensive and

© University of Pretoria

locating the nearest neighbor is itself a problem that is studied in computer science. Therefore, work is ongoing to optimize the classifier, for example by methods called prototype selection [41] and by combining nearest neighbor with clustering [57]. In this thesis the standard version of the *k*-Nearest Neighbor algorithm is used, as computational efficiency is not the main goal but instead the comparison of different PSO algorithms under the same (in this case sub-efficient) circumstances.

3.2.4 Data preprocessing

The representation and quality of the data is one of the main factors which affect the success of machine learning in general and classification in particular [77]. Data pre-processing is used to transform the raw dataset into one that can be handled by the chosen classifier and is likely to yield the most accurate classifications. Data pre-processing in general includes methods such as data cleaning, transformation, normalization, and feature selection. The product of data pre-processing is a final dataset that can be used effectively by machine learning algorithms.

Irrelevant and redundant information also makes knowledge discovery during the training phase difficult. This is the problem that feature selection tries to address. Feature selection can be seen as part of the data pre-processing. In this thesis, however, the focus lies specifically on using the FSP as a problem to solve and therefore feature selection is *not* treated as a step in the data pre-processing.

The other three general methods of data pre-processing are discussed in turn below.

3.2.4.1 Data cleaning

Data cleaning deals with incorrect values in the dataset. These come in two types: missing or illegal values for an attribute or class, or legal but incorrect values. The former two types are easily recognized, while the latter can be much harder to spot.

Such missing or illegal values can make the task of classification impossible for specific classifiers, although other classifiers may be able to cope with these errors. For example, it is not possible to calculate the Euclidean distance between two data instances if one attribute value is missing in one of the instances, hence making an *k*-nearest neighbor classifier fail. Illegal values occur especially in real-life datasets, e.g. where a letter is stated as value for a numerical attribute, or a negative number for an attribute that must be positive (like the number of children for an individual). If a dataset contains such illegal or missing data, these instances need to be discarded or cleaned the ensure all classifiers work properly.

The methods to deal with illegal values and missing values are the same and can mostly be applied on an attribute-by-attribute manner, meaning that each case of a missing or illegal value can be resolved separately. Note that illegal combinations of attribute values can also exist (for example, an instance of data with the attributes “Gender” = Male, and “Pregnant” = Yes). Such combinations can often only be caught by understanding the underlying concepts encoded in the data, and thus an expert needs to construct checks to catch such illegal combinations. If it is not possible to identify which of the multiple attributes has a correct value, all conflicting attribute values should be treated as illegal individually.

Kotsiantis *et al.* [77] mentioned the following methods to to fill in missing values or overwrite illegal

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

values:

- *Ignoring instances with unknown feature values*: All instances with at least one missing or illegal feature value are ignored.
- *Most common feature value*: The value of the feature that occurs most often is selected to be the value for all the unknown values of the feature.
- *Concept most common feature value*: This time the value which occurs the most common within the same class is selected to be the value for all the unknown values of the feature.
- *Mean substitution*: Substitute a feature's mean value computed from available cases to fill in missing data values on the remaining cases. Usually, the mean for all samples belonging to the same class is used to fill in the missing value.
- *Regression or classification methods*: Develop a regression or classification model based on complete case data for a given feature, treating that feature as the outcome and using all other relevant features as predictors.
- *Hot deck imputation*: Identify the most similar case to the case with a missing value and substitute the most similar case's Y value for the missing case's Y value.
- *Treating missing feature values as special values*: Treating unknown itself as a new value for the features that contain missing values.

Each of these methods introduces its own, sometimes subtle, bias into the resulting dataset and they should therefore be used with care. In general, the larger the dataset is and the smaller the proportion of instances with errors, the less important the errors become and removing the incorrect instances from the set or replacing the missing values with the most common or average value is fine. In a smaller dataset or if the error occurs in an instance that is classified as a relatively rare class, more effort should be expended to find a replacement value that is expected to introduce the smallest possible bias: a regression, hot deck imputation might be better, although more work. In real-life applications, using a special value for the missing or illegal values will make explicit the problem the missing values pose. If this problem leads to poor performance on the classification task, one can compare the costs and benefits of additional effort to obtain better data.

The second type of incorrect values is just that: an attribute has a value (it is not missing) and that value is a legal value, but it is an incorrect value. This can be due to measurement errors, incorrect data entry, or copying errors. Whereas for illegal and missing values an error is clearly recognizable, for this second type of errors there is no such clear distinction. If the value of an attribute shows a large deviation from the other values for that attribute, it can be considered an *outlier*. The main approach to search for such outliers is by statistical analysis of the values for that attribute in the dataset. An even more difficult problem is if an attribute's value is in error, but still falls within the probable range of values for that attribute. In this case the single-attribute statistical analysis for outliers will yield no result. These errors are therefore called *inliers*. Only by multi-variate statistical analysis across multiple or all attributes can such errors be found. For both outliers as well as inliers the expert performing the experiments has a

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

large subjective influence on the choice of what constitutes an outlier or inlier error. Once recognized, this value can be treated the same as illegal or missing values in the data cleaning.

3.2.4.2 Data transformation

Data transformation alters the representation of an attribute's value. The transformation makes the dataset better suitable for the classifier that is subsequently applied to the dataset and hence is intended to improve classification. Several methods of transformation are recognized: discretization, smoothing, and aggregation [50]. It is also possible to see normalization, the subject of the next section, as a specific form of transformation. Note that this description of transformation deals with the transformation of single attributes or features only, not with combining multiple features into new features.

3.2.4.2.1 Discretization: Discretization can refer to both the conversion of non-numerical data to numbers, and the conversion of continuous-valued data into categories. When converting non-numerical data into numbers, one should be careful not to remove concepts embedded in the data, nor to add structure that does not exist in the original data. For example, if the answers to a question in a questionnaire are “seldom”, “sometimes”, “often”, “always” these answers are an ordered description of frequency. The numerical representation of this attribute should retain this order.

When discretization is applied to attributes that already have numerical values, the transformation is about reducing the number of different values. This is usually done by assigning all values within an interval to a category. The problem hereby is how to chose the interval borders and also the number of categorical values for the discretization.

3.2.4.2.2 Smoothing: Smoothing means removing noise from the data. Noise is usually due to measurement error. The main smoothing techniques employed in machine learning include binning, regression, and clustering. In binning all values within a range of values (the bin) are assigned a single value form the bin. Regression contains the assumption that the measurement error is distributed randomly around the “true” value to be measured. The data is fitted to a function which then no longer contains the error noise. Further classification is then done using the function output for the attribute instead of the original noise attribute values.

3.2.4.2.3 Aggregation: Aggregation refers to the summarization of data. This can serve to reduce the amount of data to more manageable proportions, but can also be a form of smoothing. A classic example is daily sales data, which is quite noisy due to the vagaries of life: most retail shops see large sales jumps on Saturday, while Sundays and Mondays see no or little sales. Aggregation of the data to a weekly level removes these vagaries for the most part. Also, data structures that are beneficial for classification are sometimes more easily spotted by classifiers in aggregated data and hence lead to better classification.

3.2.4.3 Data normalization

Classifiers like neural networks and *k*-NN are known to perform better if all attributes are scaled to the same range of values [77]. If one or more attributes have values with a much wider range, these attributes

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

will dominate in a naive (Euclidean) distance calculation and this will negatively effect the effectiveness of the classifier. Two well-known methods to conduct this scaling are min-max normalization and *Z*-score normalization.

Min-max normalization transforms each attribute to lie within the range $[a,b]$ using linear scaling: first the minimum value a_{min} and maximum value a_{max} of the attribute are determined. For each instance the attribute's scaled value a'_i is set equal to a plus b times the ratio of (i) its original numerical value a_i minus the minimum value a_{min}, divided by (ii) the difference between the maximum value a_{max} and minimum value a_{min}:

$$a'_i = a + b\,\frac{a_i - a_{\text{min}}}{a_{\text{max}} - a_{\text{min}}} \tag{3.21}$$

Z-score normalization transforms each attribute not to lie in a fixed range, but such that the mean value of the transformed attributes is 0 with a standard deviation of 1. This is done by first determining the mean mean(a) and standard deviation stdev(a) of the original attribute values a_i. Then the transformed value is calculated by:

$$a'_i = \frac{a_i - \text{mean}(a)}{\text{stdev}(a)} \tag{3.22}$$

3.2.5 Performance measures for classification

In order for a learning algorithm to improve in its task based on experience, a metric is needed to measure the performance on that task. In the case of the classification problem, the learning algorithm is the classifier as defined in section 3.2.3. The performance measure for classification is defined by two components: the actual metric used and the way the dataset is divided. Both components are discussed in separate sections below.

3.2.5.1 Performance metrics

The simplest metric for the performance of a classifier is the *classification accuracy*, the fraction of a given test set of labeled instances that is correctly classified by the classifier, defined as:

$$Accuracy = \frac{\text{\# instances classified correctly}}{\text{\# instances in test set}} \tag{3.23}$$

The classification problem contains a dataset of examples that is used to train the classifier. Because no other data is available, this same dataset must also be used to determine the classification accuracy. This leads to two problems: what data from the dataset to use to train the classifier, and what data to use to determine the classification accuracy. If the same or overlapping data is used for both purposes, the classification accuracy is not a fair estimate of the classifier's performance. Hence the dataset of examples needs to be split into two disjoint parts: one for training the classifier and one for generalization. The latter part is the test set mentioned in equation (3.23).

Other measures for the classification performance can be defined for binary classification (problems with only two output classes with one class called "positive" and one called "negative"). In binary classification accurately classified instances can be split into true positives and true negatives, while misclassified instances can be split into false positives (incorrectly labeled positive) and false negatives

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

(incorrectly labeled negative). These other measures are precision, i.e.

$$Precision = \frac{\text{\# true positives}}{\text{\# true positives + \# false positives}}, \tag{3.24}$$

sensitivity (true positive rate or recall rate), i.e.

$$Sensitivity = \frac{\text{\# true positives}}{\text{\# true positives + \# false negatives}}, \tag{3.25}$$

and specificity (true negative rate), i.e.

$$Specificy = \frac{\text{\# true negatives}}{\text{\# true negatives + \# false positives}}. \tag{3.26}$$

These measures can also be combined into even more measures like balanced accuracy, informedness, and F-score [114].

Another way of combining these measures is by a method called Receiver Operating Characteristics (ROC) analysis [34]. ROC analysis consists of constructing a plot of the true positive rate (tp rate) on the y axis and the false positive rate (fp rate) on the x axis. The performance metric is the area under the ROC-curve (AUC): starting at the bottom left, visiting all outcomes (tp rate, fp rate) measured for the classifier and ending in the top right corner. A random classification would result in an AUC of 0.5, while perfect classification yields an AUC of 1.0.

The classification accuracy for a classifier on a specific dataset can be determined simply by following algorithm 4. This process is the same for the other metrics mentioned above.

Algorithm 4: Performance measurement for classification task: single calculation

```
Consider a classification task with dataset $D$ and chosen classifier $C$ ;
Let $f$ be the performance metric chosen to evaluate $C$ ;
Randomly split the dataset, $D$, into a training set $L$ and test set $T = D \backslash L$ ;
Train classifier $C$ on training set $L$ ;
Determine $f(T;C)$, the performance of classifier $C$ on $T$ using metric $f$.
```

3.2.5.2 Cross validation

The result of calculating the classification accuracy following algorithm 4 from the previous section is that the outcome is dependent on *how* the dataset, D, is split into a training set and test set. The simplest way to do this is by defining a single, fixed, training and test set and perform all required classification accuracy calculations using the same training and test set. The downside of this approach is that the resulting accuracy depends strongly on the chosen split between training and testing data, and this split may not be representative of the classifier's behavior on the dataset in general: a different choice of training set and test set could lead to a different accuracy.

A better method (i.e. one that better represents the classifier's performance on the dataset in general) is to divide the dataset into training and test sets multiple times, calculate the accuracy on each such split and determine the average accuracy over all splits. This was already recognized in the work of Mosier [100]. The problem of determining the classification accuracy of a classifier on a particular dataset can

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

then be refined as finding a way to estimate the *general* classification accuracy of the classifier on the dataset, which in theory should be independent of the actual split of the dataset into training and test data. Cross validation is a method to get an estimate of this general classification accuracy by calculating the accuracy as an average accuracy over multiple splits of the data. This process is outlined in algorithm 5.

Algorithm 5: Performance measurement for classification task: cross validation

```
Consider a classification task with dataset $D$ and chosen classifier $C$ ;
Let $f$ be the performance metric chosen to evaluate $C$ ;
Let $CV$ be the chosen cross validation method to split the dataset ;
for $i = 1, \dots, k$ do
    Using $CV$ divide $D$ into multiple training sets $L_i$ and testing sets $T_i = D \backslash L_i$ ;
    Train classifier $C$ on training set $L_i$ ;
    Determine $f(T_i, C)$, the performance of classifier $C$ on testing set $T_i$ using metric $f$ ;
end
Set $f(T, C)$, the final performance measure, as the average over all $f(T_i, C)$.
```

Various ways exist for the cross validation to split the dataset into training and test sets:

- *Repeated random sub-sampling validation* repeatedly splits the dataset at random into a training set and a test set, with the proportion of the dataset assigned to each determined beforehand. The number of times this random split is repeated can be chosen arbitrarily.
- *k-fold cross validation* means splitting the dataset into k equally sized (as best as possible) subsets. Each of the k subsets is used as the test set once, and in each such case the remaining $k-1$ subsets are combined to form the training set. So for a dataset of n instances, a total of k splits of the data are made (where $k \leq n$).
- *Leave one out cross validation* (LOOCV) uses each instance in the data as test set once, and in each such case the remainder of the data is used as the training set. So for a dataset consisting of n instances, a total of n splits of the data are made. This can be seen as a special case of k-fold cross validation with $k = n$.

3.3 Definition of the Feature Selection Problem

The FSP, also known as the variable subset selection problem or feature reduction problem, arises in machine learning where algorithms are used to classify data. The classification problem underlying the FSP is a supervised learning problem in which the task is to determine in which discrete *class* a given *instance* of data is to be classified. Classification was discussed in section 3.2. If the underlying classification problem has a large number of features, this causes problems for the classification algorithms.

Intuitively it seems that if more information is available in the way of more features, a better classifier can be build. However, two intertwined problems surface: firstly, classifiers will tend to *overfit* the training set if the number of features is large compared to the number of instances available for training: the classifier picks up idiosyncrasies in the training set that may not be generally applicable. It can be that many features are redundant, irrelevant or too noisy for practical use for the classification task at hand

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[46]. Instances outside of the training set are then often misclassified, leading to a poor performance on the classification task. A second problem is that the classifier becomes much *slower* with an increasing number of features. These two problems are related in that the problem of overfitting can be addressed by increasing the number of training instances, which then causes the classifier to become slower.

The two problems of overfitting and slowness can be addressed by *dimensionality reduction* in which the aim is to reduce the number of features used to perform the underlying classification task. Dimensionality reduction comes in two flavors: feature extraction (FE) and feature selection (FS). FE defines new features that are combinations or transformations of the original features. In FS no new features are generated, but instead the optimum subset of features is selected from the original set of features such that, if used, the underlying classification problem can be best solved. In a very strict sense, FS can be seen as a form of FE without transformations and all combinations of features are binary: each feature is either in or out.

Somol *et al.* [131] mentioned that the benefit of using FS over FE is that if the resulting classification algorithm is used in real life to classify new inputs, only those features selected need to be measured and collected in order to classify the data. This is especially useful in application of medical diagnosis where data collection can be costly and time consuming. A second benefit of FS over FE can be that the features retain their original interpretation, whereas the transformed features generated by FE may not have a clear physical meaning. This interpretation can be important to understand the process underlying the classification problem.

Transformed features generated by FE may provide a better discriminative ability than the best subset of given features, leading to a better accuracy in solving the underlying classification problem. Generic feature construction methods include clustering; basic linear transforms of the input variables like principle component analysis (PCA) or linear discriminant analysis (LDA); and also spectral transforms (e.g. Fourier, Hadamard), wavelet transforms, or convolutions of kernels [46]. In the remainder of this thesis FE will not be studied further, but the focus will be on FS only.

The FSP is thus the problem of identifying those features that lead to the best possible classification. Kohavi and John [70] formalize this as selecting an optimal subset of features i_k from the set of all features, I, and use only those features to train the classifier. The trained classifier then yields maximal accuracy on the original classification problem.

The FSP is mathematically defined as follows: Given a set of features, I, with n features, then $\mathscr{P}(I)$ denotes the set of all possible subsets of I. Let $J(X)$ be a fitness function that evaluates feature subset $X \in \mathscr{P}(I)$, assuming that a higher value of J indicates a better feature subset. Then the feature selection problem can be formulated to find the subset X for which

$$J(X) = max\{J(S)|S \in \mathscr{P}(I)\}. \tag{3.27}$$

Note that the definition by Kohavi and John [70] makes this mathematical definition somewhat less general by stating that $J(X)$ is the classification accuracy using the feature subset X.

Ironically, the problem that FS is supposed to solve (reducing the high number of dimensions) also affects the FSP itself: the search space grows exponentially because the number of possible subsets on a domain with n features equals 2^n. Thus, a brute force approach to solving the FSP is impractical.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Furthermore, the FSP is known to be an NP-hard optimization problem [3], meaning that any algorithm guaranteed to find the optimal solution to an FSP will not scale to large number of features.

The fact that the FSP is NP-hard also makes it an interesting problem to address using stochastic optimization methods like PSO. Since the FSP can also be formulated using mathematical sets, it is a potential test-bed for the SBPSO algorithm described in chapter 4. The next section performs a review of earlier attempts to solve the FSP from literature.

3.4 Literature on solving the Feature Selection Problem

This section gives a brief overview of different approaches from literature to solve the FSP. The search space for FSP with $|I| = N$ has 2^N points, making exhaustive search infeasible for large feature sets. Actually, Amaldi and Kann [3] have proved the FSP to be NP-hard. Although accelerated methods exist that guarantee that an optimal solution is found, like methods based on the branch and bound principle proposed by Narendra and Fukunaga [101], these are still too slow for large feature sets. The focus of research has thus been on sub-optimal methods that are not guaranteed to find the optimal solution, but that are able to find a "good" solution in a reasonable amount of time.

Heuristic methods using search for finding feature subsets were one of the first approaches to solve the FSP. Sequential backward elimination, sometimes called sequential backward selection, was introduced by Marill and Green [93]. Kittler *et al.* [69] generalized the different heuristic variants including forward methods, stepwise methods, and "plus q take away r". Cover and Van Campenhout [25] showed, however, that even for multivariate normally distributed features, no greedy search algorithm that uses a monotonic measure and that selects one feature at a time can find the optimal feature subset of a desired size. Conversely, the focus of feature selection no longer lies with these heuristic methods.

Guyon and Elisseeff [46] stated that approaches to solving the FSP can also be split into three main groups: filter methods, wrapper methods, and embedded methods. Filters select subsets of variables as a pre-processing step, independently of the chosen classifier. Wrappers consider the classifier and the underlying classification problem as a black box and scores subsets of variables directly by their performance on the underlying classification problem. Wrapper methods are thus also defined independently of the chosen classifier. Embedded methods perform variable selection *during* the training of the classifier and are usually specific to a given classifier. Each of these three main approaches is detailed below in a separate section with references given to published work on specific implementations of the general approach. After this review of the main methods, section 3.4.4 focuses on research that uses PSO in solving the FSP.

3.4.1 Filter methods

Filter methods are based on performance evaluation functions calculated directly from the training data such as distance, information, dependency, and consistency, and select features subsets without involving any specific classifier. Filters can thus be seen as part of the data pre-processing, with the goal of removing those features that are considered redundant. The crux lies in the fact that "redundant" here has no direct link with the classification process. The premise is that the performance evaluation function is

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

able to identify those features that are relevant in classifying the data.

The flow of a filter algorithm for feature selection is to start with the full set of features. The filter method is then used to select the feature subset. This selected feature subset is fed into a classifier to perform the classification task underlying the FSP. The performance on the underlying classification task is then used to evaluate the performance of the filter method for FS.

Two major approaches exist in feature selection using filters: univariate or multivariate filters [11], each discussed in more detail below.

3.4.1.1 Univariate filter methods

Univariate filtering use individual evaluation of features. This approach is also known as *feature ranking* and assesses individual features by assigning them weights according to their degrees of relevance to solving the underlying classification problem. Once the features are ranked, a method is needed to decide a cut off point: all features ranked above the cut-off are included in the feature subset, all those ranked below are excluded. The method to determine this cut-off is called the *filter*. The heart of the method lies in the metric used to determine each feature's relevance. Relevance metrics that have been used are for example χ^2, t-test, information gain [121], or Wilcoxon rank sum [136].

Strong points of univariate filtering are its speed and scalability and the fact that filter methods are independent of the classifier used in the subsequent classification. This independence is also its main weakness - the filtering does not take the peculiarities of the classifier into account and the resulting subset of features may well not be optimal for the classifier used. Also, the relation between features is completely disregarded, and there is a risk that features that are individually considered less important may jointly be of paramount importance. A problem related to the fact that feature selection is disconnected from the classification, is that the threshold levels for the ranking or outright choice of the number of features to select are chosen without reference to the classifier.

Two well-known univariate filter algorithms are:

- The FOCUS algorithm originally designed by Almuallim and Dietterich [2] for boolean domains. FOCUS exhaustively searches the space of feature subsets until it finds the minimum combination of features that divides the training data into single classes. This is referred to as the *min-features bias*. Following feature selection, the final feature subset is passed to a decision tree classifier.

- The RELIEF algorithm designed by Kira and Rendell [68], which assigns a *relevance weight* to each feature, which is meant to denote the relevance of the feature to the target concept. The algorithm is randomized: it samples instances randomly from the training set and updates the relevance values based on the difference between the selected instance and the two nearest instances of the same and opposite class. RELIEF is therefore an instance based filter method for feature selection. Kononenko [74] later extended the method to ReliefF, which can handle multi-class problems and is robust and can handle incomplete and noisy data.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

3.4.1.2 Multivariate filter methods

A common disadvantage of univariate filter methods is that they ignore the interaction with the classifier as the search in the feature subset space is separated from the classification task. This means that each feature is considered separately, thereby ignoring feature dependencies, which may lead to worse classification performance when compared to other types of feature selection techniques [121]. In order to overcome the problem of ignoring feature dependencies, a number of multivariate filter techniques were introduced, aiming at the incorporation of feature dependencies to some degree. Multivariate filtering thus considers subsets of features together, but still as a form of pre-processing and without reference to any classifier.

The crux of multivariate filter methods lies in how subsets of features are scored on relevance for the underlying classification task and redundancy (i.e. roughly containing the same information) with other features. A popular choice was to look at pair-wise comparisons to identify redundancy using measures like Pearson correlation [98], Fishers criterion [38], or the t-test criterion [138]. These methods all look at pairs of features and can therefore become computationally impractical with large numbers of features, reducing one of the core strengths of filter techniques. Therefore other approaches tried to directly select the feature subset based on data information, but without resorting to exhaustive pairwise comparisons. Market blanket theory is one such approach [71] which is referred to in more detail in the examples below.

Strong points of multivariate filtering are the fact that filter methods are independent of the classifier used in the subsequent classification and that feature interactions can be taken into account - a benefit over univariate filtering. Multivariate filtering is slower and less scalable than univariate methods, but still has better computational complexity than wrapper methods. The main weakness of univariate filtering is that for all filtering approaches, the relation between features is completely disregarded and the choice of the number of features to select contains some subjective choices uncoupled from the classifier [121].

Some examples of multivariate filter algorithms are:

- *Correlation-based Feature Selection* (CFS) was introduced by Hall [49]. CFS is a filter method that uses correlation measures to rank the features. It assumes that useful feature subsets contain features that are predictive of the class but uncorrelated with one another. CFS computes a heuristic measure of the merit of a feature subset from pair-wise feature correlations and a formula adapted from test theory. Heuristic search is used to traverse the space of feature subsets in reasonable time; the subset with the highest merit found during the search is reported.

- *Markov blanket filter* (MBF) is a filter method for feature selection proposed by Koller and Sahami [71] based on information theory. The method uses cross-entropy to minimize the amount of predictive information lost during feature selection. The Markov blanket criterion removes features if and only if it is unnecessary. MBF can be used in both a forward selection and backward elimination setup. The algorithm was tested on six datasets from the UCI Machine Learning repository, i.e. Corral, LED24, Vote, DNA, Reuters1, and Reuters2. These experiments showed limited gains in classification accuracy compared to classification on the full datasets (i.e. without feature selection) using either a naive Bayes classifier or a C4.5 decision tree classifier.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- *Fast Correlation-Based Filter* (FCBF) is a filter method proposed by Yu and Liu [154]. Underlying this method is the concept of predominant correlation, which can identify relevant features as well as redundancy among relevant features without pairwise correlation analysis. The authors performed experiments using FCBF on 10 datasets with the number features ranging from 57 to 650 and compared the average classification accuracy to three other filter methods (ReliefF, CorrSF, ConsSF) using either a naive Bayes classifier or a C4.5 decision tree classifier. No statistical analysis was performed, but accuracy was comparable to the other filter methods, while computation time was 8 to 100 times shorter.

3.4.2 Wrapper methods

Wrappers use the classifier to score subsets of variables according to their classification power. Wrapper methods were first proposed by [59]. These methods require one predetermined classifier and use its estimated performance as a scoring function to determine the quality of the selected subset of features. They attempt to find features better suited to the classifier aiming to improve its performance. Note that wrapper methods select feature subsets optimized for the given classifier and the same subset may perform worse for a different classifier.

According to Guyon and Elisseeff [46] a wrapper method for FS is defined by three choices:

1. how the space of all possible feature subsets is searched,
2. how the performance of a classifier which guides and halts the search is assessed, and
3. which classifier is used.

Often-used strategies to search the space of all possible feature subsets include best-first, branch-and-bound, simulated annealing [121], but population based algorithms have also been popular, for example genetic algorithms (GA) [58, 140], genetic programming (GP) [116, 123], ant colony optimization (ACO) [134, 151], and PSO. Studies that used the PSO to solve the FSP are reviewed in more depth in section 3.4.4 below.

The performance of the classifier (fitness function) is usually measured as the classification accuracy over a single validation set or using cross-validation. But other performance measures like those mentioned in section 3.2.5.1 have also been used in practice. Additionally, the number of features (to be minimized) is sometimes added to the fitness function [59], especially for problems where it is assumed that a small number of features is preferred, or this reduction in the number of features is a goal in and of itself.

Wrapper methods can be used in combination with any classifier. Section 3.2.3 lists possible choices for classifiers. Popular classifiers include decision trees, naive Bayes, least-square linear predictors, and SVM [46].

The flow of a wrapper algorithm for feature selection is to start with the full set of features. This is fed into a recurring loop whereby the wrapper method selects candidate feature subsets. These subsets are then passed onto a classifier and the performance of this classifier on the dataset drives the choice for the subsequent subsets to be evaluated in the wrapper. At some point the loop is exited, resulting in the

© University of Pretoria

final subset of features as solution for the FSP. This subset of features is then used by the classifier to perform the final classification.

In general, wrapper methods achieve better performance than filter methods as they can take the idiosyncrasies of the underlying classifier into account when selecting the optimal subset of features. However, wrapper methods tends to be more computationally expensive than the filter approach, especially if the number of features and number of instances in the dataset grows. Also, wrapper methods run the risk of overfitting, in that the feature subset they select is only optimal for the classifier used and this subset may be a poor choice when using a different classifier [121].

Besides the studies already mentioned in this section, a few other approaches to wrapper methods for FSP should be mentioned:

- John *et al.* [59] introduced the concept of a wrapper. First three concepts were defined: irrelevance and two different degrees of relevance. Then the three parts of the wrapper were chosen and applied in experiments on three artificial and three real world datasets:
  - To search the space of feature subsets, various search algorithms were used, i.e. backward elimination, forward selection, and a deterministic version of the Relief method;
  - To assess the classification accuracy of subsets of features, 25-fold cross validation was used; and
  - To classify the data instances, both ID3 and C4.5 classifiers were used.
- Langley and Sage [81] combined the concept of wrapper methods with a naive Bayes classifier to form the selective Bayesian classifier (SBC). To search the space of feature subsets sequential forward selection was used. Subsets of features were assessed on the average classification accuracy over 20 random splits of the dataset. The SBC was compared to the a naive Bayesian classifier and C4.5 both using all features. The experiments were performed on six datasets from the UCI machine learning repository, with three datasets selected where naive Bayes generally outperforms C4.5, and three with the reverse bias. The feature selection in SBC was successful in that the SBC significantly outperformed naive Bayes on those datasets were Bayesian classifier usually perform less well.
- Sequential search was used by Inza *et al.* [56] in their wrapper method for feature selection on two well-known DNA microarray datasets. They compared their method with a group of different filter metrics. The wrapper method was tested using different supervised classifiers, namely a C4.5 decision tree, simple naive Bayes, and a SVM. The wrapper approach showed a more accurate behavior than filter metrics, but it required significantly more computation time.

3.4.3 Embedded methods

Embedded methods try to combine the advantages of the speed of filter methods and the way that wrapper methods optimize their performance linked to the underlying classifier. They do this by integrating the feature selection process into the classifier training. Training the classifier and selecting features is thus

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

one inseparable learning process, that may be looked upon as a special form of wrappers. Embedded methods are thus usually specific to given classifiers.

The CART algorithm introduced by Breiman *et al.* [14] is a decision tree classifier that has feature selection built into it, and can thus be seen as an early version of embedded methods. According to Guyon and Elisseeff [46], embedded methods can be divided into two sub-classes: nested feature subsets and direct objective optimization.

Nested feature subset methods search the space of feature subsets by estimating changes in the performance measure (this can be classification accuracy or other metrics) value incurred by making moves in that space. These estimates are combined with greedy search strategies such as backward elimination or forward selection to yield nested subsets of features. The search strategy which estimates changes in the performance measure are a crucial part of this method. Strategies employed are finite difference calculation [133], quadratic approximation of the cost function [83], and sensitivity of the performance measure calculation [113].

In direct objective optimization the objective function to be optimized directly combines the classification and the feature selection. Generally, the objective function consists of two terms that compete with each other: the goodness-of-fit (to be maximized), and the number of variables (to be minimized). The number of variables can be minimized using so-called shrinking regularizers, which enforce a predetermined maximum number of features. An important point to note is thus that these methods require a choice for the maximum number of features to be selected. These shrinking regularizers can be effectively combined into training some classifier. This was done for linear predictors by Weston *et al.* [148] and for SVM classifiers by Boser *et al.* [12].

The flow of an embedded method starts with the full set of features. This is fed into a single loop whereby the embedded method performs a joint estimation of the classifier model (training) while reducing the number of selected features used in training. Note that embedded methods thus work well with classifiers such as linear predictors and SVMs that are built up in discrete steps for each of the feature dimensions. At each such step the chosen feature can be eliminated by the embedded feature selection method. The resulting classifier is already trained on the selected features. Only at this point is the final performance measure for the classification determined.

Embedded methods offer computational performance competitive to wrappers, enable faster learning, but produce results tightly coupled with a particular classifier. In order to use a different classifier, the learning process needs to be rebuilt to a certain extent. Also, the need to chose either the number of features to select or a threshold that indirectly defines this number as an input is a weakness that embedded methods share with filter methods. Embedded methods are able to take the relations between multiple features into account in the feature selection process [121].

Other examples of embedded methods for FSP can be found in the work of Díaz-Uriarte and De Andres [27] who embedded FS into a random forest classifier, Guyon *et al.* [47] who embedded FS into the weight vector of an SVM classifier, and Ma and Huang [92] who did so for the weights in a logistic regression.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

3.4.4 PSO applied to FSP

This section focuses on the literature on the use of PSO in the domain of feature selection. A number of recent surveys have focused on the use of PSO on solving the FSP: Kothari *et al.* [75] surveyed a number of different PSO algorithms used in FS in general, while Omar *et al.* [108] reviewed nine recent studies applying PSO in feature selection for classification. Of these nine studies, six compared a PSO algorithm to other methods including rough set theory, fuzzy-rough feature selection, and GA. Omar *et al.* [108] claimed that PSO offered better feature selection leading to a higher classification accuracy, but offered no statistical proof for this statement. Xue *et al.* [150] looked at different fitness functions that can be used for PSO in FS, and included a review of applications of the binary PSO on the FSP.

A number of studies from these aforementioned surveys and other PSO methods used on the FSP are reviewed below. The studies are split into separate sections based on which FS method is used in conjunction with PSO: filters, wrappers or embedded methods.

3.4.4.1 PSO and filter methods

Cervante *et al.* [15] proposed a somewhat rare approach to using PSO to solve the FSP that uses the filter approach. Two different methods were proposed, the first of which combines the binary PSO with a filter that uses the the mutual information of each pair of features to determine relevance and redundancy of the selected feature subset. The second algorithm combines the binary PSO with a filter that uses the entropy of a subset of features to evaluate that subset's relevance and redundancy. Different weights for the relevance and redundancy in the fitness functions of the two proposed algorithms were tuned to further improve their performance in terms of the number of features and the classification accuracy. The binary PSO's parameters were 0.7298 for the inertia weight, $c1 = c2 = 1.49618$ and a maximum velocity per dimension of 6. The swarm consisted of 30 particles with a star topology, and the algorithms ran for 500 iterations. While the PSO algorithms run, the filter methods are used to determine the fitness of the particles and thus drive the PSO's search. At the end of the PSO algorithms' runs, the subsets with the highest fitness determine the features selected. These were then fed into a decision tree classifier to evaluate the classification accuracy. For each dataset, 30 independent runs were performed of the PSO algorithms. Four different datasets from the UCI machine learning repository were used in experiments to compare the algorithms, with the number of features ranging from 18 to 61 and the number of instances from 148 to 3196. The results of this well-structured study show that, after tuning, the two proposed algorithms can significantly reduce the number of features and achieve similar or even higher classification accuracy.

3.4.4.2 PSO and wrapper methods

Chuang *et al.* [19] designed the improved binary PSO (IBPSO) for use in feature selection for gene expression data. The PSO is used in a wrapper method setup for a total of 100 iterations. The k-nearest neighbor classifier with $k = 1$ is used as the classifier and LOOCV is used as cross validation method. The PSO algorithm itself is mainly a standard binary PSO with velocity clamping using a star topology, but the *gbest* fitness is reset to zero if it has been stagnant for 3 iterations. No tuning of the

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

PSO parameters is performed. Experimental results show that this method effectively simplifies feature selection and reduces the total number of features needed. The classification accuracy obtained by the IBPSO method has the highest classification accuracy in nine of the 11 datasets tested when compared to a range of classifiers that did not include feature selection. This result thus does not give any insight into the strength of the IBPSO in FS compared to other FS approaches.

Xue *et al.* [150] looked at different fitness functions for use in feature selection with PSO in a wrapper method. The PSO algorithm used was the binary PSO which was not tuned but parameters were set to standard values from literature. The swarm contained 30 particles with a star topology. A maximum of 100 iterations were performed and for each dataset 40 independent runs were performed. Three different fitness functions were compared. The first was normal classification accuracy. The second fitness function was a weighted sum of (i) the classification accuracy and (ii) the portion of features left out. For this second fitness function, the weight of the second part increased from 0.0 at the start to 0.2 at the end of the 100 iterations. The third fitness function looked only at the classification accuracy for the first 50 iterations, and used the same weighted sum approach as the second fitness function for the last 50 iterations. During these last 50 iterations the weight of the proportion of features left out was set at a constant 0.2. Experiments were performed on 10 different datasets with 13 to 617 features, 2 to 4 classes and 32 to 4400 instances. For each dataset, the instances were randomly divided with 70% serving as the training set and 30% as the test set. The classifier used was the k-nearest neighbor with k equal to 5. Although no statistical test was included, results show that the BPSO with the second and third fitness functions could not be distinguished from each other, but they both outperformed the first fitness function that only looked at accuracy. Care should be taken to interpret the results of this study that a fitness function that "rewards" feature subsets with fewer features are to be preferred. For datasets with a large number of features of which many are redundant, this approach can improve the results, but in a real life setting the optimal number of features is not known.

Tu *et al.* [137] combined the binary PSO with an SVM classifier in a wrapper method to solve the FSP. No information is given about the PSO's parameters used, nor about the swarm size. The star topology was employed for the swarm and the algorithm was run for 100 iterations on each dataset, but only a single run of the algorithm was performed in each case. The classifier used was the SVM with a radial basis function kernel. This classifier was tuned for each dataset to find optimal values for its parameters r and C. Performance was measured by classification accuracy using 10-fold cross validation. Five datasets from the UCI machine learning repository were used to test and compare the algorithms. The number of features ranged from 10 to 60 and the number of instances from 104 to 990. The PSO-SVM algorithm was compared to eight filter methods using a 1-nearest neighbor classifier from the study of Oh *et al.* [105]. For the filter methods, the number of features to select is fixed beforehand. Four different choices of the number of features to select are tested for each filter method. The PSO-SVM algorithm performed best on four of the five datasets. However, the comparison is flawed on different levels: no reason is given for why these five datasets were chosen from the 11 datasets evaluated in the study by Oh *et al.* [105]. Also, because different classifiers are used and the comparison is made on classification accuracy, little information is gained as to the contribution of the feature selection, nor of the contribution of the PSO to this process. Since only one classifier is used, this can subtly bias the resulting accuracies

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

and impact of the feature selection. Finally, simply comparing a wrapped method to filter methods is wrought with danger as the computational costs vary greatly.

Yun *et al.* [155] studied the application of the PSO feature selection and compared it against a GA in the same setup. Besides a basic wrapper approach, a second variant combining the wrapper with a filter method was also used. The filter called minimal-redundancy and maximal-relevance (mRMR) is used in the PSO to override the selection of a single feature if the filter considers it minimal-redundant, and to override a feature's exclusion if it is considered maximal-relevant. The standard binary PSO was used with parameters $c_1 = c_2 = 2.0$ and a swarm of 20 particles with a star topology. The PSO was run for 40 iterations on each dataset. Three different classifiers were used, namely naive Bayes, the C4.5 decision tree, and a SVM with a linear kernel. The performance measure used in the study was classification accuracy using 10-fold cross validation. In the experiments, 20 real-world datasets were used from the UCI machine learning repository. The number of features ranged from 19 to 649. The number of instances ranged from 32 to 6236. The basic wrapper methods, both using the GA and the PSO, outperformed the filter methods to which they were compared. The PSO wrapper outperformed the GA wrapper using each of the three classifiers in a statistically significant manner. The variants combining the wrapper with the mRMR filter performed even better. Here the PSO-mRMR outperformed the GA-mRMR using the Naive Bayes and C4.5 classifiers. Using the SVM classifier, no difference could be discerned between the PSO-mRMR and GA-mRMR. This well-built study shows how population based methods can be used in a wrapper setup and compared. It is shown that the choice of classifier is important for the outcome and that multiple classifiers should ideally be used in the comparison. A point of critique is that in each case the PSO and GA algorithms are run only once, disregarding the fact that multiple independent runs are necessary to compare algorithms with a stochastic component.

Yang *et al.* [152] adapted the binary PSO by using two different chaotic maps to form two versions of the chaotic binary PSO (CBPSO). The purpose of the chaotic maps was to determine the inertia weight of the BPSO. The CBPSO was used in a wrapper method approach to solve the FSP. The PSO algorithm used a star topology with parameters $c_1 = c_2 = 2$ and velocity clamping with maximum velocity of 6. No further details were given on the specification. The k-nearest neighbor classifier with $k = 1$ was used to determine the classification accuracy, which was measured using LOOCV. Five different datasets from the UCI machine learning repository were studied with the number of features ranging from 13 to 60 and the number of instances from 178 to 846. The CBPSO was compared to filter methods for solving the FSP, including sequential forward search with and without plus and take away, sequential forward floating search, sequential GA, and different hybrid GAs. CBPSO performed the best on four of the five datasets. The same critique as on the work by Tu *et al.* [137] applies to this study: no reason was given for why these five datasets were chosen from the 11 datasets evaluated in the study by Oh *et al.* [105]. Also, because different classifiers were used and the comparison was made on classification accuracy, little information was gained as to the contribution of the feature selection, nor of the contribution of the PSO to this process. Since only one classifier was used, this can subtly bias the resulting accuracies and impact of the feature selection. Finally, simply comparing a wrapped method to filter methods is wrought with danger as the computational costs vary greatly.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

3.4.4.3 PSO and embedded methods

Gomez *et al.* [45] introduced a two-phased wrapper based approach to solving the FSP. The heuristic search for the optimal feature subset is split into two stages, in an algorithm called the two step PSO. The PSO algorithm used is the binary PSO with parameters $c_1 = c_2 = 2$, with the inertia weight decreasing over time from 1.4 to 1.0. The swarm size was set to 21 and 120 total iterations were performed, split over the two steps. Candidate solutions from the first stage are used to help initialize the swarm for the second stage: features that are present in many of the candidate solutions are seeded into the swarm for the second phase, while features that occur in few candidate solutions are filtered out of the swarm at this mid point. The thresholds of these in- and exclusions, as well as the length of stage 1 versus stage 2 are parameters of the two step PSO algorithm. Instead of a classifier, the reduct concept of rough set theory is used. A reduct is a minimal set of features that preserves the partitioning of the dataset in classes and hence the ability to perform classifications. The FSP is thus transformed from finding the optimal subset for classification, to finding the minimal reduct. Performance was measured as the weighted sum of classification accuracy of the reduct and the proportion of features excluded from the selected feature subset: less features meaning a better performance. The accuracy was weighted by 0.54 versus 0.46 for the number of features. Experiments were performed on six datasets with 9 to 56 features and 32 to 1000 instances. The two step PSO was compared to the same PSO wrapper setup but without the two step approach and to a two step approach involving ACO. The two step PSO was able to find the smallest minimal reducts. The two step PSO algorithm, however, also introduces three parameters, in addition to the regular PSO parameters, to the process of solving the FSP with PSO.

Unler and Murat [139] proposed an embedded feature selection method using an adjusted version of the binary PSO. The algorithm contains a procedure called “adaptive feature subset selection” which constructs a feature subset by selecting features in turn until the pre-determined number of features is reached. The PSO particle’s position is a probability weighted input for the adaptive feature subset selection procedure. The binary PSO algorithm was adjusted in two ways: first, a time-decreasing inertia weight was used which decreased from 0.995 to 0.5 in a linear fashion during the PSO’s run. Secondly, an extra term was introduced to the velocity update equation with parameter c_3 for attraction to the swarm’s best performing particle in that iteration. The PSO parameters were tuned in an unspecified manner to $c_1 = 2$, $c_2 = 1.5$, $c_3 = 0.5$. A star topology was used on a swarm of 20 particles, with a maximum number of 300 iterations. The embedded PSO feature selection algorithm was run only once on each dataset. The algorithm was tested on 11 large datasets. The number of features ranges from 8 to 93, but the number of instances ranged from 351 to 581012, with most datasets having more than 4500 instances. The classifier used for the final classification was the logistic regression model. Due to the large number of instances in the datasets, the validation method for each dataset consisted of a single training set of up to 1300 instances and 10 test datasets of 200 instances chosen randomly. The performance measure was the average accuracy over these 10 independently chosen test sets. The resulting classification accuracy was compared to and exceeded the accuracy reported in an earlier study [109]. This earlier study used two filter methods as well as a tabu search combined with a logistic regression model. The embedded PSO algorithm showed slightly better results, but not statistically significant. This well-constructed study showed the potential benefit of an embedded approach using PSO for large datasets, where a wrapper

© University of Pretoria

approach may become computationally burdensome. The need to determine beforehand the correct number of features to select, however, becomes a problem in datasets with a large number of features where the optimal feature subset size is unknown.

3.5 Conclusions

The first and foremost objective of this chapter consisted in formally introducing the FSP. This was done by starting with an overview of the field of machine learning and then shifting focus to the problem of classification. This problem was defined as a supervised learning problem in which the goal is to determine the correct class label of a data instance based on its input features. An investigation of the classification problem led to a review of the concept of a classifier and a detailed discussion of three specific classifiers: the *k*-nearest neighbor classifier, the decision tree, and the Gaussian naive Bayes classifier. Also, ancillary technical issues were discussed that will play a role in later numerical experiments on the FSP: data preprocessing and how to measure performance of an algorithm on a classification task. The latter led to the introduction of various variants of classification accuracy and the use of cross validation in training and testing.

A large number of features in classification leads to the curse of dimensionality which shows itself in two intertwined ways: if the number of features grows larger, classifiers increasingly suffer from overfitting and they require an increasing number of computational resources. Feature selection was introduced as a technique that can help overcome this curse. Hence, the FSP was defined as selecting those input features in a classification problem that allow for better classification: a higher classification accuracy due to less overfitting and also faster classification.

A second objective for this chapter was to show that, similarly to what was done for the MKP in the previous chapter, the FSP can serve as a valid test problem on which to compare the performance of various PSO algorithms used to solve it. This objective naturally led to a review of the different approaches from literature used in solving the FSP. The wide range of these approaches and ongoing research into them showed that the FSP is a complex, even NP hard, optimization problem with many real world applications especially in the realm of bio-informatics.

A review of the use of PSO on the FSP showed that this is also an active area of research. The approaches using PSO on the FSP were divided along the lines of the three main approaches in solving the FSP, namely filters, wrappers, and embedded methods. The wrapper method has been the most popular way in which the PSO is used to solve the FSP. Studies that use filter or embedded methods also exist, but are less numerous. At first glance, though, each method can be used as a framework in which various PSO methods can be compared fairly. Two reasons can be given to prefer using a wrapper method with PSO to solve the FSP over filter and embedded methods, if the goal is to compare different PSOs. A first reason why a wrapper works better for this particular comparison is because a wrapper puts the biggest onus on the PSO algorithm to search the space of feature subsets. In contrast, filter and embedded methods use information measures on single or multiple features as the main driver of the search for successful feature subsets. Hence, if a difference in performance on discrete optimization problems exists between PSO algorithms, the wrapper approach should be better able to illuminate this

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

difference. Conversely, using a filter or embedded approach might not show any difference between PSO algorithms at all, as success in the search for good feature subsets is mainly driven by the filter. A second reason to prefer a wrapper is because filter and embedded methods require an a-priori choice for the number of features to be selected. This is either done explicitly, or implicitly by setting a cut-off level for the filter. This choice adds a layer of complexity to comparisons of algorithms, as the relative performance of different PSO algorithms may differ based on the number of features to be selected. It follows that a wrapper method setup is preferable when comparing the performance of different PSO algorithms using the FSP.

A separate finding is that the quality of the studies reviewed in section 3.4.4 varies: some studies are well structured and take care to perform a valid comparison of the proposed algorithm to others from literature. Other studies fall short in a number of ways, as listed below:

- A new method for feature selection using a wrapper and/or combining new concepts is compared to older methods, mainly filter methods. The comparison then only looks at classification accuracy. This approach ignores two important differences between filters and wrappers: for filters the number of features to select needs to be set manually and this is usually confined to a small number of a-priori guesses. The wrapper method is able to select any number of features during the search of feature subspace. The drawback of wrapper methods is that they require much more computation time. By only looking at the classification accuracy, the wrapper method is given an unfair advantage.

- Assume that classification accuracy is compared between algorithm A_1 used to solve the FSP with classifier C_1 specifically, and algorithm A_2 used to solve the FSP with classifier C_2. Also, assume that the combination of algorithm A_1 and classifier C_1 achieved the higher classification accuracy. Then it is impossible to determine whether it was the difference in algorithm or the difference in classifier which (primarily) caused the difference in classification accuracy. In order to fairly compare algorithms A_1 and A_2, all settings need to be kept equal for the experiments on both algorithms. The work by Yun *et al.* [155] also showed that different classifiers can lead to different relative performance on the algorithms used to solve the FSP: using classifier C_1 algorithm A_1 may outperform A_2, while using classifier C_2 algorithm A_2 performs better than algorithm A_1. It is therefore advisable to use a number of different classifiers and perform multiple comparisons of the PSO algorithms on the FSP, one for each classifier.

- The performance of two or more stochastic algorithms was compared incorrectly. Sometimes only a single run of the stochastic algorithm (which PSO is) is performed instead of multiple independent runs. In other cases algorithms are compared on a single or very few datasets only. In order to achieve statistically significant results, instead, a large enough number of datasets should be used. Even if enough different datasets are included in the comparison, it is important that proper statistical tests are performed to ensure that any difference in performance recorded is more than a random fluctuation.

Avoiding these shortcoming will help to properly structure the numerical experiments on the FSP to be conducted in chapter 6.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

This chapter concludes the first part of this thesis which laid the theoretical groundwork for the remainder of the text. This theoretical introduction consisted of the PSO algorithm in chapter 1 and the two optimization problems on which a new and various existing PSO algorithms are to be tested and compared: the MKP in chapter 2 and the FSP in this chapter. The next two parts describe new research. The first of those two parts introduces a new PSO algorithm called the set-based PSO in chapter 4. The second part of new research applies the set-based PSO and other PSO problems to the MKP and the FSP.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Part II

Generic set-based particle swarm optimization

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 4

Set-based particle swarm optimization

A number of known discrete PSO adaptions using sets were discussed in chapter 1, leading to the conclusion that a functioning, generic, set-based PSO algorithm does not yet exist. The current chapter introduces a PSO algorithm called Set-Based PSO that is intended to be such an algorithm. The objective of this chapter is therefore to define SBPSO as a generic, set-based PSO algorithm that can be applied to discrete optimization problems. The algorithm is intended to contain the features identified in chapter 1 to determine a PSO algorithm, but formulated using mathematical sets. This chapter is structured by first defining the PSO concepts of position and velocity in a universe of mathematical sets. Then set-based operators are defined that are used to combine and transform these sets. Finally, these definitions are combined in update equations for velocity and position, and the general flow of the algorithm is made explicit.

4.1 Introduction

Part I forms the theoretical background for this thesis. Chapter 1 provided an overview of the PSO field, specifically of discrete PSO algorithms, in chapter 1. Chapters 2 and 3 respectively discussed the MKP and the FSP. Those two problems were introduced to form a testbed for the set-based PSO algorithm presented in this chapter. Part II of this thesis consists of only a single chapter and contains the specification of the Set-Based PSO algorithm, which is intended as a functioning, generic, set-based PSO algorithm. A question that arises naturally is whether such an algorithm already exists. In order to answer this question, a review of the literature was presented in chapter 1. Section 1.3.2 listed a number of PSO algorithms that use mathematical sets. However, each of the existing methods reviewed fell short in one of three areas:

1. it was not truly set-based,
2. it was not a truly functioning algorithm in that it did not yield sufficiently good results on discrete optimization problems, or
3. it was not generically applicable to all discrete optimization problems, but instead contained problem specific features that severely restricted its application.

© University of Pretoria

Hence there is room for a functioning, truly set-based, generically applicable PSO algorithm and such an algorithm would be a contribution to the field of swarm intelligence. The abbreviation SBPSO is used in the remainder of this thesis for the new algorithm intended to fill this void.

Note that two papers outlining the SBPSO algorithm have already appeared in print, namely “A Generic Set-Based Particle Swarm Optimization Algorithm” [79] and a later revised version and with a much expanded analysis in “Set-based particle swarm optimization applied to the multidimensional knapsack problem” [80].

The current chapter addresses a number of the sub-objectives of this thesis identified in the preface, most directly the objective to formally define the SBPSO algorithm by formulating the PSO update equations in terms of set-theory. This is done using the basic components that make up a PSO algorithm as identified in chapter 1. On top of these basic features, a further objective is to identify if, and if so which, additional components are required to make the new algorithm a functioning algorithm. Finally, the new algorithm needs to be formulated generically to not include any problem domain specific features in the algorithm itself: the only link to the problem domain should lie in the fitness function.

This chapter is divided into three main parts, excluding this introduction and a final section of conclusions. The PSO concepts of position and velocity are defined in a universe of mathematical sets in section 4.2. The universe of discourse is defined, and elements within this universe are elements that define the position of a particle in the swarm. The discrete make up of the velocity is defined mathematically and illustrated graphically. The second part of this chapter deals with the operators that are needed to formulate position and velocity update equations in keeping with the general PSO paradigm. Six different operators are defined in section 4.3, the first four of which directly correspond to parts of the position update equation (1.2) and velocity update equation (1.4) from chapter 1:

1. the addition of two velocities,
2. the difference between two positions,
3. the multiplication of a velocity by a scalar, and
4. the addition of a velocity and a position.

Two further operators are introduced to ensure to functioning of the algorithm regardless of the initial positions and velocities by preventing that parts of the search space become inaccessible to the particles in the swarm:

1. the removal of elements from the cross section of the current position, and the personal and neighborhood best positions,
2. the addition of elements from outside the current position, and the personal and neighborhood best positions.

Section 4.4 combines the set-based position and velocity with the newly defined operators to formulate SBPSO’s position and velocity update equations. This section also contains a schematic overview of the flow of the SBPSO algorithm in solving a discrete optimization problem.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

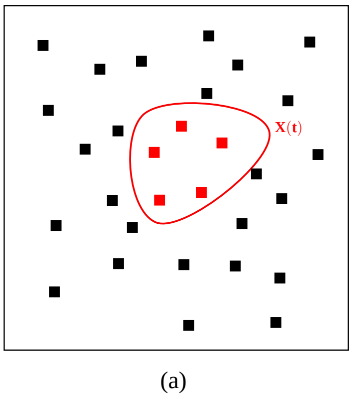
4.2 Set-based Concepts

SBPSO defines a particle's position and velocity as mathematical sets. The position is a set of elements from the universe of discourse U, that is, the universe of elements defined by the problem. The velocity is a set of *operation pairs* defined below. The solution that SBPSO finds for the optimization problem is thus the best position found by the swarm, represented as a set of elements from U.

The definitions below assume that SBPSO is applied to a maximization task, but a similar definition for a minimization task is easily derived from this. Let

- $U = \{e_n\}_{n \in N_U}$ be the universe of discourse containing all elements, e_n, of which there are a finite number N_U,
- $X_i(t)$ be the position of particle i at iteration t, a subset of U,
- $V_i(t)$ be the velocity of particle i at iteration t,
- f be the objective function to be optimized,
- $Y_i(t)$ be the personal best position of particle i, that is, $Y_i(t) = X_i(\tau)$, $\tau \in \{1, \ldots, t\}$, such that $f\big(Y_i(t)\big) = f\big(X_i(\tau)\big) = \max\{f(X_i(s)) \big| s = 1, \ldots, t\}$,
- $\widehat{Y}_i(t)$ be the neighborhood best position for particle i at iteration t, that is, $\widehat{Y}_i(t) = Y_j(t)$ for the particle j in i's neighborhood that maximizes $f\big(Y_j(t)\big)$.

Figure 4.1(a) shows a particle position $X(t)$ as a set in the universe U. This universe and mathematical sets in general do not have a spatial structure, so the placements of the elements denoted with small squares is arbitrary and no elements can be said to be close to or far away from each other.



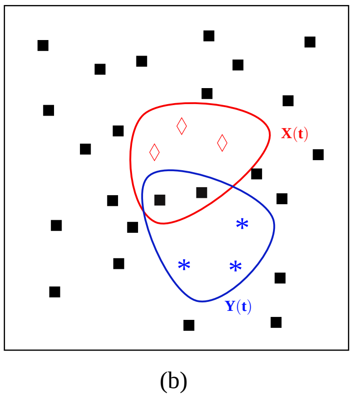


Figure 4.1: Particle positions in SBPSO: (a) shows a particle position $X(t)$ in SBPSO is a set in the universe U. The small squares represent elements in the universe U. (b) shows a particle position $X(t)$ and a particle's personal best position $Y(t)$. The open diamonds ($\diamond$) represent elements in $X(t)$ that are *not* in $Y(t)$, and the asterisks (*) represent elements in $Y(t)$ that are *not* in $X(t)$.

The PSO paradigm is built on the idea of movement through the search space, using the concept of velocity. For SBPSO this idea of movement needs to be defined. In continuous PSO, attraction of a particle to its personal best position partly determines the particle's velocity. In SBPSO the same

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

attraction to the personal best applies. Figure 4.1(b) shows a particle position $X(t)$ and personal best position $Y(t)$. Here $X(t)$ and $Y(t)$ are shown to partially overlap, though this is not necessarily true. The *movement* of $X(t)$ *towards* $Y(t)$ in SBPSO means that the two sets are made more similar by removing elements from $X(t)$ that are not in $Y(t)$ (pictured as $\diamond$), and by adding to $X(t)$ missing elements that are in $Y(t)$ (pictured as *). Elements that are in both $X(t)$ and $Y(t)$ are not affected by this attraction, nor are elements that lie outside both $X(t)$ and $Y(t)$.

The velocity is defined as a set of *operation pairs*, where an operation pair is the addition *or* deletion of a single element. An operation pair is denoted as $(\pm,e)$, with $(+,e)$ for the addition of element $e \in U$ and $(-,e)$ for the deletion of element e. The velocity of particle i, $V_i(t)$, is then written as $\{v_{i,1},\ldots,v_{i,k}\} = \{(\pm,e_{n_{i,1}}),\ldots,(\pm,e_{n_{i,k}})\}$, where k is the number of operation pairs in $V_i(t)$, and each $e_{n_{i,j}}$ is an element in U identified by the index $n_{i,j}$.

As an example, consider position $X = \{a,c\}$ and velocity $V = \{(+,b),(-,c)\}$ consisting of two operation pairs. Adding velocity V to position X means that element b is added while element c is removed, resulting in a new position, $X' = \{a,b\}$.

Attraction towards the personal best $Y(t)$ does not have to mean that the position $X(t)$ moves to the personal best position in one step such that $X(t+1) = Y(t)$. Velocity update equation (1.4) contains the attraction to $\vec{y}_i(t)$ as $c_1\vec{r}_1(t)\,[\vec{y}_i(t) - \vec{x}_i(t)]$, meaning that the difference between $y_{i,j}(t)$ and $x_{i,j}(t)$ is scaled by a factor $\gamma_j(t) = c_1 r_{1,j}(t)$, for all $j = 1,\ldots,n$. If $\gamma_j(t) = 1$, then $x_{i,j}(t+1) = y_{i,j}(t)$, if the other terms of equation (1.4) are disregarded. If $\gamma_j(t) < 1$ then $\vec{x}_i(t)$ is pulled only partly towards $\vec{y}_i(t)$ in dimension j, while if $\gamma_j(t) > 1$ then $\vec{x}_i(t)$ will overshoot $\vec{y}_i(t)$ in dimension j. In a set-based representation, this overshooting can not be defined because there is no direction in which $X(t)$ can overshoot $Y(t)$ since U has no spatial structure. In contrast, for $X(t+1) = \vec{\gamma}(t)[Y(t) - X(t)]$, the case $\vec{\gamma}(t) < 1$ can be defined in a set-based representation, by making some but not all of the changes required to turn set $X(t)$ into $Y(t)$. Figure 4.2(a) illustrates this in action, assuming $\vec{\gamma}(t) = 0.5$. The set $X(t)$ requires six changes to “move to” $Y(t)$: the three elements indicated as $\diamond$ need to be deleted from $X(t)$, and the three elements indicated as * need to be added to $X(t)$. The scaling of the “move” by a factor of 0.5 is set to mean that only three of these changes, selected randomly, are made to $X(t)$. This results in the new position, $X(t+1)$. The attraction of $X(t)$ to the particle’s neighborhood best position $\widehat{Y}(t)$ works in a similar manner.

Figure 4.2(b) shows positions $X(t), Y(t)$, and $\widehat{Y}(t)$ to partially overlap, with one common element indicated by a triangle ($\triangle$), although this does not necessarily happen in practice. However, should an element be present in all three sets $X(t), Y(t)$, and $\widehat{Y}(t)$, then the above described attraction to $Y(t)$ and $\widehat{Y}(t)$ can *not* lead to the removal of this element from $X(t)$. Also the attraction to $Y(t)$ and $\widehat{Y}(t)$ can *not* lead to the addition of any element to $X(t)$ that is outside of both $Y(t)$ and $\widehat{Y}(t)$. Such elements are indicated with symbol ‘+’ in figure 4.2(b). For both cases a mechanism needs to be included in SBPSO to ensure that the whole universe U is in theory reachable from every possible starting position[1]. These two mechanisms are defined in section 4.3.

[1] Consider a particle i in SBPSO. Because the swarm usually consists of multiple particles, movement of particles other than i can change $\widehat{Y}_i(t)$ by finding a new best candidate solution. This can then cause $\widehat{Y}_i(t)$ to contain an element e that was first outside of $X_i(t), Y_i(t)$, and $\widehat{Y}_i(t)$. So strictly speaking only elements that are outside of $X_j(t)$ and $Y_j(t)$ for *all* particles j in the swarm (and hence also outside $\widehat{Y}_j(t)$ for all j) can not be added to $X_i(t)$ by the attraction mechanism. Similarly, only an element e that is contained in $X_j(t)$ and $Y_j(t)$ for *all* particles j in the swarm is one that can not be removed by the attraction mechanism.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

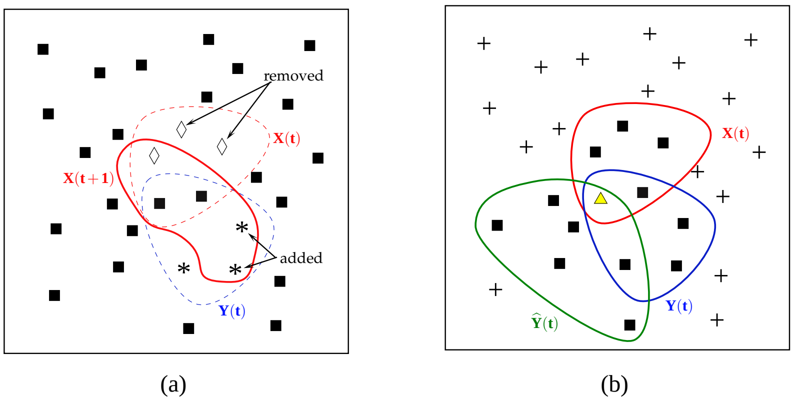


Figure 4.2: Particle attraction and movement in SBPSO: (a) shows how a particle moves from its current position $X(t)$ in direction of its personal best position $Y(t)$ to its new position $X(t+1)$, (b) shows a particle position $X(t)$, its personal best position $Y(t)$, and the neighborhood best position $\widehat{Y}(t)$.

For a strict mathematical definition of position, velocity, and objective function, denote with $\mathscr{P}(U)$ the power set (that is, the set of all subsets) of U. A position $X_i(t)$ is an element of $\mathscr{P}(U)$. The objective function f maps a position to a quality score in $\mathbb{R}$, written as $f : \mathscr{P}(U) \to \mathbb{R}$. The velocity $V_i(t)$ is generally defined as a function that maps a position to a new position, that is, $V_i(t) : \mathscr{P}(U) \to \mathscr{P}(U)$.

Note that the definition of velocity using operation pairs is narrower than the general mapping, $V : \mathscr{P}(U) \to \mathscr{P}(U)$. Consider for example $U = \{0,1\}$, and mapping V such that

1. $V(\emptyset) = \emptyset$ (V can not contain any additions),
2. $V(U) = U$ (V can not contain any deletions),
3. $V(\{0\}) = \{1\}$ (requires one addition and one deletion), and
4. $V(\{1\}) = \{0\}$ (requires one addition and one deletion).

Then, V is a valid mapping from $\mathscr{P}(U)$ to $\mathscr{P}(U)$ that can not be denoted as a set of additions and deletions.

4.3 Operators

To describe SBPSO mathematically, new operators need to be defined. These operators act on velocities (sets of operation pairs) and positions (sets of elements from U) in ways equivalent to the additions, subtractions and scalings of positions and velocities in the canonical continuous PSO algorithm. Two special operators that ensure SBPSO can reach the entire search space are defined to allow (i) adding elements to a particle's position that are not in the personal best $Y_i(t)$ nor in the neighborhood best $\widehat{Y}_i(t)$, and (ii) removing elements from a particle's position that are present in $X_i(t)$ as well as both $Y_i(t)$ and $\widehat{Y}_i(t)$.

The addition of two velocities, $V_1 \oplus V_2$, is a mapping $\oplus : \mathscr{P}(\{+,-\} \times U)^2 \to \mathscr{P}(\{+,-\} \times U)$, that takes two velocities as input and yields a new velocity. Denoted as $V_1 \oplus V_2$, the mapping is defined as the

© University of Pretoria

simple union of the two sets of operation pairs:

$$V_1 \oplus V_2 = V_1 \cup V_2. \tag{4.1}$$

The difference between two positions, $X_1 \ominus X_2$, is a mapping $\ominus : \mathscr{P}(U)^2 \to \mathscr{P}(\{+,-\} \times U)$, that takes two positions as input and yields a velocity. If a particle moves by the resulting velocity, the difference between the two positions X_1 and X_2 is the "distance" that is traversed in one step. This mapping is defined as a set of velocity pairs that indicate the steps required to convert X_2 into X_1 using additions and removals of single elements:

$$X_1 \ominus X_2 = \big(\{+\} \times (X_1 \backslash X_2)\big) \cup \big(\{-\} \times (X_2 \backslash X_1)\big). \tag{4.2}$$

Therefore, $X_1 \ominus X_2$ is the union of (i) the product of $\{+\}$ and all elements in X_1 not in X_2 (all such elements are added) and (ii) the product of $\{-\}$ and all elements in X_2 not in X_1 (all such elements are removed). This operator thus yields the velocity V to get from X_2 to X_1.

The multiplication of a velocity by a scalar, $\eta \otimes V$, is a mapping $\otimes : [0,1] \times \mathscr{P}(\{+,-\} \times U) \to \mathscr{P}(\{+,-\} \times U)$ that takes a scalar and a velocity and yields a velocity. The mapping is defined to mean picking a subset of $\lfloor \eta \times |V| \rfloor$ elements at random from velocity V to yield a new velocity. Here $\lfloor x \rfloor$ for $x \in \mathbb{R}^+$ denotes the largest $v \in \mathbb{N}$ for which $x \geq v$. The operand η is restricted to values in $[0,1]$ since sets can not have a negative number of elements and sets do not allow multiple instances of the same element. Note that $0 \otimes V = \emptyset$ and $1 \otimes V = V$.

The addition of a velocity and a position, $X \boxplus V$, is a mapping $\boxplus : \mathscr{P}(U) \times \mathscr{P}(\{+,-\} \times U) \to \mathscr{P}(U)$ that takes a position and a velocity and yields a position. Recall that a velocity is itself a function that maps a position to a new position. The operator $\boxplus$ is defined as the action of applying the velocity function V to the position X:

$$X \boxplus V = V(X) \tag{4.3}$$

This is further specified as applying the full set of operation pairs $V = \{v_1, \ldots v_n\}$ to the position X one-by-one and, for each operation pair, one element is added to X or removed from X.

Section 4.2 referred to two special mechanisms to remove elements from $X(t)$ that are in $X(t) \cap Y(t) \cap \widehat{Y}(t)$ and to add elements to $X(t)$ from outside of $X(t) \cup Y(t) \cup \widehat{Y}(t)$. These mechanisms are explained below.

The removal of elements in $X(t) \cap Y(t) \cap \widehat{Y}(t)$ from a position $X(t)$ uses the operator $\odot^-$. Denoted $\beta \odot^- S$, where S is shorthand for the set of elements $X(t) \cap Y(t) \cap \widehat{Y}(t)$, this is a mapping $\odot^- : [0, |S|] \times \mathscr{P}(U) \to \mathscr{P}(\{+,-\} \times U)$, which takes a scalar and a set of elements, and yields a velocity. The operator $\odot^-$ is implemented as *randomly selecting* a number of elements determined by β from S to remove from $X(t)$ and constructs operation pairs that are deletions:

$$\beta \odot^- S = \{-\} \times \tfrac{N_{\beta,S}}{|S|} \otimes S) \tag{4.4}$$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The number of elements that are selected from S is denoted by $N_{\beta,S}$, and defined as

$$N_{\beta,S} = \min\{|S|, \lfloor\beta\rfloor + \mathbb{1}_{\{r<\beta-\lfloor\beta\rfloor\}}\} \tag{4.5}$$

for a random number $r \sim U(0,1)$. Here $\mathbb{1}_{\{bool\}}$ is the indicator function with $\mathbb{1}_{\{bool\}} = 1$ if $bool = true$ and $\mathbb{1}_{\{bool\}} = 0$ if $bool = false$. Thus the number of elements selected is at least $\lfloor\beta\rfloor$, and the fractional remainder $\beta - \lfloor\beta\rfloor$ is the probability of the number of elements selected being one larger. The number of elements is also capped at the number of elements in S, which in turn means that β is also capped at the number of elements in S.

The choice is made to *randomly* select elements from S instead of spending more computational effort to select good candidate elements for removal from $X(t)$. Note that the aim of this operation is to allow exploration of the entire search space. It will likely lead to a worse objective function value at present, as the element removed from $X(t)$ is likely of "good quality" given that it is included in both the personal best and the neighborhood best. The assumption is that any extra effort to select a better element to remove from $X(t)$ will yield only a limited return above that from random selection.

The addition of elements outside of $X(t) \cup Y(t) \cup \widehat{Y}(t)$ to $X(t)$ uses the operator $\odot^+$. Denoted $\beta \odot^+ A$, where A is shorthand for the set of elements $U \backslash \big(X(t) \cup Y(t) \cup \widehat{Y}(t)\big)$, this is a mapping $\odot^+ : [0,|A|] \times \mathscr{P}(U) \to \mathscr{P}(\{+,-\} \times U)$, which takes a scalar and a set of elements, and yields a velocity. The operator $\odot^+$ is implemented to use *marginal objective function* information for the position $X(t)$ to choose which elements from A to add to $X(t)$, and constructs operation pairs that are additions. The marginal objective function value of element e for a particle with position $X(t)$ is defined as the objective function value of a new particle with position equal to $X(t)$ *plus* e, that is, $X(t) \cup \{e\}$. A k-tournament selection algorithm incorporating this marginal objective function information is used to select elements to add to $X(t)$, as outlined in algorithm 6. The implementation of the operator $\odot^+$ thus depends on the parameter k used in the tournament selection, and is denoted as $\odot_k^+$. The operator $\odot_k^+$ thus is defined as

$$\beta \odot_k^+ A = \{+\} \times k\text{-Tournament Selection}(A, N_{\beta,A}) \tag{4.6}$$

where $N_{\beta,A}$, the number of elements to be added to $X(t)$, is defined as in equation (4.5). The number of elements to be added is capped at the number of elements in A, which in turn means that β is also capped at the number of elements in A.

Algorithm 6: k-Tournament Selection(A,N)

```
Set $V_{temp} = \emptyset$ ;
for $n = 1,\ldots,N$ do
    for $j = 1,\ldots,k$ do
        Randomly select $e_j$ from $A$;
        Set $score_j = f\big(X(t) \cup \{e_j\}\big)$;
    end
    Select $m \in \{1,\ldots,k\}$ such that $score_m = \max_j\{score_j\}$;
    Set $V_{temp} = V_{temp} \oplus (\{+\} \times e_m)$;
end
Return $V_{temp}$;
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

In summary, $\beta \odot_k^+ A$ means selecting $N_{\beta,A}$, possibly overlapping, elements $\{e_j\}_{j=1}^{N_{\beta,A}}$, where each element e_j in turn is the best performing in a tournament of k elements selected randomly from A. The best performing element e' here means maximizing the objective function value of $X_i \cup \{e'\}$. Note that a higher value of β leads to more elements from A being added to the position $X(t)$, while a higher value of k means the algorithm is more greedy in selecting which elements to add.

Extra computational effort is exerted in SBPSO by using the k-tournament selection to find a "good" element to add to $X(t)$: an additional k objective function evaluations are required. This is done because the set A will, in general, contain many elements that lead to a worse objective function value when added to $X(t)$. Good elements to add to $X(t)$ will thus tend to be rare. The assumption made in this paper is that the extra effort to locate these good elements is worth the extra objective function evaluations.

4.4 Update equations

Using the redefined operators from section 4.3, the velocity update equation for SBPSO becomes

$$\begin{aligned} V_i(t+1) &= c_1r_1 \otimes \big(Y_i(t) \ominus X_i(t)\big) \oplus c_2r_2 \otimes \big(\widehat{Y}_i(t) \ominus X_i(t)\big) \\ &\quad \oplus \big(c_3r_3 \odot_k^+ A_i(t)\big) \oplus \big(c_4r_4 \odot^- S_i(t)\big) \end{aligned} \tag{4.7}$$

where $A_i(t) = U \backslash \big(X_i(t) \cup Y_i(t) \cup \widehat{Y}_i(t)\big)$ and $S_i(t) = X_i(t) \cap Y_i(t) \cap \widehat{Y}_i(t)$. Parameters are $c_1, c_2 \in [0,1]$, $c_3, c_4 \in [0, |U|]$, and the random numbers r_i are all independently drawn from the uniform distribution on $(0,1)$. Besides the additional velocity components involving $\odot^-$ and $\odot_k^+$, one more difference between equation (4.7) for SBPSO and equation (1.4) is the absence of an inertia term. This can be explained by first looking at the position update equation for SBPSO:

$$X_i(t+1) = X_i(t) \boxplus V_i(t+1) \tag{4.8}$$

The velocity $V_i(t+1)$ is a set of operation pairs $\{(\pm, e_1), \ldots, (\pm, e_m)\}$ that is fully applied to the position $X_i(t)$, where each operation pair is an addition or a deletion. Once an element e has been added to the position $X_i(t)$, adding the element again has no impact as a set can only contain a single instance of each element. Therefore, once the velocity has been applied to $X_i(t)$, each operation pair in $V_i(t+1)$ will have no impact if applied to $X_i(t+1)$. Hence, there is no need to include part of $V_i(t)$ in $V_i(t+1)$, which is what the inertia term would do. The SBPSO algorithm combining the update equations (4.7) and (4.8) is given in algorithm 7.

Note that the order in which the operation pairs from $V_i(t+1)$ are applied to $X_i(t)$ is not relevant, because the individual additions and deletions $v_{i,j}$ in $V_i(t+1)$ from equation (4.7) can overlap, but can *not* cancel each other out. In other words, there can not be a $j_1 \neq j_2$ such that $v_{i,j_1} = (+, e)$ and $v_{i,j_2} = (-, e)$ are two operation pairs in $V_i(t+1)$ for the same element e. To illustrate, assume that $V_i(t+1)$ contains both $(+, e)$ and $(-, e)$ for some element e:

- Since attraction towards $Y_i(t)$ or $\widehat{Y}_i(t)$ can only create deletions for elements in $X_i(t) \backslash \big(Y_i(t) \cup \widehat{Y}_i(t)\big)$, while the $\odot^-$ operation can only create deletions for elements in $S_i(t)$, the presence of deletion $(-, e)$ in $V_i(t+1)$ implies that $e \in X_i(t)$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 7: Set-Based PSO algorithm (SBPSO) for maximization problems

```
Set $N$ equal to the number of particles in the swarm;
for $i = 1, \dots, N$ do
    Initialize $X_i$ as random subset of $U$;
    Initialize $V_i = \emptyset$;
    Calculate $f(X_i)$;
    Initialize $f(Y_i) = -\infty$;
    Initialize $f(\widehat{Y}_i) = -\infty$;
end
while stopping condition is false do
    for $i = 1, \dots, N$ do
        // set the personal best position;
        if $f(X_i) > f(Y_i)$ then
            $Y_i = X_i$;
        end
        // set the neighborhood best position;
        for all neighbors $l$ of particle $i$ do
            if $(f(Y_i) > f(\widehat{Y}_l)$ then
                $\widehat{Y}_l = Y_i$;
            end
        end
    end
    for $i = 1, \dots, N$ do
        Update $V_i$ according to equation (4.7);
        Update $X_i$ according to equation (4.8);
        Calculate $f(X_i)$;
    end
end
```

- Since attraction towards $Y_i(t)$ or $\widehat{Y}_i(t)$ can only create additions for elements in $X_i(t) \setminus \big(Y_i(t) \cup \widehat{Y}_i(t)\big)$, while the $\odot_k^+$ operation can only create additions for elements in $A_i(t)$, the presence of addition $(+,e)$ in $V_i(t+1)$ implies that $e \notin X_i(t)$ or $e \in \big(U \setminus X_i(t)\big)$.
- For e it must then hold that $e \in X_i(t) \cap \big(U \setminus X_i(t)\big) = \emptyset$. Therefore, such an e can not exist in $V_i(t+1)$.

4.5 Conclusions

This chapter formally introduced SBPSO as an algorithm to solve discrete optimization problems that allow for a set-based representation. By doing so, this chapter has addressed three of this thesis' sub-objectives identified in the preface, namely:

- the objective to formally define the SBPSO algorithm by formulating the PSO update equations in terms of set-theory;
- the objective to identify which additional components, on top of the basic features that define a

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

PSO algorithm, are required to make the SBPSO algorithm a functioning algorithm; and

- the objective to ensure that the algorithm is formulated generically and does not include any problem domain specific features in the algorithm itself - the only link to the problem domain should lie in the objective function.

The main concepts that form a PSO algorithm were identified in chapter 1 as a swarm of particles which each have a position and a velocity, whereby a velocity update equation forms the elegant but effective mechanism that provides PSO's effectiveness. Before being able to define a position or velocity update equation for SBPSO, first the position and velocity were defined as mathematical sets in section 4.2. The definition of position used as the universe of discourse U a (finite) set of elements determined by the DOP to be solved. A particle's position is a subset of elements from U, without any further structure or order, other than that elements can only be included at most once as is prescribed for true mathematical sets. A particle's velocity was defined as a mathematical set containing so-called operation-pairs. Each operation pair defines an operation of adding a single element to, or removing a single element from the particle's position.

Using these definitions, the goal was to define SBPSO's update equations in line with the basic components identified for the PSO in chapter 1. These core components of PSO are (i) the position of a particle is updated in consecutive iterations by adding the particle's velocity to its current position and (ii) the velocity itself evolves iteratively according to the velocity update equation which contains three components:

1. a cognitive component that describes the attraction of the particle to the best position in the search space found by that particle previously,
2. a social component that describes the attraction of the particle to the best position in the search space found by any particle in its neighborhood, and
3. an inertia component that causes the velocity to retain part of the direction it currently has.

Four operators were defined that form the set-based equivalent of basic multiplication, addition and subtraction used in the update equations for continuous PSO. Using the above definitions of position and velocity and these four operators, position and velocity update equations could be defined corresponding to that for continuous PSO.

Two findings became apparent when trying to translate the three components of the continuous PSO algorithm into their set-based equivalents for SBPSO: firstly, the concept of an inertia component was inapplicable to the SBPSO as the velocity consists of discrete additions and deletions. Once these additions and deletions are applied to the particle's position, there is no longer a "direction" in which to continue the "movement" implied by the velocity: an element that has been removed from the position can not be removed again. Hence the inertia component of the PSO paradigm was not incorporated into SBPSO.

Secondly, depending on the positions of all particles in the swarm at any point during the search, certain parts of the search space could become inaccessible. Denote by X the position of a particle, Y the personal best position of a particle, and $\widehat{Y}$ the neighborhood best position for a particle. If for *all* the

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

particles in the swarm the cross section $X \cap Y \cap \widehat{Y}$ contained the same element e, this element could not be removed from any particle's position and such points in the search space can no longer be reached. A similar problem arises for elements that lie outside the union $X \cup Y \cup \widehat{Y}$ for *all* particles in the swarm: such elements can not be added to the position of any particle in the swarm. In order to solve these problems, two further operators were introduced that for each particle in isolation can (i) remove elements from the cross section $X \cap Y \cap \widehat{Y}$, or (ii) add elements from outside the union $X \cup Y \cup \widehat{Y}$. In this way the SBPSO algorithm is guaranteed to be able to access the whole search space. The two mechanisms were defined generally, but for both a specific implementation was chosen for use in the remainder of this thesis.

It was shown that the manner in which the velocity update equation is constructed explicitly prevents offsetting operation pairs to be present in the velocity. Hence, the order in which the additions and deletion defined by the velocity's operation pairs are applied to the position is irrelevant. It may therefore be concluded that SBPSO is a well-defined set-based algorithm to solve DOPs which follows the PSO paradigm as closely as possible.

The way the SBPSO algorithm was set up also ensures that it is indeed a generic algorithm and no heuristics or components specific to the problem domain are used other than the objective function. SBPSO can thus be applied to any DOP as long as the DOP has recognizable elements that can be used to form the universe of discourse $U$and an objective function f can be constructed which needs to be minimized or maximized.

This chapter in itself completes part II of this thesis, in which the SBPSO algorithm was described. The SBPSO algorithm is tested and compared to alternative discrete PSO algorithms on two different DOPs in part III: the MKP in chapter 5 and the FSP in chapter 6. The next part therefore forms the test of whether the SBPSO algorithm is indeed a fully functioning PSO algorithm for use on discrete optimization problems that allow for a set-based representation and of how well its performance compares to that of alternative discrete PSO algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Part III

Empirical analysis

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 5

Experiments on the multidimensional knapsack problem

The previous chapter defined and described the SBPSO algorithm. Before that, in part I the MKP was introduced in chapter 2 as a discrete optimization problem that can be used as a test-bed for the new algorithm. This chapter applies SBPSO to the MKP and will compare its performance to that of three other PSO algorithms from literature. The objectives of this chapter are three-fold, where the first objective is to determine if it is possible to successfully apply the SBPSO algorithm to the MKP, which in this case means yielding quality solutions. A second objective is to compare the performance of the SBPSO algorithm on the MKP to three other PSO algorithms known from literature and the third objective is to investigate what parameter values work well for the SBPSO. This chapter is organized into four sections: experimental setup, parameter tuning, sensitivity analysis for SBPSO, and the results of the experiments on the four tuned PSO algorithms.

5.1 Introduction

The current chapter can be seen as the culmination of the path outlined in the previous chapters. Chapter 1 provided the background to show that a generic, functioning, set-based PSO algorithm did not yet exist and what components such an algorithm should contain. Chapter 4 introduced the SBPSO algorithm with the claim that it is a generic, functioning, set-based PSO algorithm. In order to validate this claim, the SBPSO algorithm needs to be tested on discrete optimization problems (DOP). The MKP was introduced in chapter 2 and it was argued that the MKP is a non-trivial DOP that forms a valid test-bed for the SBPSO and other discrete PSO algorithms. This chapter brings together the two parts and applies SBPSO to the MKP. The review of the literature in chapter 2 also suggested a number of other discrete PSO algorithms the SBPSO can be compared to.

As mentioned above, the following objectives from the preface are addressed in this chapter:

- To test the new algorithm on DOPs;
- To compare the performance (in terms of quality of the solution found) of the new algorithm against known discrete PSO algorithms from literature which have been applied on the MKP;

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- To investigate the new algorithm's control parameter values that yield good results.

The first two of these objectives are self-explanatory: any new algorithm that is introduced needs to go through vigorous tests and comparisons in order to assess what its value is and in what situations it can be a good choice to use in practice. Such tests can also help identify possible limitations and areas of improvement.

The third objective is linked to the fact that SBPSO is a new algorithm and little is yet known about which choices of parameters yield adequate or good results. The variant of SBPSO outlined in chapter 4 contains five control parameters to be tuned: c_1, c_2, c_3, c_4 and k, the size of the tournament used to select elements to add to particle positions. An extensive method of tuning is used to find SBPSO's best parameter combination for use on the small and large MKP separately. Besides finding the best parameter combination to use in the experiments, a sensitivity analysis is conducted on the tuning results to better understand what areas of the parameter space yield the best results. Also, the impact of SBPSO's five parameters is compared to see which parameters have the most influence on the quality of the solutions and which parameters are thus most important to be tuned well.

At this point it is also important to repeat the objectives from the preface involving the MKP that are defined to be *outside* the scope of this thesis, and by extension, outside the scope of this chapter:

- find an algorithm that is better at solving the MKP than known state-of-the-art algorithms;
- find the most efficient algorithm in terms of number of iterations or fitness function evaluations, total number of computations (flops) or total time needed to complete;
- to compare the performance of the new algorithm against non-PSO methods used to solve DOPs.

The main goal of this research is to define the SBPSO as a generic, set-based PSO algorithm and show that it is able to find good solutions to different DOPs with a performance that is on par or even better than other generic discrete PSO algorithms. The review of previous work on the MKP in chapter 2 showed that the state-of-the-art algorithms rely heavily on heuristics like repair operators to find the best solutions. Because the goal is to define SBPSO without problem domain specific features, it is expected that SBPSO will not be able to compete with more specialized algorithms.

Although performance can be defined in many different ways, this first detailed investigation of SBPSO focused purely on the quality of the solutions found. Any comparison based on the efficiency of the algorithm would require at the minimum a rebuild of the code used to perform all experiments to ensure an optimal implementation of the algorithm. The question of efficiency and (relative) speed is left for later investigation, as such work is only of interest if the algorithm can first be shown to yield sufficiently good solutions.

Lastly, this thesis compares SBPSO only to other PSO algorithms. This is done because the main novelty of SBPSO lies in the way it uses a set-based approach to PSO; thus it is interesting to see how that approach compares to other existing PSO algorithms. In that way the contribution of the set-based aspect in SBPSO can be determined more clearly than if SBPSO was compared to, say, ACOs or GAs.

The structure of the chapter is as follows: first the experimental setup is described, including parameter tuning of the PSO algorithms. The tuning results for the SBPSO algorithm are then used to

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

perform a sensitivity analysis on SBPSO's parameters to find good values for each and which parameters are most influential on SBPSO's performance. This is then followed by a summary of the results on the experiments performed on the PSO algorithms, split into results for the small MKP and large MKP. The chapter ends with conclusions drawn from the tuning and testing.

The experimental setup used to produce the results in this chapter is described in detail in section 5.2. This setup begins with the selection of the MKP benchmark problems to be used in the experiments. This is followed by listing the three PSO algorithms which are chosen to be compared with the SBPSO. In order to allow for full replication of the results in this thesis, all aspects of the configuration of the four PSO algorithms are given next and explained in a sequence of subsections dealing with swarm size, swarm topologies, objective function, initialization mechanisms, stopping conditions, and number of independent runs performed.

Section 5.3 then details how the parameters of each of the four PSO algorithms were tuned: parameter combinations are generated that span the parameter space of the PSO algorithms and performance on the tuning problems is weighed to find the best parameter combinations. Note that all four PSO algorithms are tuned separately on the small and the large MKPs. This is done to gain insight in the tuning procedure: since the large MKPs form a well-described and homogeneous set for which a perfect tuning set can be constructed, the tuning is expected to work quite well. The small MKP set is more heterogeneous, so the tuning might work less well or provide quite different results. Section 5.4 uses the results of this parameter tuning to analyze the sensitivity of SBPSO to different values of its control parameters. Each of SBPSO's five parameters is investigated separately and the order of their importance to the quality of the tuned solution is determined.

The final results obtained for the four PSO algorithms are then given in section 5.5, split into those for the small MKP and large MKP separately. Since experiments are performed on four PSO algorithms, each combined with three different swarm topologies, results are first compared per topology (all four algorithms use the same topology) and then per algorithm (a single PSO algorithm with the three different swarm topologies). All results are subjected to statical analysis in order to determine if any algorithm or topology has performed better than the competing ones, and if any pattern can be detected in the performance. Although a number of tables are included in this results section that summarize the outcomes of the experiments performed, the volume of such results does not allow all details to be included in this chapter. Therefore, two appendices are included at the end of this thesis with more detailed results: appendix A for those on the small MKPs and appendix B for those on the large MKPs.

5.2 Experimental procedure

This section describes the procedure that is used in the experiments for tuning the PSO algorithms and evaluating their performance. The following design choices are touched upon in separate sub-sections: the selection of benchmark problems, the selection of PSO algorithms to compare with the SBPSO, the swarm size, the swarm topologies, the objective function used in the optimization process, the initialization process for each PSO algorithm, stopping conditions, and the number of independent runs performed in each experiment.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.2.1 Benchmark problems used

The MKPs used in the experiments consist of both problem sets used by Chu and Beasley [18]: 55 small MKPs and 270 large MKPs as described in section 2.3. The two sets of problems were each further split into a tuning set used to find the best parameters for the algorithms, and a test set that is used to compare the performance of the tuned algorithms. This section describes how these splits were made and the exact resulting tuning and test sets.

5.2.1.1 Small MKP

For the small MKPs, a tuning set of 15 problems was manually chosen. The remaining 40 problems formed the test set. Which small MKPs were selected for the tuning set and which for the test set is summarized in table 5.1. The tuning set was chosen to reflect the range of problem sizes in the entire set of 55 problems, with the number of variables n ranging from 20 to 90, and the number of constraints m ranging from 2 to 30.

Table 5.1: Split of small MKPs into tuning and test problems

Tuning Set			**Test Set**								
problem	n	m	problem	n	m	problem	n	m	problem	n	m
mknap1-4	20	10	mknap1-1	6	10	mknap2-23	50	5	mknap2-42	34	4
mknap1-5	28	10	mknap1-2	10	10	mknap2-24	60	5	mknap2-43	18	2
mknap2-10	71	2	mknap1-3	15	10	mknap2-25	60	5	mknap2-44	20	10
mknap2-15	30	5	mknap1-6	39	5	mknap2-27	60	5	mknap2-46	37	30
mknap2-17	40	5	mknap1-7	50	5	mknap2-29	70	5	mknap2-47	28	4
mknap2-2	60	30	mknap2-1	60	30	mknap2-3	24	2	mknap2-5	24	2
mknap2-20	50	5	mknap2-11	30	5	mknap2-30	70	5	mknap2-6	24	2
mknap2-26	60	5	mknap2-12	30	5	mknap2-31	70	5	mknap2-7	24	2
mknap2-28	70	5	mknap2-13	30	5	mknap2-32	80	5	mknap2-8	24	2
mknap2-33	80	5	mknap2-14	30	5	mknap2-34	80	5	mknap2-9	71	2
mknap2-39	90	5	mknap2-16	40	5	mknap2-35	80	5			
mknap2-4	24	2	mknap2-18	40	5	mknap2-36	90	5			
mknap2-41	27	4	mknap2-19	40	5	mknap2-37	90	5			
mknap2-45	40	30	mknap2-21	50	5	mknap2-38	90	5			
mknap2-48	35	4	mknap2-22	50	5	mknap2-40	90	5			

The three smallest problems (mknap1-1, mknap1-2, mknap1-3) were left out of the tuning set on purpose, as the search spaces for these problems are small. Since each item in the MKP is either in or out of the knapsack (two options), the number of points in the search space is equal to 2 to the power of the number of items in the problem. For the three smallest problems this amounts to $2^6 = 64$, $2^{10} = 1024$, and $2^{15} = 32768$ points in the search space respectively. For such simple problems, little difference is to be expected in the performance of the algorithm using different control parameters, so the problems yield little information on which parameters are best.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.2.1.2 Large MKP

For the large MKP, the total set of 270 problems consists of 27 subsets of problems, each of which contains 10 random instances for a given combination of problem parameters n, m, and tightness ratio r as described in section 2.3. For the tuning set, one problem was selected at random from each of the 27 subsets, and the remaining 243 problems formed the test set. The 27 tuning problems, each with the number of variables, the number of constraints, and tightness ratios are summarized in table 5.2.

Table 5.2: Split of large MKPs into tuning and test problems. Only the 27 tuning problems are shown, the remaining 243 problems form the test set.

problem	n	m	r	**problem**	n	m	r	**problem**	n	m	r
mknapcb1-6	100	5	0.25	mknapcb4-3	250	5	0.25	mknapcb7-1	500	5	0.25
mknapcb1-17	100	5	0.50	mknapcb4-12	250	5	0.50	mknapcb7-19	500	5	0.50
mknapcb1-27	100	5	0.75	mknapcb4-27	250	5	0.75	mknapcb7-30	500	5	0.75
mknapcb2-7	100	10	0.25	mknapcb5-7	250	10	0.25	mknapcb8-10	500	10	0.25
mknapcb2-11	100	10	0.50	mknapcb5-20	250	10	0.50	mknapcb8-16	500	10	0.50
mknapcb2-22	100	10	0.75	mknapcb5-21	250	10	0.75	mknapcb8-26	500	10	0.75
mknapcb3-3	100	30	0.25	mknapcb6-7	250	30	0.25	mknapcb9-8	500	30	0.25
mknapcb3-20	100	30	0.50	mknapcb6-16	250	30	0.50	mknapcb9-18	500	30	0.50
mknapcb3-24	100	30	0.75	mknapcb6-23	250	30	0.75	mknapcb9-26	500	30	0.75

5.2.2 PSO algorithms

The SBPSO algorithm is compared to three other PSO algorithms: BPSO by Kennedy and Eberhart [62], MBPSO by Shen *et al.* [125], and PBPSO by Zhen *et al.* [157]. Refer to sections 1.3.1.1, 1.3.1.2, and 1.3.1.3 for detailed descriptions of these algorithms. The reason these three algorithms were chosen for comparison was that they do no incorporate any domain specific methods such as a repair operator. Also, these algorithms have been used on the MKP before and this resulted in reasonable results, see for example Wang *et al.* [146].

BPSO, MBPSO, and PBPSO are all so-called binary PSO algorithms: the candidate solution in the algorithms is represented by binary-valued particle positions. In terms of the MKP this means that the bit values in the position are directly interpreted as the x_i values in equation (2.1). That is, a particle indicates the assignment of items to the knapsack. For SBPSO, in order to evaluate a solution, the x_i from equation (2.1) are set to 1 for all items that are included in the particle position set, and set to 0 for all items that are not.

5.2.3 Swarm size

An important parameter in PSO algorithms is the number of particles in the swarm. While the optimal number of particles for a specific algorithm-problem pair can be problem dependent, this study used the same number of particles for all algorithms and for all problems in each problem set: for small MKPs the number of particles was set to 25, while for large MKPs the number of particles was set to 50. Using

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the optimum number of swarm particles is not too important because the objective is not find the best possible solution to the MKP, but to compare the relative performance of the different algorithms and swarm topologies. Hence using the same number of particles in each case that is comparable is important.

5.2.4 Swarm topologies

Each of the four PSO algorithms is used with each of the following three topologies: star, ring, and Von Neumann. This results in 12 algorithm-topology pairs. The pairs with a star topology are referred to as global best PSO shortened to GB in the tables in the remainder of this document. Similarly, the pairs with a ring topology are referred to as local best PSO shortened to LB, and the pairs with a Von Neumann topology are referred to as VN in the tables.

Particles organized in a swarm topology are considered connected if they are in each other's neighborhood. Particles that are not in each other's neighborhood are connected indirectly due to overlap between neighborhoods. If, for example, particle i is not connected to particle j, but the two particles share a common neighbor k, then the path $i-k-j$ connects particles i and j in the topology. The distance between two particles in a topology is determined by the *shortest* path that connects the two particles. For particles i and j from the example, the $i-k-j$ path is the shortest path, and the distance between i and j thus is 2. The average distance across all possible pairs in a swarm, called the *average shortest path length*, is a measure of how connected the swarm is.

A swarm with the star topology always has an average shortest path length of 1, as each particle is in each other particle's neighborhood. For the Von Neumann topology, the average shortest path length depends on the number of particles in the swarm. For swarms of 25 and 50 particles, the Von Neumann topology leads to average shortest path lengths of 2.5 and 3.5 respectively. For the ring topology, the average shortest path length depends not only on the swarm size, but also on the neighborhood size. A neighborhood size of 4 was chosen for the experiments of this study, such that the swarms with a ring topology are less connected than those using either of the other two topologies. This resulted in average shortest path lengths for swarms with the ring topology of 3.5 for a swarm of 25 particles, and 6.6 for a swarm of 50 particles.

Therefore, in the experiments conducted, swarms with the star topology were the most connected, swarms with the ring topology were the least connected, and swarms with the Von Neumann topology had an intermediate level of connectedness.

5.2.5 Objective function

The MKP is defined as a maximization problem. The objective function used is the same for all the PSO algorithms. For particles that represent a feasible solution to the MKP, that is, which satisfy all m constraints in equation (2.3), the objective function value was set equal to the sum of the values of the items in the particle. Particles that do not represent a feasible solution because they violate at least one of the constraints in equation (2.3), were assigned an objective function value of minus infinity. Since a particle uses its position to represent a solution, the objective function value of a particle is computed as

© University of Pretoria

$f(X(t))$, defined as

$$f(X(t)) = \begin{cases} \sum_{i=1}^{n} v_i x_i & \text{if } \forall j \in \{1,\ldots,m\} : \sum_{i=1}^{n} w_{i,j} x_i \leq C_j \\ -\infty & \text{if } \exists j \in \{1,\ldots,m\} : \sum_{i=1}^{n} w_{i,j} x_i > C_j \end{cases} \tag{5.1}$$

Using the terminology from chapter 2, this objective function means that a penalty function approach is used. By setting the value of the objective function to minus infinity for all infeasible positions, no detailed domain specific information is used. In other words, there is no difference in the objective value of a position that is only just breaching one of the MKP's constraints and of a position that is breaching many constraints and therefore also no gradient in the infeasible part of the solution space that can be exploited by the PSO algorithms. Although an approach that uses a penalty proportional to the breach of the constraints would probably work better, such an approach also makes the algorithms less generic and generality of the algorithms plays a large role in the objectives in this thesis. By making the problem harder, any differences in performance between the four algorithms should also stand out more.

In order to facilitate a comparison of results across different problems, the results in this thesis for experiments on the MKP do not show the raw objective function values. For the small MKPs, the error between the best objective function value found and the known optimum is shown. Since the optimal solutions are not known for all the large MKPs, for these problems the error between the best found objective function value and the LP relaxation bound is shown instead. The LP relaxation bounds were obtained using the Java wrapper of *lp_solve 5.5*, which is based on the revised simplex method [8]. The bounds found corresponded perfectly with those made available by Chu and Beasley [18].

5.2.6 Initialization procedure

Particles were initialized randomly for each algorithm-topology pair. For the BPSO, MBPSO, and PBPSO algorithms, the positions were initialized randomly in $\{0,1\}^n$, while the velocities for BPSO and PBPSO were initialized randomly in $[-1,1]^n$, following [31]. For PBPSO the continuous-valued positions, $\vec{x}'_i(0)$, were initialized as $\vec{0}$, to ensure that no initial bias was included in the discrete-valued positions, $\vec{x}_i(0)$. For the SBPSO algorithm, the positions were randomly initialized, such that each element had a 0.5 chance of being included, and all velocities were initialized as an empty set.

5.2.7 Stopping conditions

For each independent run of an algorithm, the same three stopping conditions were applied. These stopping conditions were:

1. the best objective function value in the swarm equaled the known optimum (in case of small MKPs) or equaled the LP relaxed bound (in case of large MKPs),
2. the best objective function value in the swarm had not improved for 2500 iterations, or
3. more than 5000 iterations had passed.

© University of Pretoria

5.2.8 Number of independent runs

PSO is a stochastic optimization algorithm, and individual runs of the algorithm can have different results even if all circumstances other than the random number generator are kept the same. Hence, multiple independent runs of the algorithms have to be executed and the average performance reported. For the small MKPs, 30 independent runs were used for tuning the algorithms and 100 independent runs were used to ascertain the average performance on the test problems. For the large MKPs, 30 independent runs were used both for tuning the algorithms and to determine the performance on the test problems.

The difference in the number of independent runs used for determining the average performance on the test set for the small MKPs and the large MKPs is solely due to the computational load: for the large MKPs the test set contains more problems (243 versus 40 for the small MKPs) and the number of variables per problem range from 100 to 500 for the large MKPs while only from 6 to 90 for the small MKPs.

5.3 PSO parameter tuning

This section describes how the twelve PSO algorithm-topology pairs were tuned on the small MKP and the large MKP problem sets separately. Section 5.3.1 describes how the parameter tuning was performed. Sections 5.3.2 and 5.3.3 list and discuss the resulting best control parameter values per algorithm-topology pair for the small MKP and the large MKP respectively.

5.3.1 Tuning process

While efficient parameter tuning approaches like F-Race [9] exist, a different process was chosen to tune the parameters of the four PSO algorithms in these experiments. This was done because the chosen process was deemed more appropriate for the sensitivity analysis conducted in Section 5.4, while speed of the tuning process was not a priority in the experiments. All four algorithms were tuned in the same manner to preserve *ceteris paribus* in the final comparison of test results. Besides the main tuning process, a sensitivity analysis was conducted for the SBPSO algorithm only.

For each of the 12 algorithm-topology pairs, the same process was used to tune the algorithm's parameters, although the number of control parameters differed: MBPSO has only two parameters, while BPSO has four, PBPSO has six, and SBPSO has five parameters. Each algorithm-topology was tuned twice: once on the tuning set of small MKPs and once on the large MKPs. The end result of the parameter tuning was a total of 24 tuned parameter combinations.

Table 5.3 lists the ranges of possible parameter values used in the tuning process. The Cartesian product of the parameter value ranges for one algorithm forms the parameter space for that algorithm. For each of the four PSO algorithms, 128 parameter combinations were generated that span each algorithm's parameter space. Only static control parameters were considered. In order to generate the parameter combinations in a manner that ensures that the parameter space was covered well, sequences of Sobol pseudo-random numbers were used according to the method proposed by Franken [36].

Even though the number of parameters differs across the algorithms, the same number (128) of

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 5.3: Parameter ranges used in tuning the four PSO algorithms on the MKP

algorithm	BPSO	PBPSO	algorithm	MBPSO	SBPSO
ω	[0.50, 0.99]	[0.50, 0.99]	p_{stat}	[0.00, 1.00]	
c_1	[0.00, 5.00]	[0.00, 5.00]	p_{reset}	[0.00, 1.00]	
c_2	[0.00, 5.00]	[0.00, 5.00]	c_1		[0.00, 1.00]
V_{max}	[1.00, 10.00]	[1.00, 10.00]	c_2		[0.00, 1.00]
R		[1.00, 100.00]	c_3		[0.50, 5.00]
p_{mut}		[0.00, 0.50]	c_4		[0.50, 5.00]
			k		$\{1,\ldots,9\}$

parameter combinations were used in tuning each of the algorithm-topology pairs on the two problem sets. Hence, for the MBPSO algorithm, which has only two parameters, the parameter combinations provided a denser covering of the two-dimensional parameter space than for the other PSO algorithms which each have at least four parameters. Thus the amount of effort expended in tuning was the same, but the MBPSO was rewarded for having fewer parameters by having those parameters tuned in more detail.

Note that the tuning process for each algorithm used the same 128 parameter combinations for each of the three topologies, and on both problem sets. Thus, for example, in tuning BPSO using a star topology on the small MKPs, the same 128 parameter combinations were considered as in tuning BPSO using a Von Neumann topology on the large MKPs.

The next step in the tuning process was to determine the best parameter combination for each of the algorithm-topology pairs on each of the problem sets. To do this, 30 independent runs were conducted for each of the parameter combinations, on all the tuning problems in the problem set. For each problem, the average of the best objective function value achieved by each of the 128 parameter combinations over the 30 runs was determined. The parameter combinations were *ranked* in order of the average objective function value for each problem separately: the parameter combination with the highest objective function value (or equivalently the lowest error) thus was assigned rank 1, while the worst combination was assigned rank 128. In case two or more combinations achieved the exact same average objective function value, the ranks for each were combined and averaged: three combinations ranked 36th, 37th and 38th, for example, would all have been assigned the average rank of 37.

Finally, the *average rank* was determined for each parameter combination by averaging over all the problems in the set. The parameter combination with the lowest average rank was deemed best and chosen as the tuning result. This method weighed the contribution of each tuning problem equally, and by using the rank of the objective function value instead of the objective function itself, a fair comparison was made using problems that have different optima and different search landscapes.

An overview of the results of tuning the 12 algorithm-topology pairs on the small MKPs and the large MKPs, in the form of the best parameter combination found for each topology-algorithm pair, are shown and discussed in sections 5.3.2 and 5.3.3 respectively. Detailed results for the small MKPs and the large MKPs can be found in appendices A and B respectively.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.3.2 Tuning results for small MKPs

This section describes and discusses the results for the small MKPs of the tuning procedure for the PSO algorithms described in the previous section. Table 5.4 summarizes the best parameters found, for each of the algorithm-topology pairs on the small MKPs.

Table 5.4: Tuned parameters for small MKPs

algorithm	BPSO			PBPSO		
topology	GB	LB	VN	GB	LB	VN
ω	0.9211	0.9594	0.9709	0.6876	0.6455	0.6455
c_1	4.6094	2.8125	4.4141	0.7422	4.2969	4.2969
c_2	1.3281	1.5625	1.9922	0.5078	4.7656	4.7656
V_{max}	5.9219	7.1875	5.1484	3.3203	4.2344	4.2344
R				37.352	64.422	64.422
p_{mut}				0.0742	0.0391	0.0391

algorithm	MBPSO			SBPSO		
topology	GB	LB	VN	GB	LB	VN
p_{stat}	0.4844	0.4766	0.4766			
p_{reset}	0.3906	0.3203	0.3203			
c_1				0.9297	0.5156	0.5156
c_2				0.2266	0.4531	0.4531
c_3				1.3086	1.8359	1.8359
c_4				2.1523	2.2578	2.2578
k				7	7	7

For BPSO, the attraction to the neighborhood best particle, c_1, increased as the swarm topology was less connected: highest for gbest BPSO, lowest for lbest BPSO. The attraction to the personal best, c_2, ranged from 1.3 for the star topology to 2.0 for the Von Neumann topology, and was clearly smaller than the values for c_1. The inertia weight ω was high for each of the three topologies, as was the V_{max}, which was above 5 in all cases.

For PBPSO, the best parameter value combinations for lbest PBPSO and the Von Neumann topology were the same, but the best parameter values found for gbest PBPSO were quite different, mainly with much lower c_1 and c_2 values. Note that, compared to BPSO, the inertia weight for the best parameter value combinations for PBPSO was much smaller.

For MBPSO, the three values found for the static probability, p_{stat}, were similar and comparable to the value of 0.5 used by the original authors, Shen *et al.* [125]. The value of p_{reset} of 32% to 39% was, however, more than triple the 10% used by Shen *et al.* [125], indicating that a high proportion of random resets was beneficial.

For SBPSO, the parameter value combinations for the ring and Von Neumann topologies were the same, while for the star topology a different parameter value combination was optimal with a much higher c_1 and lower c_2. Section 5.4 gives a detailed analysis of the sensitivity of SBPSO's parameters using the tuning results.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Section A.1 in appendix A contains the detailed results for the chosen best parameter combination for each of the 12 algorithm-topology pairs on the small MKPs. For these 15 tuning problems, the best parameter combination for each algorithm-topology pair could often find the known optimum in all of the 30 independent runs of the algorithm. The average over all 12 pairs is 5.2 problems solved perfectly out of 15 tuning problems each. The problem with solving the problems perfectly during tuning is that the discriminatory power of the tuning problems diminishes due to a ceiling effect. A quick example using the GB BPSO algorithm-topology pair is indicative: for this pair the chosen parameter combination was able to perfectly solve three tuning problems: mknap1-4, mknap2-15, and mknap2-4. For these three problems respectively, 118, 71 and 96 parameter combinations were able to perfectly solve the MKP. On these three problems the tuning process was not able to distinguish the quality of more than half of the 128 parameter combinations, making the effective tuning set closer to 12 problems than 15.

Another result that can be derived from the detailed tuning results for small MKPs on BPSO, MBPSO and PBPSO is that the best parameter combination found by the tuning process is one that is the best compromise to work effectively on the whole set of problems, and not a single best parameter combination that works best on all problems. The SBPSO algorithm instead does not seem so affected on the set of 15 tuning problems. This can be seen from the average rank of the chosen parameter combination out of 128 combinations. A single combination that works better than all other combinations on all 15 tuning problems would score an average of 1. Table 5.5 lists the average ranks for the 12 algorithm-topology pairs tuned on the 15 small MKP tuning problems. The left-hand side of the table shows the average rank across all 15 tuning problems, while the right-hand side shows the average rank on all problems not solved perfectly. The number of such problems is indicated between brackets.

Table 5.5: Average rank out of 128 of best parameter combination in tuning small MKPs across the 15 tuning problems. The columns labeled “#” indicate the number of problems that were not solved perfectly by the algorithm-topology pair.

	All 15 tuning problems			Not perfectly solved problems					
	GB	LB	VN	GB		LB		VN	
algorithm	avg. rank	avg. rank	avg. rank	avg. rank	(#)	avg. rank	(#)	avg. rank	(#)
BPSO	26.5	15.2	15.8	21.1	(12)	9.7	(8)	10.1	(10)
MBPSO	16.1	8.2	7.2	15.5	(13)	6.8	(12)	5.3	(13)
PBPSO	15.2	12.9	13.3	11.4	(12)	10.1	(9)	10.2	(10)
SBPSO	8.3	11.5	11.2	2.4	(7)	2.2	(6)	1.8	(6)

Using the average rank on the non-perfectly solved problems the BPSO, MBPSO, and PBPSO algorithms show a higher average rank than SBPSO. The average rank for the three non-SBPSO algorithms ranges between 5.3 (for VN MBPSO) to 21.1 (for GB BPSO). This indicates that, for these three algorithms, performance of the best parameter combination is not uniform across the tuning set: other parameter combinations work better on some problems, but worse on others such that the average performance is worse. This is not optimal, as it is preferable that a single combination of parameters works best across a range of problems leading to what could be called a robust tuning result.

For the SBPSO algorithm, the tuning result on the small MKPs was much more robust. Disregarding

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the problems solved perfectly (which discriminate less in quality between parameter combination), the average ranks became 2.4, 2.2, and 1.8 for GB SBPSO, LB SBPSO, and VN SBPSO respectively. In other words, for SBPSO the chosen parameter combination was on average in the top 3 out of 128 possible combinations, and the worst ranks on the non-perfectly solved problems were 8th, 4th and 3rd for the chosen parameter combinations for GB SBPSO, LB SBPSO, and VN SBPSO respectively. Note that the average errors and the average errors on the non-perfectly solved problems in table A.5 were skewed by the fact that the local optimum found for problem “mknap2-10” had an error of more than 5% to the known optimum. Excluding this problem, the average errors over the remaining 14 problems were 0.049%, 0.023%, and 0.024% for GB SBPSO, LB SBPSO and VN SBPSO respectively. The non-perfectly solved tuning problems in table A.5 posed similar problems for the BPSO, MBPSO and PBPSO algorithms.

5.3.3 Tuning results for large MKPs

Table 5.6 summarizes the best parameter values found using the parameter tuning procedure described in section 5.3.1 for each of the algorithm-topology pairs on the large MKPs.

Table 5.6: Tuned parameters for large MKPs

algorithm		BPSO			PBPSO	
topology	GB	LB	VN	GB	LB	VN
ω	0.9785	0.9785	0.9785	0.6263	0.7373	0.7373
c_1	2.4609	2.4609	2.4609	3.8672	2.7344	2.7344
c_2	4.1016	4.1016	4.1016	3.6328	1.9531	1.9531
$V_{\max}$	9.2266	9.2266	9.2266	8.9453	7.0469	7.0469
R				74.477	82.984	82.984
p_{mut}				0.0117	0.0078	0.0078
algorithm		MBPSO			SBPSO	
topology	GB	LB	VN	GB	LB	VN
p_{stat}	0.4531	0.2266	0.3828			
p_{reset}	0.1094	0.0703	0.1016			
c_1				0.9297	0.3672	0.3672
c_2				0.2266	0.9141	0.9141
c_3				1.3086	1.5898	1.5898
c_4				2.1523	1.3086	1.3086
k				7	3	3

For BPSO, the best parameter value combinations found on the large MKPs were exactly the same for each of the three topologies, characterized by a high inertia weight ω, high $V_{\max}$ and $c_2 > c_1$. The latter inequality indicates a stronger attraction to the neighborhood best position than to the personal best position, which is the reverse of the results found for BPSO on the small MKPs, where $c_1 > c_2$.

For PBPSO, the ring and the Von Neumann topologies yielded the same best parameter value combination. For all three topologies, the values found for the inertia weight, ω, were similar. These values

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

are also very similar to the corresponding values found during tuning on the small MKPs: a relative difference of only 10%-14% was seen. For all three topologies, the parameter values found for $V_{\max}$, R, and p_{mut} showed some differences between those for gbest PBPSO and the other two topologies. But these differences are much smaller than the large difference for these parameter values compared to the tuning results on the small MKPs. On the large MKPs, the best values for $V_{\max}$ and R were much higher. Also, the values for p_{mut} were lower, indicating that having many random mutations was less helpful on the large MKPs. For the gbest PBPSO, the best values for c_1 and c_2 resulted in much higher values than those found for the small MKPs, while lbest PBPSO and the Von Neumann topology yielded lower values than on the small MKPs.

For MBPSO there was some variation in the best values of p_{stat} compared to the values found on the small MKPs: a lower value was found on the large MKPs for both the gbest and lbest MBPSO, while for the Von Neumann topology, p_{stat} was higher on the large MKP. For p_{reset}, the best values found were close to the 10% used by Shen *et al.* [125].

For SBPSO, the best parameter values found for gbest SBPSO were exactly the same as those found on the small MKPs. The best parameter values for lbest SBPSO and the Von Neumann topology matched, but were quite different than those found on the small MKP: the attraction to the personal best, c_2, was much higher for the larger MKPs, while the attraction to the neighborhood best, c_1, was lower.

Section B.1 in appendix B contains the detailed results for the chosen best parameter combination for each of the 12 algorithm-topology pairs on the large MKPs. Results for the chosen parameter combinations for each algorithm-topology pair are shown separately for each of the 27 tuning problems. In general the issue of a ceiling effect identified in tuning the small MKPs is not present for the large MKPs: there is no case of the average error leveling of at the same result for multiple parameter combinations because the optimal solution has been found in all 30 independent runs. Instead a different average error is seen for almost all 128 parameter combinations in each of the 12 algorithm-parameter combinations tested. Hence it can be concluded that, across the 27 tuning problems, the various parameter combinations can be discriminated from each other with regards to the quality of results generated.

Also in general a good overall performance is seen of the chosen parameter combination across the set of 27 tuning problems. This can be noted from the average rank of the chosen parameter combination out of the 128 combinations averaged over the tuning set summarized in table 5.7. The “worst” average rank is 6.5 out of 128 for GB MBPSO which is the best performing parameter combination on only 3 out of 27 tuning problems. All pairs involving the BPSO algorithm, LB PBPSO, VN PBPSO and GB SBPSO show the most robust performance of the chosen parameter combination: almost two thirds of the tuning problems are solved best by the chosen parameters. This contrasts with the results on the small MKP tuning problems in table 5.5 and indicates that the tuning set for large MKPs is more homogeneous with regard which parameter combinations perform best.

The detailed tuning results for the BPSO algorithm in table B.2 indicate that the chosen parameter combinations are the best (rank = 1) of the 128 combinations tested for all problems with tightness ratio $\alpha = 0.75$ or 0.50. This is not the case, however, for combinations with $\alpha = 0.25$: here the chosen combination performs well (always in top 10%) but other parameter combinations are better suited to these problems.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 5.7: Average rank out of 128 of best parameter combination in tuning large MKPs across the 27 tuning problems. The columns labeled “# 1’s” indicate the number of problems for which the chosen parameter combination performed best out of 128 combinations.

	GB		LB		VN	
algorithm	avg. rank	(# 1’s)	avg. rank	(# 1’s)	avg. rank	(# 1’s)
BPSO	2.3	(20)	2.5	(19)	2.6	(18)
MBPSO	6.5	(3)	4.1	(5)	3.1	(3)
PBPSO	2.5	(11)	1.4	(17)	1.2	(22)
SBPSO	1.7	(20)	2.8	(5)	2.1	(6)

The tuning results in table B.3 indicate that for MBPSO the chosen parameter combination that performed best in tuning has a much higher average rank than that of the other three algorithms in tables B.2, B.4 and B.5. This indicates that for MBPSO no single one parameter combination was superior to the other 127 combinations used in tuning. For the Von Neumann topology this effect is less pronounced, but even here the chosen parameter combination scored best (rank = 1) on only two out of 27 tuning problems.

Table B.4 lists the detailed tuning results for the PBPSO algorithm. For VN PBPSO the chosen parameter combination scores best for 22 out of 27 problems and scores second best on the other five. VN PBPSO thus has a clear best performing parameter combination in the universe of 128 tuning combinations and it seems the algorithm can be tuned to the whole set of problems. For LB PBPSO a similar, if slightly less pronounced pattern, can be seen. The GB PBPSO performs slightly worse on the problems with a low ($n = 5$) number of constraints: the average rank on the nine problems with $n = 5$ is 4.7 versus 1.4 on the other 18 tuning problems. On problems with fewer constraints other parameter combinations thus seem to offer a better performance, but no clear link with the tightness ratio α is seen.

Table B.5 lists the detailed tuning results for the SBPSO algorithm. For GB SBPSO the best parameter combination stands out more than for LB SBPSO and VN SBPSO with a rank of 1 for 22 out of tuning 27 problems. A slightly worse relative performance is seen for problems with $m = 5$ and $\alpha = 0.25$ in other words problems with few constraints, but where the constraints are themselves very restrictive. Such problems are likely to have a solution space with more widely spaced local optima, which seems to cause the star topology to be more sensitive to its chosen parameters. This is somewhat similar to the behavior seen for PBPSO in table B.4.

5.4 Sensitivity analysis of SBPSO’s parameters

This section analyzes the sensitivity of SBPSO to different values of its control parameters. Such sensitivity analysis is important, as little is yet known about what are good values for its control parameters.

The sensitivity analysis procedure is summarized in section 5.4.1, followed by the results for each of the three topologies: the star topology in section 5.4.3, the ring topology with neighborhood size 4 in section 5.4.4, and the Von Neumann topology in section 5.4.5. A discussion of the relative importance of each of SBPSO’s parameters in respect of the algorithm’s performance is given in section 5.4.6.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.4.1 Procedure

The sensitivity of the performance of SBPSO to each individual control parameter was investigated and visualized using cumulative histograms. For each individual parameter, the horizontal axis of the histogram consists of bins which divide the parameter range into equally sized sub-ranges. The vertical axis displays the number of parameter value combinations that fall in each bin, split into four groups based on the performance of the parameter value combination in the tuning process. If a particular bin for an individual parameter contains a large number of parameter combinations that are considered "good", this implies that the sub-range for the individual parameter associated with the bin is good. This section describes how the histograms were constructed, resulting in a histogram for each of the three SBPSO-topology combinations, for each of the five control parameters. In total, 15 histograms were generated.

Note that a good parameter value combination for SBPSO requires that all five parameters individually have a good or at least reasonable value: if even one parameter has a bad value, the parameter value combination as a whole performs badly. The consequence of this is that, if a specific parameter value combination performs badly, this gives little information on whether the *individual* parameter values in that combination are good or bad: any single individual parameter value could be bad, or all values could be bad. Therefore, it is the parameter value combinations that perform well *as a whole* which contain information on the individual parameters. Hence, the sensitivity analysis focused on the 25% of parameter value combinations that performed best in the tuning process.

The performance of a parameter value combination was set equal to its average rank on the small MKPs and the large MKPs tuning sets combined, with each of the two tuning sets weighed equally. The full procedure to construct the histograms used in the following sections consisted of the following steps:

1. For each parameter value combination, the performance was set equal to 0.5 times the average rank on the small MKPs tuning set plus 0.5 times the average rank on the large MKPs tuning set.
2. The parameter value combinations were then themselves ranked based on the performance calculated in step 1.
3. The ranked parameter value combinations were split into quartiles, labeled A for the best 25%, B and C for the next two quartiles respectively, and D for the worst 25% of parameter value combinations[1].
4. Then, for each individual parameter, the parameter range was split into bins:
   (a) parameter values for c_1 and c_2 took values in the range $[0.0, 1.0]$, with the values grouped into the 10 bins, $[0.0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0]$;
   (b) parameter values for c_3 and c_4 took values in the range $[0.5, 5.0]$, with the values grouped into the nine bins, $[0.5, 1.0), [1.0, 1.5), \ldots, [4.5, 5.0]$; and
   (c) parameter values for k took values in the range $1, \ldots, 9$, with the values grouped into nine bins containing one value each.

[1]The parameter value combinations with label A are considered to be good combinations, those with label B are considered reasonable combinations.

© University of Pretoria

These bins form the horizontal axis of the histogram.

5. For each individual parameter, the parameter value combinations were allocated one by one to a bin, based on the value of the individual parameter in the combination. In each bin, a count was kept of the number of parameter value combinations labeled A, B, C, and D separately. Consider, for example, the parameter value combination labeled A with values $(0.95, 0.52, 2.03, 3.17, 3)$. Allocating this parameter value combination to a bin for the individual parameter c_1 entailed increasing by one the count of label A combinations in the sub-range bin $[0.9, 1.0]$.

6. For each individual parameter, a cumulative histogram was then constructed with, for each bin, the number of label A combinations at the bottom in black, on top of which the number of label B combinations is given in dark gray, and on top of that the number of label C combinations in light gray. The remaining parameter value combinations with label D were stacked at the top and "shown" in white.

7. As a final step, each of the bins was scaled to $[0, 1]$ for ease of comparison, as not all parameter value bins contained the same number of parameter value combinations[2].

Each histogram can be interpreted in the same manner: the black graph at the bottom shows the distribution of good parameter value combinations (labeled A for the best 25% combinations) for the individual parameter across the bins. The dark grey graph stacked on top of the black graph similarly shows the distribution of reasonable-but-not-good parameter value combinations (labeled B). Because the histogram is stacked, the top of the dark grey graph is the sum of the fractions of label A and label B combinations in each bin, indicating the fraction of parameter value combinations that are reasonable or better.

Note that, for the acceleration parameters c_1 to c_4, the bin labels on the horizontal axis of the histograms identify the *lower boundary* of the sub-range linked to that bin. For example, the bin for c_1 labeled 0.3 identifies the sub-range $[0.3, 0.4)$, and the bin for c_3 labeled 1.5 identifies the sub-range $[1.5, 2.0)$.

5.4.2 Results

The results of the sensitivity analysis of SBPSO's parameters based on the tuning results on the small and large MKP are presented in this section. Results are split according to the three SBPSO-topology pairs: gbest SBPSO, lbest SBPSO, and Von Neumann SBPSO.

5.4.3 Global best SBPSO

Figure 5.1 shows the histograms for the parameter sensitivity analysis on the gbest SBPSO resulting from the procedure described in section 5.4.1.

[2]Note that, by construction, the parameter value bins for an individual parameter contain almost the same number of parameter value combinations. For parameters c_1 and c_2, the 128 combinations were divided over 10 equally sized bins, resulting in 12 or 13 combinations in each bin. For parameters c_3, c_4, and k, the 128 combinations were divided over nine equally sized bins, resulting in 14 or 15 combinations in each bin. By dividing the results in each bin by the total number of combinations in the bin, the number of combinations with each label was changed instead into the fraction of all combinations with that label,

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

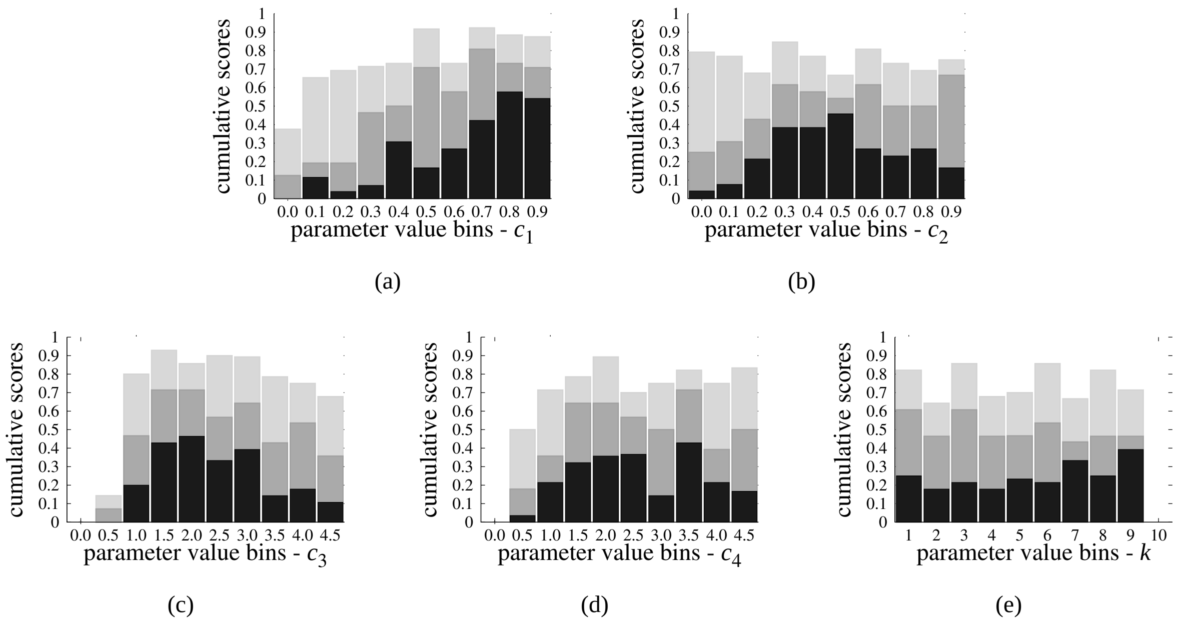


Figure 5.1: Sensitivity analysis of gbest SBPSO parameters: (a) c_1, (b) c_2, (c) c_3, (d) c_4, and (e) k.

For gbest SBPSO, high c_1 values led to better results: parameters in the range $c_1 \geq 0.8$ covered 20% of the parameter space but accounted for more than 45% of label A (the best quantile) parameter combinations. For the c_1 bins with $c_1 < 0.4$, only a few combinations were labeled A. For parameter c_2, the best results were found in the sub-range $[0.3, 0.6)$, which accounted for half the combinations labeled A. Values for c_2 up to 0.3 resulted in bad performance. For c_3 most of the best parameter values were in the range $[1.5, 3.5)$, while the performance of those four bins was approximately the same. For c_4 parameter values between 1.5 and 4.0 scored best, with higher bins performing slightly better, except for the $[3.0, 3.5)$ bin. Larger values of k (indicating that a larger tournament was used to select each element to add based on marginal objective function values) led to better results, but the difference across the bins was quite small.

5.4.4 Local best SBPSO

Figure 5.1 shows the histograms for the parameter sensitivity analysis on the lbest SBPSO with neighborhood size 4 resulting from the procedure described in section 5.4.1.

For lbest SBPSO, high c_1 values led to better performance: parameters in the range $c_1 \geq 0.8$ covered 20% of the parameter space but accounted for more than 47% of label A parameter value combinations. Low c_1 values had few results labeled A, especially those for $c_1 < 0.3$. For parameter c_2, the best values were found in the range $[0.5, 0.6)$, but all bins with $c_2 > 0.4$ scored comparably well, while values $c_2 < 0.4$ clearly performed worse. The best c_3 parameter values were in the range $[1.0, 2.5)$, and performance worsened proportionally for parameter values further away from 2.0. For c_4, the two bins $[2.0, 2.5)$ and $[3.5, 4.0)$ clearly had the most good results, while the parameter values between 2.5 and

so that results are better comparable across bins.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

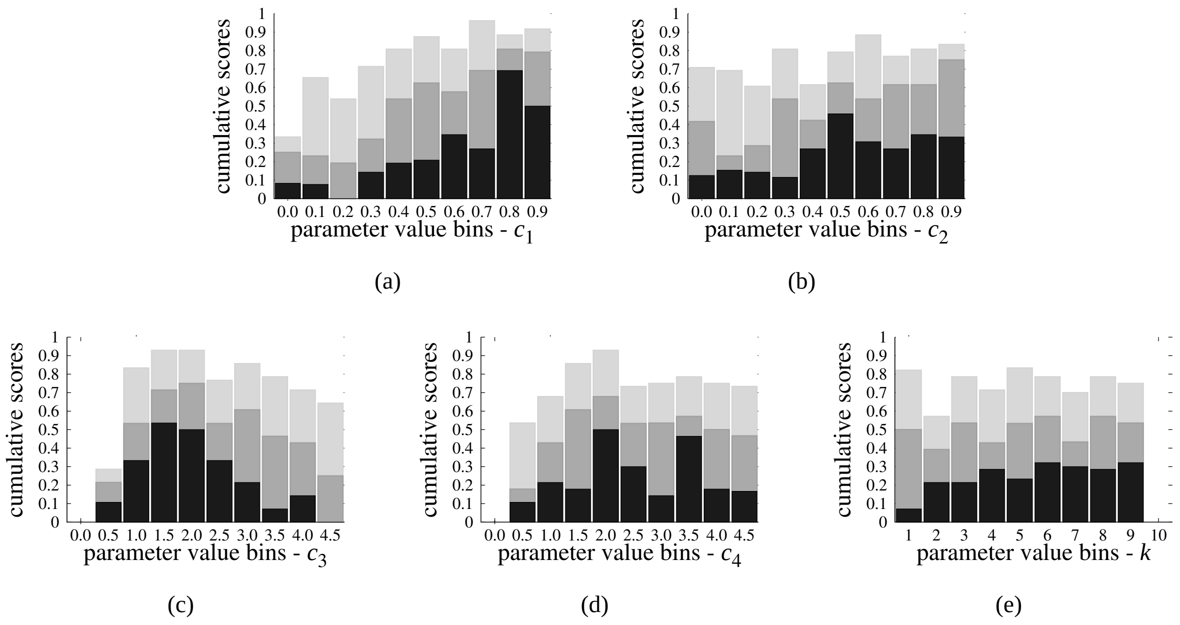


Figure 5.2: Sensitivity analysis of lbest SBPSO parameters: (a) c_1, (b) c_2, (c) c_3, (d) c_4, and (e) k.

3.5 scored worse. Larger values of k led to more good results, but only $k = 1$ clearly performed worse based on the fraction of label A combinations. Combining label A and label B contributions resulted in no significant difference between the performance of each of the nine values of k: any value of k led to the same number of reasonable parameter value combinations.

5.4.5 Von Neumann SBPSO

Figure 5.3 shows the histograms for the parameter sensitivity analysis on the Von Neumann SBPSO resulting from the procedure described in section 5.4.1.

For SBPSO with the Von Neumann topology, high c_1 values led to better performance: parameters in the range $c_1 \geq 0.8$ covered 20% of the parameter space but accounted for more than 46% of good parameter value combinations. Low c_1 values had few good parameter value combinations, especially those for $c_1 < 0.3$. For parameter c_2, the best results were found in the range $[0.5, 0.6)$, but all bins with $c_2 > 0.4$ scored comparably well. For c_3, the best parameter values were in the range $[1.5, 2.5)$, and performance worsened proportionally for parameter values further away from 2.0. For c_4 the two bins $[2.0, 2.5)$ and $[3.5, 4.0)$ clearly had the best results, while the values between 2.5 and 3.5 scored worse. Combining label A and label B, the values $c_4 < 1.5$ scored worse, but all values $c_4 \geq 1.5$ performed at least reasonably. For parameter k, high values led to a higher proportion of label A results, but all values $k \geq 6$ scored comparably. Combining label A and label B contributions, there was no significant difference between the performance of each of the nine values of k: any value of k led to the same number of reasonable parameter value combinations.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

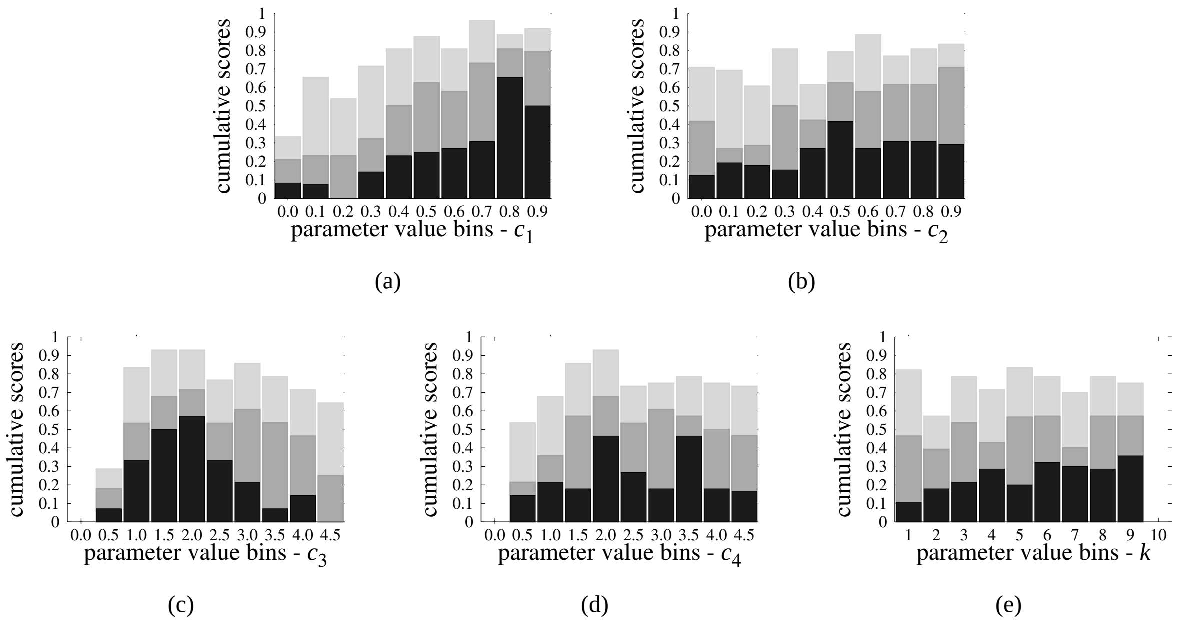


Figure 5.3: Sensitivity analysis Von Neumann SBPSO parameters: (a) c_1, (b) c_2, (c) c_3, (d) c_4, and (e) k.

5.4.6 Relative importance of SBPSO's control parameters

There is no reason to assume that all SBPSO's control parameters have an equal impact on performance. The tuning experiments described above instead indicate that the five control parameters each effect performance differently. For example, the conclusion in sections 5.4.3 to 5.4.5 for parameter k was that very little difference was seen between values 1 through 9 with respect to reasonable-to-good performance. In contrast, for parameter c_1, values of 0.8 or higher clearly were an indication of better performance, while values of 0.3 or lower were detrimental. Therefore, the performance of the SBPSO algorithm on the MKP is more sensitive to parameter c_1, than to parameter k. This section contains a systematic investigation of the relative sensitivity of the five SBPSO parameters.

A measure of the distribution of performance of an individual parameter can serve as an indication of the sensitivity of SBPSO to that parameter. As argued in section 5.4.1, most information about the performance of an individual parameter can be gained from looking at "good" parameter value combinations only, where good was defined as the best 25% (label A) parameter value combinations. Therefore, for each individual parameter, the distribution of the label A combinations was used as a proxy for the distribution of the performance.

For each parameter, and each of the three topologies, the distribution of label A combinations across bins was converted to a single measurement using the following steps:

1. For each bin, the fraction of label A parameter value combinations was obtained, and the fractions themselves were ordered from high to low.
2. The sum of the *highest* five fractions was labeled $fraction_{\text{high}}$.
3. The sum of the *lowest* five fractions was labeled $fraction_{\text{low}}$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

4. The sensitivity score was then defined as the difference, $fraction_{\text{high}} - fraction_{\text{low}}$.

Note that for parameters c_3, c_4, and k, only nine bins were used, such that the bin ranked fifth was included in both $fraction_{\text{high}}$ and $fraction_{\text{low}}$ and drops out of the sensitivity score.

The sensitivity score ranges between 0% and 100%. A score of 0% means that all bins contained exactly the same fraction of label A combinations, indicating that good parameter value combinations show little to no sensitivity to the individual parameter. A score of 100% means that at least five bins contained *zero* label A combinations, but that these combinations are instead concentrated in the remaining bins. For this case, good parameter value combinations show a high sensitivity to the individual parameter.

Table 5.8 summarizes the resulting sensitivity score for each individual parameter, split by the topology used, and ranks the sensitivity scores of the five parameters for each topology.

Table 5.8: Performance distribution per individual control parameter

parameter	GB SBPSO	rank GB	LB SBPSO	rank LB	VN SBPSO	rank VN
c_1	58%	(1)	52%	(2)	48%	(2)
c_2	31%	(4)	25%	(4)	16%	(5)
c_3	53%	(2)	62%	(1)	65%	(1)
c_4	41%	(3)	39%	(3)	33%	(3)
k	20%	(5)	22%	(5)	25%	(4)

The sensitivity scores indicated that the performance of SBPSO had the highest sensitivity to control parameters c_1 (attraction to the personal best) and c_3 (the maximum number of elements to add to the solution set randomly). Hence, it can be concluded that, when applying SBPSO to the MKP, these two parameters are the most important to be tuned well. This result held for all three topologies investigated. All three topologies were the least sensitive to parameters c_2 (attraction to the neighborhood best) and k (the size of the tournament used).

Note that an equal amount of tuning effort was expended on all five SBPSO parameters: the process described in section 5.3.1 meant finding the best out of 128 randomly chosen parameter value combinations spread evenly across the five dimensional parameter space.

5.5 Experimental results

This section describes the results of the experiments conducted on the MKP. The benchmark problems that are used in testing the tuned PSO algorithms were listed in section 5.2.1. The process used to tune each of the twelve PSO algorithm-topology pairs on the small and large MKPs separately was outlined in section 5.3. The experiments were split into those involving the small MKPs, with results given in sections 5.5.1, and those involving the large MKPs with results given in section 5.5.2.

For both the small MKPs and the large MKPs, the respective results sections each contain five tables comparing the performance of the algorithm-topology pairs: the first three tables each summarize and compare the performance of the four PSO algorithms using a single topology. The fourth table compares the results of each of the four PSO algorithms, across all of the topologies. The final table has more de-

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

tailed results per problem and compares the four PSO algorithms using each algorithm's best performing topology.

The performance of the algorithm-topology pairs was compared for statistical significance using the Iman-Davenport (ID) test and post-hoc analysis was conducted using the Nemenyi-test. Details on the statistical procedure used can be found in appendix E.

The Z-score from the ID-test, the associated p-value, and the Holm-adjusted α are provided in the bottom rows of each of the results tables. If a p-value is smaller than the corresponding Holm α, the algorithm-topology pair *underperformed* the best pair in the comparison by a statistically significant margin. For the best performing algorithm-topology pair, the average error or classification accuracy is shown in **bold**. If the ID-test indicated a statistically significant difference in performance, but the Nemenyi post-hoc tests did not indicate a single best pair, all algorithm-topology pairs that were indistinguishable from the best are shown in bold.

5.5.1 Testing results for small MKP

This section list the results of the test experiments on the tuned algorithm-topology pairs on the small MKPs, each using the parameters listed in table 5.4. First a comparison is made across algorithms for each of the three topologies in section 5.5.1.1, to determine which algorithm performs best when using the same topology. Following this, in section 5.5.1.2 a comparison is made across topologies for each of the four PSO algorithms, to determine which topology is best for use with each PSO algorithm on the small MKPs.

Both sections 5.5.1.1 and 5.5.1.2 are split into subsections per topology and algorithm respectively. Each such subsection contains a table with a summary of the results. Appendix A provides the same comparisons on the level of single problems used in testing.

5.5.1.1 Results per topology

This section compares the performance of the four PSO algorithms on the small MKPs by topology: the star topology in section 5.5.1.1.1, the ring topology in section 5.5.1.1.2, and the Von Neumann topology in section 5.5.1.1.3.

Each section contains a table listing the average and standard deviation of the error (the best fitness found compared to the known optimum), and the average rank of the errors. This is followed by the average and standard deviation of the success rate (shortened SR in the tables), and the average rank of the success rate. The success rate of an algorithm-topology pair on a single MKP was defined as the percentage of independent runs that were successful in finding the optimum. The next two rows in each table shed light on the consistency of the algorithm: the row labeled "# perfect" reports the number of problems for which all independent runs found the optimum, and the row labeled "# failure" reports the number of problems for which all independent runs failed to find the optimum.

For the algorithm-topology comparisons that are reported in each of the tables in this section, the ID-test indicated that the median performance showed statistically significant differences. Hence, in all three cases, post-hoc tests were conducted and the results are reported at the bottom of the respective

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

tables.

5.5.1.1.1 Star topology Table 5.9 shows that the gbest SBPSO outperformed the other three algorithms with a star topology by a statistically significant margin. If success rate was used as the performance measure instead of average error, gbest SBPSO also performed best in a statistically significant manner (p-values and α's are not shown). The average success rate of gbest SBPSO was 82.5%, while the second best performer was gbest PBPSO with an average success rate of 51.4%.

Table 5.9: Summary of small MKP test results for the star topology. Bold face indicates statistically significant outperformance.

	GB BPSO		GB MBPSO		GB PBPSO		GB SBPSO	
problem	error	(rank)	error	(rank)	error	(rank)	error	(rank)
average error	1.117 %	(2.80)	1.089 %	(3.56)	0.628 %	(2.45)	**0.444 %**	(1.19)
stdev error	1.913 %		1.592 %		1.625 %		1.640 %	
average SR	42.8 %	(2.81)	29.9 %	(3.38)	51.4 %	(2.50)	82.5 %	(1.31)
stdev SR	41.3 %		34.7 %		35.1 %		31.8 %	
# perfect	5	(2.5)	3	(4)	5	(2.5)	21	(1)
# failure	11	(4)	4	(2)	4	(2)	4	(2)
Z-score		5.58		8.21		4.36		
p-value		0.0000		0.0000		0.0000		
Holm α		0.0250		0.0500		0.0167		

For all 40 problems, the success rate for the gbest SBPSO exceeded or matched that of the other three gbest PSO algorithms. Gbest SBPSO was also more consistent than the other gbest PSO algorithms, as the optimum was found in all independent runs for 21 out of 40 problems. For the other three algorithms, the optimum was found in all independent runs for at most five problems.

5.5.1.1.2 Ring topology Table 5.10 shows that the lbest SBPSO outperformed the other three algorithms with a ring topology by a statistically significant margin. If success rate was used as the performance measure instead of average error, lbest SBPSO also performed best in a statistically significant manner. The average success rate of lbest SBPSO was 81.9%, while the second best performer was lbest PBPSO, scoring an average success rate of 63.4%.

For 38 out of 40 problems, the success rate for the lbest SBPSO exceeded or matched that for the other three lbest PSO algorithms. Lbest SBPSO was also more consistent than the other local best PSO algorithms, as the optimum was found in all independent runs for 23 out of the 40 problems. For the other three algorithms, the optimum was found in all independent runs for at most 12 problems. Note that the number of problems solved perfectly by lbest PBPSO (that is, 12) is significantly higher than was the case for the gbest PBPSO (that is, five).

5.5.1.1.3 Von Neumann topology Table 5.11 shows that SBPSO with a Von Neumann topology outperformed the other three PSO algorithms by a statistically significant margin. If success rate was used

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 5.10: Summary of small MKP test results for the ring topology. Bold face indicates statistically significant outperformance.

	LB BPSO		LB MBPSO		LB PBPSO		LB SBPSO	
problem	error	(rank)	error	(rank)	error	(rank)	error	(rank)
average error	0.841 %	(2.95)	0.639 %	(3.35)	0.521 %	(2.31)	**0.440 %**	(1.39)
stdev error	1.716 %		1.620 %		1.634 %		1.641 %	
average SR	50.3 %	(2.93)	45.7 %	(3.24)	63.4 %	(2.28)	81.9 %	(1.56)
stdev SR	43.3 %		37.4 %		36.8 %		33.2 %	
# perfect	7	(3)	4	(4)	12	(2)	23	(1)
# failure	10	(4)	4	(2)	4	(2)	4	(2)
Z-score		5.40		6.79		3.19		
p-value		0.0000		0.0000		0.0007		
Holm α		0.0250		0.0500		0.0167		

as the performance measure instead of average error, SBPSO with a Von Neumann topology also performed best in a statistically significant manner. The average success rate of the Von Neumann SBPSO was 82.7%, while the second best performer was the Von Neumann PBPSO with an average success rate of 64.8%.

Table 5.11: Summary of small MKP test results for the Von Neumann topology. Bold face indicates statistically significant outperformance.

	VN BPSO		VN MBPSO		VN PBPSO		VN SBPSO	
	BPSO	VN	MBPSO	VN	PBPSO	VN	SBPSO	VN
problem	error	(rank)	error	(rank)	error	(rank)	error	(rank)
average error	0.609 %	(2.81)	0.613 %	(3.45)	0.510 %	(2.28)	**0.439 %**	(1.46)
stdev error	1.635 %		1.623 %		1.633 %		1.641 %	
average SR	56.6 %	(2.76)	48.6 %	(3.31)	64.8 %	(2.36)	82.7 %	(1.56)
stdev SR	41.1 %		35.3 %		36.8 %		32.6 %	
# perfect	9	(3)	4	(4)	12	(2)	25	(1)
# failure	6	(4)	4	(2)	4	(2)	4	(2)
Z-score		4.68		6.89		2.84		
p-value		0.0000		0.0000		0.0023		
Holm α		0.0250		0.0500		0.0167		

For 37 out of the 40 problems the success rate for SBPSO with the Von Neumann topology exceeded or matched that for the other three PSO algorithms. SBPSO was also more consistent than the other PSO algorithms using the Von Neumann topology, as the optimum was found in all independent runs for 23 out of 40 problems. For the other three algorithms, the optimum was found in all independent runs for at most 12 problems. The number of problems solved perfectly by PSO algorithms using the Von Neumann topology closely matched the results for the corresponding lbest PSO algorithms, with only lbest BPSO

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

(seven out of 40) scoring differently than BPSO with the Von Neumann topology (nine out of 40).

A problem by problem comparison of the four algorithms each using the Von Neumann topology can be found in table A.9 in appendix A. The four problems for which SBPSO with the Von Neumann topology failed to find the optimum in all independent runs are mknap2-6, mknap2-11, mknap2-13, and mknap2-18. The other 11 algorithm-topology pairs all similarly failed for these four problems. For the algorithm-topology pairs combining the Von Neumann topology with SBPSO, PBPSO, and MBPSO respectively, these four problems were also the only failures. For the Von Neumann-BPSO pair, additionally problems mknap2-43 and mknap2-47 caused failures.

Combining the results shown in tables 5.9, 5.10, and 5.11 one can seen that for each of the four PSO algorithms the Von Neumann topology performed best.

5.5.1.2 Results per algorithm

This section compares the performance of the four PSO algorithms on the small MKPs by algorithm: BPSO in section 5.5.1.2.1, MBPSO in section 5.5.1.2.2, PBPSO in section 5.5.1.2.3, and SBPSO in section 5.5.1.2.4. Each section contains a table listing the results of the experiment by algorithm. These tables are constructed as described in section 5.5.1.1, but here each table compares results across all algorithm-topology pairs that involve the same algorithm.

For three out of the four algorithm-topology comparisons (BPSO, MBPSO, PBSPO) that are reported in this section, the ID-test indicated that the median performance showed statistically significant differences. In these cases, post-hoc tests were conducted and the results are reported at the bottom of the respective tables. For SBPSO no statistically significant difference was found between the performance of using either of the three topologies with SBPSO.

5.5.1.2.1 BPSO Table 5.12 compares the results of the BPSO algorithm on the small MKP for the three different topologies tested: star (GB), ring (LB) and Von Neumann (VN).

Table 5.12: Summary of small MKP test results across topologies for BPSO. Bold face indicates statistically significant outperformance.

	GB BPSO		LB BPSO		VN BPSO	
Measure	error	(rank)	error	(rank)	error	(rank)
avg error	1.117 %	(2.65)	**0.841 %**	(1.80)	**0.609 %**	(1.55)
stdev error	1.913 %		1.716 %		1.635 %	
average SR	42.8 %	(2.45)	50.3 %	(2.03)	56.6 %	(1.53)
stdev SR	41.3 %		43.3 %		41.1 %	
# perfect	5	(3)	7	(2)	9	(1)
# failure	11	(3)	10	(2)	6	(1)
Z-score		4.92		1.12		
p-value		0.0000		0.1314		
Holm α		0.0500		0.0250		

For BPSO the ID-test yielded a p-value less than 0.0001, indicating that a statistically significant

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

difference in performance existed. It was the star topology that underperformed, while the difference in performance between the ring topology and the Von Neumann topology yielded a p-value of 0.1314 using the Nemenyi post-hoc test at a Holm α of 0.0250. Therefore, although Von Neumann BPSO performed best, the difference in error with lbest BPSO was not statistically significant. The Von Neumann BPSO also scored best on the average success rate, the number of problems solved perfectly, and the number of problems on which the algorithm failed.

5.5.1.2.2 MBPSO Table 5.13 compares the results of the MBPSO algorithm on the small MKP for the three different topologies tested: star (GB), ring (LB) and Von Neumann (VN).

Table 5.13: Summary of small MKP test results across topologies for MBPSO. Bold face indicates statistically significant outperformance.

	GB MBPSO		LB MBPSO		VN MBPSO	
Measure	error	(rank)	error	(rank)	error	(rank)
avg error	1.089 %	(2.93)	**0.639 %**	(1.65)	**0.613 %**	(1.43)
stdev error	1.592 %		1.620 %		1.623 %	
average SR	29.9 %	(2.78)	45.7 %	(1.68)	48.6 %	(1.55)
stdev SR	34.7 %		37.4 %		35.3 %	
# perfect	3	(3)	4	(1.5)	4	(1.5)
# failure	4	(2)	4	(2)	4	(2)
Z-score		6.71		0.98		
p-value		0.0000		0.1635		
Holm α		0.0500		0.0250		

For MBPSO the ID-test yielded a p-value less than 0.0001, indicating that a statistically significant difference in performance existed. It was the star topology that underperformed, while the difference in performance between the ring topology and the Von Neumann topology yielded a p-value of 0.1635 using the Nemenyi post-hoc test at a Holm α of 0.0250. Therefore, although Von Neumann MBPSO performed best, the difference in error with lbest MBPSO was not statistically significant. There was little difference in the number of problems which the MBPSO algorithm-topology pairs solved perfectly, and no difference at all in the number of problems on which they failed. With reference to success rate, gbest MBPSO clearly underperformed lbest MBPSO and Von Neumann MBPSO.

5.5.1.2.3 PBPSO Table 5.14 compares the results of the PBPSO algorithm on the small MKP for the three different topologies tested: star (GB), ring (LB) and Von Neumann (VN).

For PBPSO the ID-test yielded a p-value less than 0.0001, indicating that a statistically significant difference in performance existed. It was the star topology that underperformed, while the difference in performance between the ring topology and the Von Neumann topology yielded a p-value of 0.1515 using the Nemenyi post-hoc test at a Holm α of 0.0250. Therefore, although Von Neumann PBPSO performed best, the difference in error with lbest PBPSO was not statistically significant. In all listed measures, gbest PBPSO clearly underperformed, while there was very little difference between lbest

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 5.14: Summary of small MKP test results across topologies for PBPSO. Bold face indicates statistically significant outperformance.

	GB PBPSO		LB PBPSO		VN PBPSO	
Measure	error	(rank)	error	(rank)	error	(rank)
avg error	0.628 %	(2.68)	**0.521 %**	(1.78)	**0.510 %**	(1.55)
stdev error	1.625 %		1.634 %		1.633 %	
average SR	51.4 %	(2.4)	63.4 %	(1.9)	64.8 %	(1.7)
stdev SR	35.1 %		36.8 %		36.8 %	
# perfect	5	(3)	12	(1.5)	12	(1.5)
# failure	4	(2)	4	(2)	4	(2)
Z-score		5.05		1.03		
p-value		0.0000		0.1515		
Holm α		0.0500		0.0250		

PBPSO and the Von Neumann PBPSO, with tied scores in the number of perfectly solved problems as well as the number of problems on which they both failed.

5.5.1.2.4 SBPSO Table 5.15 compares the results of the SBPSO algorithm on the small MKP for the three different topologies tested: star (GB), ring (LB) and Von Neumann (VN).

Table 5.15: Summary of small MKP test results across topologies for SBPSO. Bold face indicates statistically significant outperformance.

	GB SBPSO		LB SBPSO		VN SBPSO	
Measure	error	(rank)	error	(rank)	error	(rank)
avg error	0.444 %	(2.13)	0.440 %	(2.01)	0.439 %	(1.86)
stdev error	1.640 %		1.641 %		1.641 %	
average SR	82.5 %	(2.09)	81.9 %	(2.04)	82.7 %	(1.88)
stdev SR	31.8 %		33.2 %		32.6 %	
# perfect	21	(3)	23	(2)	25	(1)
# failure	4	(2)	4	(2)	4	(2)
Z-score		1.21		0.67		
p-value		0.1131		0.2514		
Holm α		0.0500		0.0250		

For SBPSO, the ID-test yielded a *p*-value of 0.5134, which indicated that the null hypothesis of equal performance of gbest SBPSO, lbest SBPSO, and Von Neumann SBPSO was *not* rejected. Therefore, no statistically significant difference in performance could be found between the three topologies for SBPSO.

The listed measures for SBPSO all indicated that there was little difference in performance between the three SBPSO algorithm-topology pairs: the relative difference in the average errors of the three pairs was 1.1%, while the relative difference in the average success rate of the three pairs was 1.0%. Only the number of problems solved perfectly showed some differentiation, as gbest SBPSO solved 21 out of the

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

40 problems perfectly, while lbest SBPSO completely solved 23 problems, and Von Neumann SBPSO 25 problems.

Excluding the four problems on which SBPSO completely failed to find the optimum (a success rate of 0%), the *lowest success rate* recorded for SBPSO on any of the remaining 36 problems was reasonable: 50% for gbest SBPSO (average success rate on the 36 problems of 89.2%), 20% for lbest SBPSO (average success rate of 88.5%), and 30% for SBPSO using the Von Neumann topology (average success rate of 89.4%). Thus SBPSO can be said to be successful not only in achieving a low average error, but it is able to find the actual optimum in a good portion of the independent runs. Gbest SBPSO score best of all twelve algorithm-topology pairs in this measure of minimum success rate, although not by a significant margin.

5.5.2 Testing results for large MKP

This section list the results of the test experiments on the tuned algorithm-topology pairs on the large MKP, each using the parameters listed in table 5.6. First a comparison is made across algorithms for each of the three topologies in section 5.5.2.1, to determine which algorithm performs best when using the same topology. Following this, in section 5.5.2.2 a comparison is made across topologies for each of the four PSO algorithms, to determine which topology is best for use with each PSO algorithm on the large MKP. Finally, a comparison is made between four algorithm-topology pairs, one for each of the four PSO algorithms combined with the topology that performed best for that algorithm. The results of this comparison are given in section 5.5.2.3.

Both sections 5.5.2.1 and 5.5.2.2 are split further into subsections per topology and algorithm respectively. Each such subsection contains a table with a summary of the results. Appendix B provides the same comparisons on the level of single problems used in testing.

5.5.2.1 Results per topology

This section shows the results of the test experiments performed on the large MKP, comparing performance of the four PSO algorithms (BPSO, MBPSO, PBPSO, and SBPSO) across one topology at a time. Results are summarized in three tables 5.16, 5.17, and 5.18 for the star, ring, and Von Neumann topology respectively. Each table lists the average and standard deviation of the error (the best fitness found compared to the LP relaxation bound), and the average rank of the errors. The average error is shown on three different cross-sections of the problem set:

1. The number of items, n, with values 100, 250, and 500.
2. The number of constraints, m, with values 5, 10, and 30.
3. The tightness ratio, r, with values 0.25, 0.50, and 0.75.

Please refer to section 2.3 for details on these parameters and the problem set.

The ID-test indicated that, for the algorithm-topology comparisons that are reported in each of the tables, the median performance showed statistically significant differences. Hence, where required post-hoc tests were conducted and the results are reported at the bottom of the respective tables.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.5.2.1.1 Star topology Table 5.16 summarizes the large MKP results for the four PSO algorithms, each using the star topology. The table shows that the gbest SBPSO was the best performing algorithm: it scored the smallest average error of 1.74%, and the average rank of the error shown on the same line was exactly 1, meaning that gbest SBPSO was the best performing algorithm on each of the 243 test problems. The post-hoc tests showed that the outperformance of gbest SBPSO was also statistically significant: pair-wise comparisons with the three other PSO algorithms yielded Z-scores above 10, which resulted in p-values smaller than 10^{-22}. Gbest PBPSO was the second best performer on 193 problems, gbest BPSO performed second best for the remaining 50 problems, and gbest MBPSO usually ranked last out of the four algorithm-topology pairs.

Table 5.16: Summary of large MKP test results for the star topology. Bold face indicates statistically significant outperformance.

Measure		GB BPSO error	(rank)	GB MBPSO error	(rank)	GB PBPSO error	(rank)	GB SBPSO error	(rank)
average error		4.679 %	(2.909)	5.619 %	(3.885)	3.250 %	(2.206)	**1.740 %**	(1.000)
stdev error		3.468 %		2.723 %		1.718 %		1.170 %	
n	100	3.831 %	(2.877)	5.160 %	(3.889)	2.568 %	(2.235)	**1.260 %**	(1.000)
n	250	4.679 %	(2.877)	5.663 %	(3.889)	3.286 %	(2.235)	**1.758 %**	(1.000)
n	500	5.526 %	(2.975)	6.034 %	(3.877)	3.896 %	(2.148)	**2.201 %**	(1.000)
m	5	3.037 %	(2.383)	4.354 %	(4.000)	3.134 %	(2.617)	**1.875 %**	(1.000)
m	10	3.942 %	(3.012)	5.521 %	(3.988)	2.763 %	(2.000)	**1.553 %**	(1.000)
m	30	7.057 %	(3.333)	6.983 %	(3.667)	3.853 %	(2.000)	**1.791 %**	(1.000)
r	0.25	8.253 %	(3.122)	8.664 %	(3.659)	5.264 %	(2.220)	**3.141 %**	(1.000)
r	0.50	3.751 %	(2.831)	5.344 %	(4.000)	2.799 %	(2.169)	**1.355 %**	(1.000)
r	0.75	1.909 %	(2.769)	2.712 %	(4.000)	1.613 %	(2.231)	**0.676 %**	(1.000)
Z-score			16.30		24.63		10.30		
p-value			0.0000		0.0000		0.0000		
Holm α			0.0250		0.0500		0.0167		

The relative performance of the four PSO algorithms using the star topology was stable across each of the three splits of the problem set, with gbest SBPSO $>$ gbest PBPSO $>$ gbest BPSO $>$ gbest MBPSO in each individual split except one: for the $243/3 = 81$ problems with $m = 5$, gbest BPSO (average rank 2.383) scored better than gbest PBPSO (average rank 2.617). Here the symbol '$>$' is used to mean "has a lower (better) average rank than".

A difference in performance was seen with regards to the split of the problems based on the number of items, n: a larger number of items led to a higher average error for each of the gbest PSO algorithms. However, this effect was not equally strong for each of the algorithms: for problems with $n = 500$ compared to those with $n = 100$, the average error of gbest SBPSO was 75% higher, while for gbest MBPSO the increase in average error was only 16%.

Problems with tightness ratio $r = 0.25$ were most challenging for all gbest PSO algorithms, with the average error substantially higher than for problems with $r = 0.50$ or 0.75. A smaller r means that each of the m weight constraints is more restrictive (lower capacity), which, *in general*, has two effects on the

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

optimal solution compared to that for problems with a higher tightness ratio:

1. the optimal solution using a small *r* contains fewer items, and
2. the fitness value at the optimum using a small *r* is lower, as fewer items are included in the knapsack.

Detailed results for the star topology on the large MKPs can be found in table B.7 in appendix B.

5.5.2.1.2 Ring topology Table 5.17 summarizes the large MKP results for the four PSO algorithms, each using the ring topology. The table shows that the lbest SBPSO was the best performing algorithm with an average rank of 1.333. The ID-test and post-hoc tests confirmed that lbest SBPSO outperformed each of the other three pairs, but the difference in performance between lbest SBPSO and lbest PBPSO was smaller than that seen between gbest SBPSO and gbest PBPSO in table 5.16.

Table 5.17: Summary of large MKP test results for the ring topology. Bold face indicates statistically significant outperformance.

Measure		LB BPSO error	(rank)	LB MBPSO error	(rank)	LB PBPSO error	(rank)	LB SBPSO error	(rank)
average error		7.006 %	(3.737)	3.922 %	(2.959)	3.650 %	(1.971)	**2.292 %**	(1.333)
stdev error		5.037 %		2.059 %		2.591 %		1.331 %	
n	100	6.348 %	(3.778)	3.044 %	(2.852)	3.101 %	(2.037)	**1.767 %**	(1.333)
n	250	6.951 %	(3.753)	3.917 %	(2.938)	3.626 %	(1.975)	**2.366 %**	(1.333)
n	500	7.719 %	(3.679)	4.805 %	(3.086)	4.221 %	(1.901)	**2.743 %**	(1.333)
m	5	3.091 %	(3.210)	3.289 %	(3.790)	**1.994 %**	(1.000)	2.334 %	(2.000)
m	10	7.520 %	(4.000)	3.654 %	(2.963)	3.112 %	(2.037)	**2.075 %**	(1.000)
m	30	10.407 %	(4.000)	4.824 %	(2.123)	5.842 %	(2.877)	**2.468 %**	(1.000)
r	0.25	11.961 %	(3.829)	6.185 %	(2.817)	6.059 %	(2.024)	**3.893 %**	(1.329)
r	0.50	5.817 %	(3.723)	3.608 %	(3.060)	3.066 %	(1.892)	**1.957 %**	(1.325)
r	0.75	3.063 %	(3.654)	1.878 %	(3.000)	1.738 %	(2.000)	**0.966 %**	(1.346)
Z-score			20.53		13.88		5.45		
p-value			0.0000		0.0000		0.0000		
Holm α			0.0500		0.0250		0.0167		

The relative performance of the four PSO algorithms using the ring topology was stable across each of the three splits of the problem set into three subsets, with lbest SBPSO $>$ lbest PBPSO $>$ lbest MBPSO $>$ lbest BPSO, except for two cases:

1. for the problems with $m = 5$, lbest PBPSO (average rank 1.000) scored better than lbest SBPSO (average rank 2.000) on all 81 problems in the subset, while lbest BPSO (average rank 3.210) scored better than lbest MBPSO (average rank 3.790), and
2. for the problems with $m = 30$, lbest MBPSO (average rank 2.123) scored better than lbest PBPSO (average rank 2.877).

The relative performance of the lbest MBPSO and lbest PBPSO algorithm-pairs was correlated with the number of constraints, *m*: lbest MBPSO performed relatively better for an increasing number of

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

constraints, while lbest PBPSO performed relatively worse with increasing m. For both lbest PBPSO and lbest MBPSO the average error increased when m increased, but for lbest PBPSO this deterioration was worse. For all the lbest PSO algorithms, the average error was most sensitive to changes in r.

A possible explanation for lbest PBPSO having outperformed lbest SBPSO on problems with $m = 5$, is that the lbest SBPSO algorithm was better tuned to the problems with a larger number of constraints ($m = 10$ or 30), while the lbest PBPSO algorithm was better tuned for problems with fewer constraints. An alternative explanation is that the k-tournament selection used in LB SBPSO helped the particles to stay in the feasible part of the solution space. This feature has extra value in the case of a larger number of constraints, where particles will encounter the edge of the feasible part of the solution space more often.

Detailed results for the ring topology on the large MKPs can be found in table B.8 in appendix B.

5.5.2.1.3 Von Neumann topology Table 5.18 shows that the Von Neumann SBPSO was the best performing algorithm with an average rank of 1.342. The ID-test and post-hoc tests confirmed that the Von Neumann SBPSO outperformed each of the other three pairs, with the Von Neumann PBPSO scoring second best. The difference in performance between the Von Neumann SBPSO and the Von Neumann PBPSO was approximately the same as seen between lbest SBPSO and lbest PBPSO in table 5.17.

Table 5.18: Summary of large MKP test results for the von Neumann topology. Bold face indicates statistically significant outperformance.

Measure		VN BPSO error	(rank)	VN MBPSO error	(rank)	VN PBPSO error	(rank)	VN SBPSO error	(rank)
average error		6.973 %	(3.823)	3.403 %	(2.811)	3.348 %	(2.025)	**2.249 %**	(1.342)
stdev error		5.039 %		1.742 %		2.533 %		1.275 %	
n	100	6.291 %	(3.815)	2.647 %	(2.790)	2.762 %	(2.049)	**1.772 %**	(1.346)
n	250	6.920 %	(3.864)	3.418 %	(2.765)	3.330 %	(2.037)	**2.294 %**	(1.333)
n	500	7.707 %	(3.790)	4.145 %	(2.877)	3.954 %	(1.988)	**2.680 %**	(1.346)
m	5	3.076 %	(3.469)	2.980 %	(3.506)	**1.783 %**	(1.000)	2.433 %	(2.025)
m	10	7.465 %	(4.000)	3.191 %	(2.914)	2.847 %	(2.086)	**2.046 %**	(1.000)
m	30	10.377 %	(4.000)	4.039 %	(2.012)	5.416 %	(2.988)	**2.266 %**	(1.000)
r	0.25	11.943 %	(3.976)	5.382 %	(2.610)	5.739 %	(2.085)	**3.789 %**	(1.329)
r	0.50	5.775 %	(3.855)	3.089 %	(2.819)	2.693 %	(2.000)	**1.917 %**	(1.325)
r	0.75	3.023 %	(3.628)	1.658 %	(3.013)	1.533 %	(1.987)	**0.981 %**	(1.372)
Z-score			21.18		12.54		5.83		
P-value			0.0000		0.0000		0.0000		
Holm α			0.0500		0.0250		0.0167		

The relative behavior of the four PSO algorithms using the Von Neumann topology was the same as that seen for the lbest PSO algorithms in table 5.17: across each of the three splits of the problem set, the result was Von Neumann SBPSO $>$ Von Neumann PBPSO $>$ Von Neumann MBPSO $>$ Von Neumann BPSO in each individual split, except for two cases:

1. for the problems with $m = 5$, the Von Neumann PBPSO (average rank 1.000) performed best on

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

all 81 problems in the subset, with the Von Neumann SBPSO (average rank 2.025) scoring second best. Also the Von Neumann BPSO (average rank 3.469) narrowly outperformed the Von Neumann MBPSO (average rank 3.506), and

2. for the problems with $m = 30$, the Von Neumann MBPSO (average rank 2.012) scored better than the Von Neumann PBPSO (average rank 2.988).

Detailed results for the Von Neumann topology on the large MKPs can be found in table B.9 in appendix B.

5.5.2.2 Results per algorithm

This section shows the results of the test experiments performed on the large MKP, comparing performance of the three topologies (star, ring, and Von Neumann) across one PSO algorithm at a time. Results are summarized in four tables 5.19, 5.20, 5.21, and 5.22 for BPSO, MBPSO, PBPSO, and SBPSO respectively. Each table lists the average and standard deviation of the error (the best fitness found compared to the LP relaxation bound), and the average rank of the errors. The average error is shown on three different cross-sections of the problem set:

1. The number of items, n, with values 100, 250, and 500.
2. The number of constraints, m, with values 5, 10, and 30.
3. The tightness ratio, r, with values 0.25, 0.50, and 0.75.

Please refer to section 2.3 for details on these parameters and the problem set.

The ID-test indicated that, for all the algorithm-topology comparisons reported in this section, the median performance showed statistically significant differences. Hence, post-hoc tests were conducted and the results are reported at the bottom of the respective tables resulting in each case in one topology outperforming the other two.

5.5.2.2.1 BPSO Results comparing the BPSO algorithm across three different topologies on the large MKPs are given in table 5.19. The gbest BPSO performed much better than BPSO using either of the other two topologies. The average error was 4.68% for gbest BPSO, with lbest BPSO and the Von Neumann BPSO scoring 7.01% and 6.97% respectively. The gbest BPSO scored best on 198 out of 243 problems, but was outperformed on problems with few constraints ($m = 5$) combined with a high tightness ratio of $r = 0.75$. Here gbest BPSO performed worst out of the three BPSO pairs on the entire subset of 27 problems. For problems with $m = 5$ and $r = 0.5$, gbest BPSO's performance was comparable to the other two pairs and yielded an average rank of 1.944. Detailed results for the BPSO algorithm on the large MKPs can be found in table B.11 in appendix B.

5.5.2.2.2 MBPSO Results comparing the MBPSO algorithm across three different topologies on the large MKPs are given in table 5.20. For MBPSO, the relative performance of the three topologies was very stable across the entire problem set with the Von Neumann MBPSO scoring the best (with an average

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 5.19: Summary of large MKP test results across topologies for BPSO. Bold face indicates statistically significant outperformance.

Measure		GB BPSO error	(rank)	LB BPSO error	(rank)	VN BPSO error	(rank)
average error		**4.679 %**	(1.340)	7.006 %	(2.510)	6.973 %	(2.150)
stdev error		3.468 %		5.000 %		5.000 %	
n	100	**3.831 %**	(1.296)	6.348 %	(2.537)	6.291 %	(2.167)
n	250	**4.679 %**	(1.327)	6.951 %	(2.531)	6.920 %	(2.142)
n	500	**5.526 %**	(1.395)	7.719 %	(2.451)	7.707 %	(2.154)
m	5	**3.037 %**	(2.019)	3.091 %	(2.179)	**3.076 %**	(1.802)
m	10	**3.942 %**	(1.000)	7.520 %	(2.704)	7.465 %	(2.296)
m	30	**7.057 %**	(1.000)	10.407 %	(2.636)	10.377 %	(2.364)
r	0.25	**8.253 %**	(1.037)	11.961 %	(2.555)	11.943 %	(2.409)
r	0.50	**3.751 %**	(1.307)	5.817 %	(2.530)	5.775 %	(2.163)
r	0.75	**1.909 %**	(1.692)	3.063 %	(2.429)	**3.023 %**	(1.878)
p-value					0.0000		0.0000
Holm α					0.0250		0.0167

rank of 1.010), lbest MBPSO achieved an average rank of 1.990, and gbest MBPSO scored worst on all problems. The Von Neumann MBPSO failed to outperform lbest MBPSO on only three of the 243 problems. Detailed results for the MBPSO algorithm on the large MKPs can be found in table B.12 in appendix B.

Table 5.20: Summary of large MKP test results across topologies for MBPSO. Bold face indicates statistically significant outperformance.

Measure		GB MBPSO error	(rank)	LB MBPSO error	(rank)	VN MBPSO error	(rank)
average error		5.619 %	(3.000)	3.922 %	(1.990)	**3.403 %**	(1.010)
stdev error		2.723 %		2.100 %		1.700 %	
n	100	5.160 %	(3.000)	3.044 %	(1.988)	**2.647 %**	(1.012)
n	250	5.663 %	(3.000)	3.917 %	(2.000)	**3.418 %**	(1.000)
n	500	6.034 %	(3.000)	4.805 %	(1.975)	**4.145 %**	(1.025)
m	5	4.354 %	(3.000)	3.289 %	(1.963)	**2.980 %**	(1.037)
m	10	5.521 %	(3.000)	3.654 %	(2.000)	**3.191 %**	(1.000)
m	30	6.983 %	(3.000)	4.824 %	(2.000)	**4.039 %**	(1.000)
r	0.25	8.664 %	(3.000)	6.185 %	(2.000)	**5.382 %**	(1.000)
r	0.50	5.344 %	(3.000)	3.608 %	(1.988)	**3.089 %**	(1.012)
r	0.75	2.712 %	(3.000)	1.878 %	(1.974)	**1.658 %**	(1.026)
p-value			0.0000		0.0000		
Holm α			0.0250		0.0167		

5.5.2.2.3 PBPSO Results comparing the PBPSO algorithm across three different topologies on the large MKPs are given in table 5.21. The Von Neumann PBPSO performed best with reference to the

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

average rank of errors, with an average rank of 1.5. However, gbest PBPSO achieved a lower average error, scoring 3.25% while the Von Neumann PBPSO had an average error of 3.35%. This can be explained by the more consistent behavior of gbest PBPSO: its standard deviation of the error was 1.72%, while for the Von Neumann PBPSO this was 2.5%. The Von Neumann PBPSO scored well for problems with $m = 5$, but scored badly for problems with $m = 30$: the difference in average error on the two subsets was 5.42% − 1.78% = 3.63%. For gbest PBPSO the sensitivity to the problem parameter m was much smaller, and the difference between the subset on which it performed best ($m = 10$) and worst ($m = 30$) was only 3.85% − 2.76% = 1.09%. Detailed results for the PBPSO algorithm on the large MKPs can be found in table B.13 in appendix B.

Table 5.21: Summary of large MKP test results across topologies for PBPSO. Bold face indicates statistically significant outperformance.

Measure		GB PBPSO error	(rank)	LB PBPSO error	(rank)	VN PBPSO error	(rank)
average error		3.250 %	(1.860)	3.650 %	(2.650)	**3.348 %**	(1.500)
stdev error		1.718 %		2.600 %		2.500 %	
n	100	**2.568 %**	(1.790)	3.101 %	(2.667)	**2.762 %**	(1.543)
n	250	3.286 %	(1.864)	3.626 %	(2.654)	**3.330 %**	(1.481)
n	500	3.896 %	(1.914)	4.221 %	(2.617)	**3.954 %**	(1.469)
m	5	3.134 %	(3.000)	1.994 %	(2.000)	**1.783 %**	(1.000)
m	10	**2.763 %**	(1.568)	3.112 %	(2.951)	**2.847 %**	(1.481)
m	30	**3.853 %**	(1.000)	5.842 %	(2.988)	5.416 %	(2.012)
r	0.25	**5.264 %**	(1.695)	6.059 %	(2.646)	**5.739 %**	(1.659)
r	0.50	2.799 %	(1.904)	3.066 %	(2.663)	**2.693 %**	(1.434)
r	0.75	1.613 %	(1.974)	1.738 %	(2.628)	**1.533 %**	(1.397)
p-value			0.0011		0.0000		
Holm α			0.0167		0.0250		

5.5.2.2.4 SBPSO Results comparing the SBPSO algorithm across three different topologies on the large MKPs are given in table 5.22. The star topology was most successful, with gbest SBPSO performing best on all 243 problems. Little difference in performance was observed between lbest SBPSO and the Von Neumann SBPSO, which is probably related to the fact that the same control parameter values were used for both pairs (refer to table 5.6 for the parameter values). Hence, the only difference between the pairs was that the Von Neumann SBPSO has a more closely connected swarm compared to lbest SBPSO. Only for the split of the problem set based on the number of constraints, m, some difference in performance was seen between lbest SBPSO and the Von Neumann SBPSO, where lbest SBPSO performed better on problems with $m = 5$, and the Von Neumann PBPSO performed better on problems with $m = 30$. Considering the number of constraints, both lbest SBPSO and the Von Neumann SBPSO performed best on the subset of problems with $m = 10$. Having a more closely connected swarm helped the Von Neumann SBPSO on problems with more constraints. Detailed results for the SBPSO algorithm on the large MKPs can be found in table B.14 in appendix B.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 5.22: Summary of large MKP test results across topologies for SBPSO. Bold face indicates statistically significant outperformance.

Measure		GB SBPSO error	(rank)	LB SBPSO error	(rank)	VN SBPSO error	(rank)
average error		**1.740 %**	(1.000)	2.292 %	(2.570)	2.249 %	(2.430)
stdev error		1.170 %		1.300 %		1.300 %	
n	100	**1.260 %**	(1.000)	1.767 %	(2.519)	1.772 %	(2.481)
n	250	**1.758 %**	(1.000)	2.366 %	(2.636)	2.294 %	(2.364)
n	500	**2.201 %**	(1.000)	2.743 %	(2.543)	2.680 %	(2.457)
m	5	**1.875 %**	(1.000)	2.334 %	(2.173)	2.433 %	(2.827)
m	10	**1.553 %**	(1.000)	2.075 %	(2.549)	2.046 %	(2.451)
m	30	**1.791 %**	(1.000)	2.468 %	(2.975)	2.266 %	(2.025)
r	0.25	**3.141 %**	(1.000)	3.893 %	(2.659)	3.789 %	(2.341)
r	0.50	**1.355 %**	(1.000)	1.957 %	(2.566)	1.917 %	(2.434)
r	0.75	**0.676 %**	(1.000)	0.966 %	(2.468)	0.981 %	(2.532)
p-value					0.0000		0.0000
Holm α					0.0250		0.0167

5.5.2.3 Compare best topology per algorithm

The results in the previous sections showed that, for each PSO algorithm, a single topology performed best by a statistically significant margin: for MBPSO and PBPSO the Von Neumann topology scored best, while for BPSO and SBPSO it was the star topology that scored best. In order to see which algorithm has performed best on solving the large MKP, regardless of which topology was chosen, these four best algorithm-topology pairs are compared in this section. Note that a similar comparison was not performed for the small MKPs in section 5.5.1, as the Von Neumann topology was the best performing topology for each of the four PSO algorithms on the small MKPs.

A detailed comparison of the four PSO algorithms, each using its best performing topology, is given in table 5.23. The four best performing algorithm-topology pairs are gbest BPSO, Von Neumann MBPSO, Von Neumann PBPSO, and gbest SBPSO. With an average error of 1.72%, gbest SBPSO scored better than the other three pairs, with the second best pair, Von Neumann PBPSO, scoring an average error of 3.32 %. The ID-test followed by post-hoc tests indicated that gbest SBPSO outperformed the other three pairs by a statistically significant margin.

For gbest SBPSO, the average rank was 1.26, followed by Von Neumann PBPSO, Von Neumann MBPSO, and gbest BPSO with average ranks of 2.13, 2.81, and 3.80 respectively. The gbest SBPSO had the lowest error on 179 of the 243 problems, and was second best on the remaining 64, for which the Von Neumann PBPSO scored best each time. Gbest BPSO performed worst on 194 problems, and the second worst on the remaining 47.

Each of the first 27 rows of table 5.23 represents results for the subset of nine problems that correspond to the given MKP parameters n, m, and r. For all 27 problem subsets, the ID-test indicated a difference in performance across the four algorithm-topology pairs. However, in only two cases was a

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

single algorithm-topology pair shown to outperform the other three[3]: Gbest SBPSO statistically outperformed for $n = 100$, $m = 10$, $r = 0.25$ and $n = 250$, $m = 10$, $\alpha = 0.25$.

Table 5.23: Summary of large MKP test results for the best algorithm-topology pairs per algorithm. Bold face indicates statistically significant outperformance.

			GB BPSO		VN MBPSO		VN PBPSO		GB SBPSO	
n	m	r	error	(rank)	error	(rank)	error	(rank)	error	(rank)
100	5	0.25	3.772 %	(3.67)	3.625 %	(3.33)	**1.757 %**	(1)	**1.951 %**	(2)
100	5	0.50	1.835 %	(3.56)	1.774 %	(3.44)	**0.796 %**	(1)	**0.873 %**	(2)
100	5	0.75	1.155 %	(3.89)	1.077 %	(3.11)	**0.504 %**	(1.56)	**0.505 %**	(1.44)
100	10	0.25	5.334 %	(4)	4.064 %	(2.56)	4.052 %	(2.44)	**2.181 %**	(1)
100	10	0.50	2.384 %	(4)	2.160 %	(3)	**1.868 %**	(2)	**0.853 %**	(1)
100	10	0.75	1.217 %	(3.56)	1.206 %	(3.44)	**1.031 %**	(2)	**0.399 %**	(1)
100	30	0.25	11.396 %	(4)	**5.132 %**	(2)	8.411 %	(3)	**2.773 %**	(1)
100	30	0.50	4.786 %	(4)	**3.057 %**	(2)	3.902 %	(3)	**1.195 %**	(1)
100	30	0.75	2.327 %	(3.13)	**1.566 %**	(2)	2.364 %	(3.88)	**0.540 %**	(1)
250	5	0.25	5.182 %	(3.67)	4.995 %	(3.33)	**3.066 %**	(1)	**3.294 %**	(2)
250	5	0.50	2.680 %	(3.67)	2.617 %	(3.33)	**1.437 %**	(1)	**1.541 %**	(2)
250	5	0.75	1.390 %	(3.67)	1.362 %	(3.33)	**0.740 %**	(1.56)	**0.735 %**	(1.44)
250	10	0.25	6.917 %	(4)	4.895 %	(2.67)	4.801 %	(2.33)	**2.845 %**	(1)
250	10	0.50	3.161 %	(3.95)	2.904 %	(3.05)	**2.312 %**	(2)	**1.186 %**	(1)
250	10	0.75	1.546 %	(3.75)	1.522 %	(3.25)	**1.237 %**	(2)	**0.561 %**	(1)
250	30	0.25	12.765 %	(4)	**6.333 %**	(2)	9.356 %	(3)	**3.451 %**	(1)
250	30	0.50	5.556 %	(4)	**3.921 %**	(2)	4.308 %	(3)	**1.471 %**	(1)
250	30	0.75	2.738 %	(4)	**2.055 %**	(2)	2.595 %	(3)	**0.670 %**	(1)
500	5	0.25	6.349 %	(3.67)	6.299 %	(3.33)	**4.565 %**	(1)	**4.749 %**	(2)
500	5	0.50	3.134 %	(3.33)	3.219 %	(3.67)	**2.026 %**	(1.44)	**2.066 %**	(1.56)
500	5	0.75	1.837 %	(3.5)	1.849 %	(3.5	**1.157 %**	(1.33)	**1.160 %**	(1.67)
500	10	0.25	8.357 %	(4)	5.943 %	(3)	**5.652 %**	(2)	**3.404 %**	(1)
500	10	0.50	3.776 %	(4)	3.522 %	(3)	**2.677 %**	(2)	**1.455 %**	(1)
500	10	0.75	1.889 %	(3.5)	1.905 %	(3.5	**1.399 %**	(2)	**0.726 %**	(1)
500	30	0.25	14.189 %	(4)	**7.089 %**	(2)	10.003 %	(3)	**3.592 %**	(1)
500	30	0.50	6.398 %	(4)	**4.655 %**	(2.11)	4.822 %	(2.89)	**1.593 %**	(1)
500	30	0.75	3.083 %	(4)	**2.378 %**	(2)	2.811 %	(3)	**0.764 %**	(1)
average			4.635 %	(3.80)	3.375 %	(2.81)	3.320 %	(2.13)	**1.723 %**	(1.26)
Z-score				21.68		13.22		7.34		
p-value				0.0000		0.0000		0.0000		
Holm α				0.0500		0.0250		0.0167		

For each of the remaining 25 problems, the post-hoc tests did not indicate a single best algorithm-topology pair, but instead resulted in two best pairs with indistinguishable performance: no significant difference could be seen between the two best performing pairs, while the two worst pairs underper-

[3] Even if all problems yield the same ranks, resulting in average rankings of 1, 2, 3, and 4 for the four algorithm-topology pairs, the post-hoc Nemenyi test did *not* show a statistically significant difference between ranks 1 and 2, at a confidence level of $\alpha = 0.05$, which led to a Holm-α of 0.0167 for the comparison of the two best performing pairs.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

formed the best two in a statistically significant manner. For nine out of 25 problem specifications, all with $m = 30$, gbest SBPSO and the Von Neumann MBPSO performed best, while gbest BPSO and the Von Neumann PBPSO underperformed. For the remaining 16 out of 25 cases, gbest SBPSO and the Von Neumann PBPSO performed best, while gbest BPSO and the Von Neumann MBPSO underperformed.

The performance of the Von Neumann PBPSO deteriorated for larger values of m, compared to the other algorithm-topology pairs in table 5.23. The Von Neumann PBPSO outperformed the other three pairs on problems with $m = 5$, but the difference with gbest SBPSO became smaller for larger values of α. For problems with $m = 10$, the Von Neumann PBPSO performed second best on 74 out of 81 problems. However, for problems with $m = 30$, the Von Neumann PBPSO ranked better than third only once out of 81 problems, and performed worse than both gbest SBPSO and the Von Neumann MBPSO. As mentioned in the discussion of the results in table 5.17, the parameters chosen for the Von Neumann PBPSO (which were the same as for lbest PBPSO) were probably better suited to problems with a lower number of constraints.

5.6 Conclusions

This chapter had three objectives. The most important objective was to determine if it is possible to successfully apply the SBPSO algorithm to the MKP, where success meant yielding solutions of sufficient quality, but not necessarily solutions on par with algorithms that directly incorporate MKP specific heuristics. The investigation of SBPSO's efficiency in the number of iterations, fitness function evaluations or flops was explicitly set out of scope. The second objective was to compare the performance of the SBPSO algorithm on the MKP to that of three other PSO algorithms known from literature. The third objective was to investigate what parameter values work well for the SBPSO.

5.6.1 Comparing the algorithms

Comparing results on the test problems, SBPSO has outperformed the other three PSO algorithms by a considerable margin. PBPSO yielded better results than SBPSO in a small number of cases (for large MKPs with $m = 5$ constraints when a ring or Von Neumann topology was used for both PBPSO and SBPSO). The problems on which PBPSO outperformed, were likely caused by PBPSO having been better attuned to those specific MKPs than the SBPSO algorithm. In all other cases the SBPSO algorithm was superior, regardless of problem set or topology, to the other three algorithms. Therefore it can be concluded that the first two objectives of this chapter were achieved: SBPSO was successfully applied to the MKP yielding high quality solutions, and SBPSO outperformed the three PSO algorithms it was compared to in a statically significant manner.

A separate finding was that for SBPSO the relative performance of the three swarm topologies differed between the small and the large MKP. On the large MKP, gbest SBPSO performed significantly better than lbest SBPSO and Von Neumann SBPSO: gbest SBPSO achieved the smallest error on *all* of the 243 large test problems. On the small MKPs the difference between gbest SBPSO, lbest SBPSO and Von Neumann SBPSO was marginal and no algorithm-topology pair statistically outperformed or underperformed the other two. Also all three algorithm-topology pairs involving the SBPSO were able

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

to perfectly solve at least 21 out of 40 small test MKPs while they all failed to find the optimum in any of the 100 runs on the same four small test MKPs. Two factors can be identified that possibly caused this difference in relative performance: the first factor is that the problems in the small MKPs were probably not hard enough to allow for meaningful differentiation between gbest SBPSO, lbest SBPSO and Von Neumann SBPSO, while the problems in the large MKP were hard enough to do so. This argument is somewhat undermined, however, by the fact that SBPSO was able to outperform the other PSO algorithms on the small MKP in a statistically significant manner using each of the three topologies. A second possible factor is that, for the large MKP, the tuning set arguably formed a better approximation of the problems in the test set than was the case for the small MKP. For the large MKP, the tuning set consists of the exact same 27 problem definitions as the test set: the tuning set for the large MKPs contained one random implementation for each of the 27 definitions (the same values for MKP parameters *n*, *m*, and *r*) and the test set contained nine random implementations of each of these same 27 definitions. This could mean that the SBPSO algorithm was better attuned in the case of the large MKPs (for each of the three topologies), and in that situation the superiority of the star topology was able to come to light. Weighing these two factors it should be concluded, however, that the results in this chapter did not lead to a definitive explanation for the difference in relative performance between gbest SBPSO, lbest SBPSO, and Von Neumann SBPSO on the small MKPs on the one hand and the large MKPs on the other.

The results from the experiments in this chapter were consistent with the results reported by Wang *et al.* [146] that were described in section 2.4.2.3: in the study by Wang *et al.* [146] the PBPSO algorithm was shown to perform better than both BPSO and MBPSO on 10 small MKPs. The same pattern was seen in the experiments in this chapter, with PBPSO outperforming MBPSO and BPSO on the small MKPs on all three topologies, in all possible measures: best rank, lowest average error, highest success rate, most problems solved perfectly, and least problems where it failed to find a solution at all. Note that the characteristics of the experiments performed by Wang *et al.* [146] were different from those used in this chapter: Wang *et al.* [146] did not perform any tuning of the algorithms, considered on a single swarm topology, used a slightly larger swarm size (30 versus 25 here) and used fewer iterations (3000 or 4000 versus 5000 here).

The solutions for the MKP found by the SBPSO algorithm falls short of the results achieved by state-of-the-art algorithms, which incorporate domain specific information, mentioned in section 2.4.1. The state-of-the-art algorithms for the MKP were all able to find the optimal solution in each run of the algorithm for all 55 small MKP. In contrast, SBPSO combined with the Von Neumann topology was able to find the solution in all independent runs on only 25 out of 40 test problems. On a further 11 test problems, Von Neumann SBPSO found the optimal solution in at least some independent runs. On the remaining four test problems, Von Neumann SBPSO failed to find the optimal solution in all independent runs. For many, but not all, of the large MKP, optimal solutions have been found by state-of-the-art methods and proven to be optimal. Based on the results listed at `http://www.cs.nott.ac.uk/~jqd/mkp/results.html` in July 2015, the optimal solution is known for 196 problems out of the 243 large MKP in the test set. SBPSO combined with the Von Neumann topology was able to find 16 of these optima. For the 47 large MKP in the test set for which the optimum is not yet known, Von Neumann SBPSO was not able to find a better solution than those already known.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.6.2 Tuning and control parameter values

The third objective to be addressed in this chapter was to investigate what parameter values work well for SBPSO. The tuning process compared performance results on a subset of MKPs for many parameter combinations. These parameter combinations were generated using Sobol pseudo-random numbers and spanned the whole parameter space. This allowed for a detailed sensitivity analysis of SBPSO's parameters on the MKP, indicating which values for SBPSO's control parameters led to good results. For the five control parameters, c_1, c_2, c_3, c_4, and k the results of the analysis are summarized in a short list here:

c_1 (attraction to the personal best): For control parameter c_1 the sensitivity analysis very clearly showed that higher values lead to better results with $0.8 \leq c_1 < 1.0$ leading to the best results, regardless of which topology was used. A small drop-off was seen for parameter combinations with $0.9 \leq c_1 < 1.0$ compared to $0.8 \leq c_1 < 0.9$, indicating that the optimal value for c_1 lies above 0.8 but still some distance below the theoretical maximum of 1.0.

c_2 (attraction to the neighborhood best): For control parameter c_2 the area of good results was less clearly marked than for c_1. The best results were achieved by SBPSO using parameter combinations with $0.5 \leq c_2 < 1.0$ for lbest SBPSO and Von Neumann SBPSO, equivalent to half the parameter space for c_2. For gbest SBPSO in contrast, $0.3 \leq c_2 < 0.6$ yielded the best results.

c_3 (the maximum number of elements to add to the solution set using k-tournament selection): For control parameter c_3 lbest SBPSO and Von Neumann SBPSO achieved the best results for values $1.5 \leq c_3 < 2.5$, but all values of c_3 between 1.0 and 3.0 looked adequate. For gbest SBPSO, the area of the parameter space that yielded good results was more evenly spread at slightly higher values with $1.5 \leq c_3 < 3.5$ yielding the best results.

c_4 (the maximum number of elements to remove from the solution set randomly): For control parameter c_4 all values $1.5 \leq c_4 < 5.0$ showed adequate results, regardless of which topology SBPSO was paired with. The best results were found for parameter combinations where c_4 had values in either the range $2.0 \leq c_3 < 2.5$ or $3.5 \leq c_3 < 4.0$. It is not clear why two distinct peaks showed and parameter combinations with $2.5 \leq c_4 < 3.5$ underperformed: this will require further investigation.

k (the size of the tournament used to select elements to add to the solution set): For control parameter k, in general performance increased for higher values of k regardless of which topology was used. For lbest SBPSO and Von Neumann SBPSO parameter combinations with $k = 1$ (which means excluding the tournament selection completely) underperformed, while the performance increase for higher values of k topped off at $k = 6$. For gbest SBPSO a different pattern was seen, with parameter combinations with $k = 1$ not underperforming, but values $k \geq 7$ outperforming.

Besides identifying which parameter values yielded the best results on the MKP, a further analysis was performed to determine which of SBPSO's five control parameters were the most important to tune well. The relative importance of SBPSO's five control parameters was approximately the same for gbest SBPSO, lbest SBPSO and Von Neumann SBPSO: parameters c_1 and c_3 were most important to the overall performance of the SBPSO algorithm. For each of the three topologies used with SBPSO, the

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

parameter c_4 ranked third out of five in importance. The parameters c_2 and k were least influential on the overall performance of the SBPSO algorithm.

The fact that parameter c_2 has a relatively small influence in the performance of the SBPSO algorithm may seem surprising given that the choice of topology was itself important for SBPSO' performance with gbest SBPSO outperforming the lbest SBPSO and Von Neumann SBPSO on all 243 large test problems: the network structure in the swarm was clearly important. However, the SBPSO algorithm was insensitive to the exact amount of attraction to the best neighboring particle in the network. It must be added that looking solely at the ranking of the importance of SBPSO's five control parameters hides the fact that the sensitivity score for c_2 using gbest SBPSO shown in table 5.8 was 31%, higher than that for lbest SBPSO (25%) and almost double that for Von Neumann SBPSO (16%). So, although even for gbest SBPSO the c_1 and c_3 parameters are most important, the attraction to the best particle in the whole swarm indicated by c_2 has some importance when a star topology is used.

The tuning process was successful in finding parameter combinations for each algorithm-topology pair that worked well on the test set, thereby validating the approach of using part of each problem set as a separate tuning set. When tuning SBPSO on the small MKP, however, the SBPSO algorithm performed so well that it actually made the tuning process harder: as evidenced in table 5.5, more than half of the 15 tuning problems were solved in all independent runs of the algorithm. Since this also happened for a number of other parameter combinations, all these combinations achieved the same average rank. Only on the problems that were not solved perfectly could the best parameter combination stand out. This poses a difficulty for using this tuning method if it is hard to identify beforehand problems that will be hard enough to provide discernibility. A similar problem arises in other cases where many runs of the algorithm yield the exact same objective function value, for example a MKP with a local optimum that traps the PSO's particles, such that the global optimum is not found: in such cases the tuning method used also is not able to distinguish between different parameter combinations.

5.6.3 Tournament selection in SBPSO

One could argue that SBPSO had an advantage over the other three PSO algorithms because it contained a special operator to add elements to a particle's position using a k-tournament selection that required additional objective function evaluations. By using a more intelligent way to select elements to add to the position, the search was improved. As the swarm size and the number of iterations were kept the same for all algorithms, the additional objective function evaluations meant that more computational effort was expended on SBPSO compared to the other three algorithms.

The tuning process results and subsequent sensitivity analysis of SBPSO's parameters provided conflicting evidence for the claim that the tournament selection used in SBPSO led to better performance and that larger tournaments were helpful. The sensitivity analysis showed an improvement in performance for SBPSO with increasing values of k, while for lbest SBPSO and Von Neumann SBPSO parameter combinations with $k = 1$ (i.e. without the tournament selection) clearly underperformed. However, two findings from the tuning process provided counterarguments that the influence of the tournament selection may have been small: firstly, the tuning process for SBPSO combined with three topologies on the small MKPs and large MKPs yielded six parameter combinations and thus six values for the control

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

parameter k, the size of the tournament. If the tournament selection provided a large enough advantage to SBPSO, it is to be expected that the chosen parameter combinations had higher than average values for k. If there was no benefit to having tournament selection, the parameter combinations would be chosen solely on the merit of the values for c_1 to c_4. Because of the way the parameter combinations were constructed using Sobol sequences, this would have led to “random” values for k in the parameter combinations that resulted from the tuning process. The six values for k that resulted from the tuning process were 7, 7, 7, 7, 3, and 3. Hence the average of the six values for k was 5.7, larger than but close to the average 5.0 of the possible values of 1 to 9 for k in the tuning process. The value of 5.7 is insufficient to reject a statistical hypothesis that the six values for k were drawn *randomly* from the range of 1 to 9 (p-value of 0.25), and it therefore does not indicate a significantly above average value for k. Secondly, in the analysis of relative importance, control parameter k scored lowest for gbest and lbest SBPSO, while it scored second-lowest for Von Neumann SBPSO. The fact that the value of k was a less important in explaining performance, seems to contradict the idea that larger tournaments provided a benefit to SBPSO. Further analysis is needed to see if the effect of the added objective function evaluations on SBPSO’s performance was significant.

5.6.4 Next steps

The next chapter applies the SBPSO algorithm to a different discrete optimization problem, namely the feature selection problem (FSP) from the domain of machine learning. Again the performance of the SBPSO will be compared to three other PSO algorithms in order to determine its relative merit within the universe of PSO algorithms for DOPs. The FSP not only lies in a completely different domain than the MKP, it also poses challenges to the successful method employed in solving the MKP on tuning the PSO algorithms: a noisy objective function and classifiers that themselves require tuning. The next chapter thus can help determine if the success achieved by SBPSO over the other three PSO algorithms on the MKP is specific to the MKP domain, or whether it can be shown to be more widely applicable.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 6

Experiments on the feature selection problem

The previous chapter described the numerical experiments in which the SBPSO was compared to other PSO algorithms on the MKP. This chapter describes another such set of experiments in which the SBPSO is applied to solve the FSP. The FSP was introduced in chapter 3 as a second test-bed for the new SBPSO algorithm. The current chapter compares the performance of the SBPSO on the FSP to that of three other PSO algorithms from literature.

The objectives of this chapter are two-fold, where the first objective is to determine if it is possible to successfully apply the SBPSO algorithm to the FSP, which in this case means yielding quality solutions. The second objective is to compare the performance of the SBPSO algorithm on the FSP to other PSO algorithms known from literature.

This chapter is organized into an introduction, five main sections, and conclusions. The first three main sections consist of a description of the procedure used in conducting the experiments on the FSP, followed by two exhaustive searches on a number of smaller datasets to investigate if this procedure is viable. The first exhaustive search in section 6.3 tests if the chosen fitness function and classifiers lead to sufficient differentiation across the feature subsets for the PSO algorithm to work. A second exhaustive search described in section 6.4 investigates if the method chosen to tune the parameterized classifiers (J48 and *k*-NN) works well enough. The penultimate main section describes the PSO parameter tuning and resulting PSO parameters, which is followed by the results of the comparisons between the tuned SBPSO and the tuned PSO algorithms from literature.

6.1 Introduction

Similar to the previous chapter, the current chapter can be seen as the culmination of the path outlined in parts I and II. Chapter 1 provided the background to show that a generic, functioning, set-based PSO algorithm did not yet exist and what components such an algorithm should contain. Chapter 4 introduced the SBPSO algorithm with the claim that it is a generic, functioning, set-based PSO algorithm. In order to validate this claim, the SBPSO algorithm needs to be tested on discrete optimization problems (DOPs).

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

This was done for the MKP in chapter 5. Chapter 3 then argued that the FSP is also a non-trivial DOP that forms a valid test-bed for the SBPSO and other discrete PSO algorithms. This chapter brings together the two parts and applies the SBPSO to the FSP. The review of the literature in chapter 3 also suggested a number of other discrete PSO algorithms the SBPSO can be compared to.

As mentioned above, two objectives from the preface are addressed in this chapter: firstly, to test the new algorithm on DOPs, in this case the FSP, and secondly, to compare the performance (in terms of quality of the solution found) of the new algorithm against known discrete PSO algorithms from literature. Recall from the experiments on the MKP that, in chapter 5, a third objective was addressed as well, namely to investigate SBPSO's control parameter values that see which values yield good results. Although a similar parameter tuning as on the MKP will be conducted on the FSP, the sensitivity analysis on SBPSO's parameters from chapter 5 will not be repeated in this chapter.

The objectives from the preface involving the FSP are recalled that are defined to be *outside* the scope of this thesis, and by extension, outside the scope of this chapter:

- find an algorithm that is better at solving the FSP than known state-of-the-art algorithms;
- find the most efficient algorithm in terms of number of iterations or fitness function evaluations, total number of computations (flops) or total time needed to complete; and
- to compare the performance of the new algorithm against non-PSO methods used to solve the FSP.

This chapter is organized into an introduction, five main sections, and conclusions. The purpose and contents of each of the main sections are described in turn below.

Section 6.2 describes the choices made in the setup of the main experiments, in which the tuned SBPSO is compared to three other tuned PSO algorithms on the FSP. First the design choices with regards to the classification problem that underlies the FSP are set out, namely the selection of benchmark datasets for tuning and testing, the way the data in these datasets is pre-processed, which classifiers are used in the experiments, and what their respective setup is. Further details on the background of the datasets used is listed in appendix D. Next, the design choices with regards to the PSO algorithms themselves are discussed. This concerns all the most important considerations that go into the PSO algorithm, and which details allow for reproduction of the experimental results. Therefore, the choices discussed are: the selection of PSO algorithms to compare with the SBPSO, the swarm size, the swarm topology used, the fitness function used in the optimization process, the initialization process for each PSO algorithm, the stopping conditions, and the number of independent runs performed in each experiment. These elements are combined into a wrapper method using a PSO algorithm to solve the FSP.

The fitness function chosen for the PSO algorithms on the FSP is the classification accuracy achieved by the classifier on the dataset. Section 6.3 tests if the classification accuracy for the three chosen classifiers lead to sufficiently different fitness values for different feature subsets. As determining the classification accuracy contains some random elements, repeated calculations of the fitness value for the same feature subset will contain some variation in outcomes. If this variation exceeds the difference between the fitness values of the various subsets, it will be hard to determine whether one PSO has found a better solution than another PSO. The experiments conducted in section 6.3 are thus to investigate if the FSP testbed can differentiate in performance between the PSO algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Besides the fitness function, a second design choice is investigated in more detail for its potential influence on the numerical experiments conducted in this chapter: Section 6.4 looks at the way the parameterized classifiers (J48 and *k*-NN) are tuned, and what impact this particular method of tuning may have on the classification accuracy. If this leads to poorly tuned classifiers, the fitness function used may not perform well enough for the PSO algorithms to effectively search the feature subspace.

Section 6.5 describes the PSO parameter tuning process used for the experiments on the FSP and the resulting PSO parameters. This process mainly follows the one used in chapter 5 to tune the PSO algorithms on the MKP, but with some small differences. Both the process and the resulting parameters are discussed. Detailed results for the PSO tuning experiments can be found in appendix C, section C.2.

Finally, section 6.6 summarizes the results of the main experiments where the SBPSO was applied on the FSP. Detailed results are listed separately in appendix C, section C.3. The results are discussed for each of the three classifiers separately, as well as for all three classifiers combined.

6.2 Experimental procedure

This section describes the procedure that is used in the experiments that compare the SBPSO to other PSO algorithms on the FSP. The overall setup of the experiments in this chapter was strongly influenced by the recommendations by Salzberg [122] on comparing classifiers. The experiments performed and described in this chapter do not directly compare classifiers but instead compare PSO algorithms wrapped around a single classifier, but the same caveats hold. Salzberg [122] recommended the following approach:

1. use a separate data set for parameter tuning than for evaluating classification performance.
2. choose other algorithms to include in the comparison, making sure to include algorithms that are similar to the new algorithm.
3. divide the data set into k subsets for cross validation (a typical experiment uses $k = 10$).
4. calculate overall accuracy averaged across all k partitions, which also gives an estimate of the variance of the algorithms' performance.
5. use correct statistical analysis to compare algorithms, adjusting the confidence level appropriately.

The first three subsections, 6.2.1 through 6.2.3, describe the design choices made regarding the underlying classification problem, namely the selection of benchmark datasets for tuning and testing, the way the datasets are pre-processed, and which classifiers are used in the experiments and what their respective setup is. Subsection 6.2.4 deals with the design choices made with regards to the PSO algorithms used and compared in the experiments. Finally, subsection 6.2.5 combines both these parts and describe the flow of the wrapper method using a PSO algorithm to solve the FSP, given an underlying classification problem.

6.2.1 Benchmark datasets used

This section describes the datasets that are used in the experiments in solving the FSP. First the chosen datasets themselves are listed in two separate sections, one for those used in tuning and one for those

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

used in testing the tuned algorithms. Then the considerations that went into selecting the datasets for the experiments are listed.

6.2.1.1 Datasets selected for PSO testing

Table 6.1 lists the 30 datasets that were selected for use in testing the tuned PSO algorithms on the FSP. Besides the abbreviated name, the number of instances, classes and features are listed, with the latter split between features with numerical and nominal values. For a detailed description of these datasets, see appendix D.

Table 6.1: Datasets used in the FSP testing experiments

dataset	# instances	# classes	# features	# numerical	# nominal
arrhythmia	452	13	279	267	12
audiology	226	24	69	0	69
australian	690	2	14	8	6
bands	540	2	39	20	19
breasttissue	106	6	9	9	0
corral	64	2	6	0	6
crx	690	2	15	6	9
dermatology	366	6	34	1	33
german	1000	2	24	7	13
glass	214	7	9	9	0
hill-valley	1212	2	100	100	0
horse-colic	368	2	36	7	29
ionosphere	351	2	34	34	0
iris	150	3	4	4	0
liver	345	2	6	6	0
monk-1	432	2	6	6	0
monk-2	432	2	6	6	0
movement-libras	360	15	90	90	0
musk-1	476	2	166	166	0
parity5-5	1024	2	10	0	10
parkinsons	195	2	22	22	0
pima	768	2	8	8	0
sonar	208	2	60	60	0
soybean	683	19	35	0	35
spectf	267	2	44	44	0
tic-tac-toe	958	2	9	9	2
vehicle	847	5	18	18	0
vote	435	2	16	0	16
vowel	990	11	10	10	0
wdbc	569	2	30	30	0

6.2.1.2 Datasets selected for PSO tuning

Table 6.2 lists the eight datasets that were selected for use in tuning the various PSO algorithms on the FSP. Besides the abbreviated name, the number of instances, classes and features are listed, with the latter

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

split between features with numerical and nominal values. For a detailed description of these datasets, see appendix D.

Table 6.2: Datasets used in the FSP tuning experiments

dataset	# instances	# classes	# features	# numerical	# nominal
echocardiogram	132	2	12	12	0
hepatitis	155	2	19	19	0
labor	57	2	16	8	8
lung-cancer	32	3	56	0	56
lymphography	148	4	18	3	15
promoter	106	2	57	0	57
wine	178	3	13	13	0
zoo	101	7	16	0	16

6.2.1.3 Considerations in choosing datasets

Datasets for classification and feature selection come in different types: some are real-world datasets, while others are constructed artificially. In shape and size, there is also an enormous variety of specifications for these datasets. The main consideration was to use only datasets that can be considered benchmarks, and which have been used in classification and feature selection studies before. The UCI machine learning repository [5] is meant to provide exactly this and therefore all datasets used in this thesis come from this repository.

The next step was to determine which of the 320 datasets from the repository should be used. A first selection was to only consider the 231 datasets that can be used to study classification. Other than this simple step, selection was mainly driven by practical considerations, namely the time it would take to run the experiments. Since the goal of this chapter is to compare PSO algorithms on the FSP, and not to study classification and feature selection algorithms for very large datasets, limiting the size of the datasets makes sense from the viewpoint of keeping the computational effort manageable. This led to choosing datasets with a limited number of features and instances. The number of features was limited to roughly 100, although two datasets (arrhythmia with 279 and musk-1 with 166 features) were added with a higher number of features. Note that most datasets in the repository have less than 100 features, although large datasets with up to 3 million features are also present. The number of instances was limited to roughly 1000, which can be considered up to medium size. Note that half the datasets available in the repository have *more* than 1000 instances, and datasets can contain up to 11 million instances. From the set of datasets that were small enough, a number were selected using a mix of real-world and artificial datasets. Also, the goal was to include as many datasets as possible within the practical bounds of computation time, so smaller datasets (fewer features and instances) were included more often.

Considerations also went into selecting which datasets to use for tuning the PSO algorithms and which to use for testing and comparing the tuned algorithms. For the testing phase, the aim is to use a larger number of problems such that the comparison of the PSO algorithms can yield statistically significant results. For tuning, a smaller number of problems can be used, although some care should be taken to prevent overfitting to a set of problems that are not representative of the testing problems. In

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

tuning, however, it is important to make the problems challenging enough that different combinations of PSO parameters lead to a difference in performance: if the tuning problems are too easy, many different parameter combinations will yield equivalent results. Too easy in this case means too few features. Hence the tuning were chosen to contain at least 12 features. This meant that the space of features subsets contained at least $2^{12} = 4096$ points, making the optimization non-trivial given the number of particles and iterations to be used as described in section 6.2.4.

Using all these considerations, eight datasets were selected for use in tuning the PSO algorithms, listed in table 6.2. For testing the tuned PSO algorithms, a further 30 datasets were selected, listed in table 6.1.

6.2.2 Preprocessing of datasets

The reason for pre-processing datasets in machine learning and the methods available to do so were outlined in section 3.2.4. This section describes the actual pre-processing performed on the datasets described in the previous section *before* these datasets were used in the classification experiments discussed in this chapter. Data cleaning, transformation and normalization are all applied to make the datasets amenable to classification using the selected classifiers.

6.2.2.1 Data cleaning

Data cleaning was performed in this thesis only to fill in missing values. Any values that were present in the datasets were assumed to be correct, and no outlier detection or instance selection was performed to discard or transform suspect data. Missing values were filled in using two different methods, depending on the type of data that is missing:

- For attributes that contain numeric values, any instance for which this attribute's value was unknown, the attribute's value was set to the mean of all known values for this attribute. This method is known as *mean substitution*. Note that no distinction was made based on the class of the instance with the missing value.
- For attributes that contain nominal values, any instance for which this attribute's value was unknown, the attribute's value was set to the most frequently occurring value for this attribute in the dataset. This method is known as the *most common feature value method*. Note that no distinction was made based on the class of the instance with the missing value.

No statistical analysis was performed to look for outliers or inliers among the known and legal values in the datasets: all legal values were taken at face value. This was done because the datasets in question are well-studied benchmark datasets and as such other studies will have used the same outliers and inliers without correction.

6.2.2.2 Data transformation

Data transformation was used to convert data into a single format that can be handled by all classifiers used in these experiments. Note that this is not a prerequisite for comparing different PSO algorithms

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

using the same classifier, because a different format of the dataset can be used with different classifiers without jeopardizing this goal. In order to allow some global comparison across classifiers, the data transformation step was still undertaken. During the data transformation, all data was converted into numerical values. The main reason for this choice is that this thesis combines the *k*-nearest neighbor classifier with an Euclidean metric, which only works with numerical values.

Attributes with nominal values were converted using the following method: first, a list is determined of all different attribute values in the order found in the dataset, starting at the first instance and working down until the last instance. Then, the nominal value is converted to a number equal to the order in which the nominal value appeared in the aforementioned list. If, for example, the attribute values in the dataset in order of appearance were A, C, C, A, B, C, then the nominal values A, B, and C were converted to the numerical values 1, 3, and 2 respectively.

6.2.2.3 Data normalization

Data normalization was applied as a last step. Due to the application of the previous two steps of data cleaning and data transformation, at this point no attribute values were missing and all attribute values were numerical. Normalization was then performed to scale each attribute separately to lie within the range $[0,1]$ using *min-max normalization*.

The end result of the three data preprocessing steps was 38 datasets without missing or unknown values, in which each instance was represented as a vector of attributes scaled to lie in the range of $[0,1]$, and a class label indicated by an integer number. These same representations were used for all three classifiers and all PSO algorithms.

6.2.3 Classifiers

An important role in solving the FSP using a wrapper method is reserved for the classifier. The classifier is used to measure the quality of the classification using a selected subset of features, and the PSO uses this measurement in steering the search towards, hopefully, better feature subsets. This section describes the classifiers used in the experiments on the FSP. It consists of three parts: firstly, which classifiers were chosen to be used in the experiments in this chapter, secondly, which implementations of these classifiers, what parameters, metrics and settings were used, and thirdly, how the parameterized classifiers were tuned.

6.2.3.1 Choice of classifiers

The goal in choosing a classifier to help solve the FSP was not to achieve a higher accuracy than previously reported in literature. Instead, only the *relative* ranking of various subset of features is important in the PSO algorithms to steer the search. As such, any classifier that benefits from feature selection can be used in this task. As a separate consideration, classifiers that have no or few parameters are to be preferred. This is because these classifier parameters will need to be tuned in order to achieve a good classification performance.

A total of three classifiers were used in the experiments so that the PSO algorithms could be compared

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

to each other in multiple settings. This was done because classifiers can potentially introduce a bias into the comparison of the PSO algorithms. When considering which classifiers to use in the experiments, the choice was made to use those classifiers that have a track-record in literature of use in feature selection with wrapper methods, which are easy to use, and which are relatively quick to train and evaluate on the datasets used.

The three classifiers chosen to be used in the experiments on the FSP in this thesis were the Gaussian naive Bayes (GNB), a specific implementation the C4.5 decision tree classifier called J48, and the k-NN classifier. The implementation details of these three classifiers are given in the following section.

6.2.3.2 Implementation of classifiers

This section lists the choices made in implementing and using the chosen classifiers in the numerical experiments. Such changes are not always listed in literature, but are required in order to allow for replication of the experimental results.

6.2.3.2.1 GNB classifier: The implementation of the GNB used was that from Weka 3: Data Mining Software in Java [48]. Some small alterations were made to the Weka code to allow control of the pseudo-random number sequence used. This was done to ensure replicability of the numerical experiments.

Laplace smoothing was *not* applied in calculating the class prior probabilities (see section 3.2.3.2). The GNB classifier does not have any further settings or parameters that had to be tuned.

6.2.3.2.2 J48 classifier: The experiments in this thesis that involved a decision tree classifier all used the implementation of the C4.5 algorithm called J48 which is part of Weka 3: Data Mining Software in Java [48]. The J48 implementation of the C4.5 decision tree has a number of options and variables that can be customized. Two of these variables are kept as parameters for the classifier:

- The minimum number of instances required for a leaf node in the tree, l. This parameter must be at least 1, with a default value of 2. This parameter can be used to obtain smaller trees and thus simpler models by explicitly setting the minimal number of instances in a single leaf. A higher number for l thus restricts the size of the decision tree.
- The confidence threshold used in the pruning process, γ. This parameter ranges between 0 and 1, with a default value of 0.25. The parameter determines a threshold of the allowed inherent error in the data when pruning the decision tree. By lowering the threshold, more pruning is applied and consequently a smaller tree and a more general model is generated.

Other options available in the J48 implementation were kept constant at the default values in Weka 3:

- The option for binary splits was set to the default value `false`;
- The option `setUnpruned` was set to `false` such that pruning of the tree *is* performed;
- The number of folds for reduced error pruning was set to the default value 3;
- The option for reduced error pruning was set to the default value `false`;

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- The option for Laplace smoothing for predicted probabilities was set to the default value `false`;
- The option for subtree raising (a technique where a node may be moved upwards towards the root of the tree, replacing other nodes along the way during a process of pruning) was set to the default value `true`.

6.2.3.2.3 *k*-NN classifier: The implementation of the *k*-nearest neighbor classifier was written in Java specifically for these experiments. The Euclidean distance function was used as the metric to determine which instances are nearest. Ties were broken randomly using a controlled pseudo-random number sequence to ensure replicability.

6.2.3.3 Tuning of classifiers

The implementations of the J48 and *k*-NN classifiers used in this thesis are both parameterized and thus values needed to be chosen for these parameters before the classifier can be used. In order for the classifiers to work well, a tuning process was performed to select parameters that allow for good classification accuracy. The GNB classifier has no parameters and thus had no need of tuning. In the experiments described in this chapter, the J48 and *k*-NN classifiers were tuned at various moments in the process of solving the FSP, but always in the same manner. This section describes the tuning process used.

Tuning was done by choosing the parameter value (for the *k*-NN classifier) or values for the parameter combination (for the J48 classifier) that maximize the classification accuracy on the dataset under consideration. This classification was performed using *all* features in the dataset. The most impactful moment where classifier tuning was used, was at the start of the PSO wrapper algorithm. At that time the classifier was tuned and this tuned classifier was subsequently used for *all* classifications during the PSO run, regardless of how many features were selected at that point of the search. In section 6.4 an investigation is made of the impact of the choice to tune the classifiers on a dataset using all features and then using the tuned classifier also on smaller features subset in the PSO run.

The classification accuracy in the classifier tuning process was determined as the average over 10 independent runs of a 10-fold cross validation accuracy calculation. By repeating the cross validation 10 times, extra computational effort was expended to ensure an as good as possible tuning, meaning that the classification accuracies for each of the parameters contained little numerical noise.

The sets of parameter values that were used in tuning the J48 and *k*-NN classifiers respectively are given below.

6.2.3.3.1 J48 classifier: To tune the J48 classifier, both its parameters needed to be optimized together: l, the minimum number of instances required for a leaf node in the tree, and γ, the confidence level. This simultaneous tuning was done by choosing the combinations of the two parameters that maximized the classification accuracy measured on a given dataset. For each parameter seven different equally spaced choices were considered in the tuning, for a total of 49 different combinations. The values for l ranged from 2 to 50, the values for γ ranged from 0.050 to 0.500. The exact values used in the tuning are given in table 6.3.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.3: Parameter values used to tune the J48 classifier

l	2	10	18	26	34	42	50
γ	0.050	0.125	0.200	0.275	0.350	0.425	0.500

6.2.3.3.2 *k*-NN classifier: To tune the *k*-NN classifier, only the value for k, the number of nearest neighbors to include in the vote to determine an to-be-classified instance's class label, needed to be optimized. In the tuning process, 10 different values of k were investigated, ranging from 1 to 19. Because ties in the nearest neighbors are broken randomly by the *k*-NN classifier, it is preferable to prevent a large number of such ties. Therefore, only odd values were considered in the tuning of k. The exact values used are given in table 6.4.

Table 6.4: Parameter values used to tune the *k*-NN classifier

k	1	3	5	7	9	11	13	15	17	19

6.2.4 Setup of PSO in solving the FSP

This section describes the experimental setup of the PSO algorithms that were used to solve the FSP. The following design choices for the PSO algorithms are touched upon in separate sub-sections: the selection of PSO algorithms to compare with the SBPSO, the swarm size, the swarm topology, the fitness function used in the optimization process, the initialization procedure for each PSO algorithm, the stopping conditions, and the number of independent runs performed in each experiment.

6.2.4.1 Choice of PSO algorithms

The proposed SBPSO algorithm is compared to three other PSO algorithms: BPSO by Kennedy and Eberhart [62], CFBPSO by Chuang *et al.* [21], and PBPSO by Zhen *et al.* [157]. Refer to sections 1.3.1.1, 1.3.1.4, and 1.3.1.3 for detailed descriptions of these algorithms.

To decide which PSO algorithms to compare the SBPSO to on the FSP, a logical first thought would be to select the same three PSO algorithms that were used in the comparison on the MKP, as described in section 5.2.2: the BPSO, MBPSO, and PBPSO. Two of these were indeed used in the comparison again, but the MBPSO algorithm was replaced in the experiments on the FSP by the CFBPSO algorithm. The MBPSO algorithm was left out because the results from the experiments on the MKP in chapter 5 showed it performed quite poorly. The CFBPSO was used as a replacement, as the work by Chuang *et al.* [21] showed it can be used in a wrapper approach in solving the FSP and performed well when compared to the BPSO in that study.

For BPSO, CFBPSO, and PBPSO the candidate solution was represented by binary-valued particle positions: the bit values b_i were interpreted as whether the i-th feature in an FSP was used in the classification or not. The SBPSO algorithm treated the features in the FSP as elements to be in- or excluded from a position.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

6.2.4.2 Swarm size

Section 5.2.3 in the chapter on experiments on the MKP already indicated the impact that the number of particles in a swarm can have on the search performance. It should be noted, however, that using the optimal number of swarm particles is not too important because the objective is not to find the best possible solution to the FSP, but to compare the relative performance of the different algorithms. Hence using the same number of particles in each case that is important, but this swarm size need only to be adequate for the problems studied.

Because the search space for the FSP in this chapter is smaller than that for the MKP studied in the previous chapter, it seems logical to use a smaller number of particles in the swarm than the 25 that were used on the MKP. Therefore in the experiments on the FSP the swarm size was set to 20 particles in all cases. This is also in line with the majority of studies that used PSO on the FSP reviewed in section 3.4.4.

6.2.4.3 Swarm topology

In the experiments on the MKP in chapter 5, three different topologies were used to investigate the SBPSO and the relative performance of SBPSO versus other PSO algorithms, namely the star, ring and Von Neumann topologies. This was done to prevent a bias that the choice of topology might introduce in the relative performance of the PSO algorithms. The results on the MKP showed that the choice of topology indeed has an impact on the performance of the PSO algorithms.

In the experiments on the FSP a similar approach to prevent bias was taken, but in this case the swarm topology was not the only choice to be made. The choice of which *classifier* to use can also bias the relative performance of different algorithms on the FSP, see for example the work by Yun *et al.* [155] discussed in section 3.4.4.2. The choice was made in this thesis to prevent a bias by using three different classifiers for the experiments in this chapter. In order to prevent the number of experiments to be conducted from becoming very large, the choice was made to use only one of the swarm topologies for all experiments on the FSP. The question then becomes *which* topology should be chosen.

Almost all papers reviewed in section 3.4.4 that apply a form of PSO on the FSP used a star topology. The review of literature does not, however, indicate that this is the result of a conscious choice in which other topologies were also considered. This is worrying, because the work by Kennedy and Mendes [64] shows that the choice of topology can significantly impact the performance of a PSO algorithm. No study could be found that investigated the specific issue of which topology works well for PSO in the domain of feature selection and classification. Instead of following the previous studies which applied PSO to the FSP in using a star topology, a Von Neumann topology was used for all different PSO algorithms and in all experiments on the FSP. This is in line with the suggestion by Kennedy and Mendes [64]. The Von Neumann topology offers a medium level of connectedness for the swarm: the ring topology is less connected and the star topology has direct connections between all different particles.

6.2.4.4 Fitness function

Swarm intelligence methods like PSO use a fitness function to steer the stochastic search. PSO steers each particle through attraction to the particle’s best position from history and the best known position in

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the particle's neighborhood, where best means the best fitness value. The fitness function must be linked to the goal of the problem to be optimized. The goal in the FSP is to find the best subset of features for classification by a classifier, so the PSO's fitness function needs to contain a performance measure for that classification. The task was to determine which performance metric to use and what cross validation method.

Xue *et al.* [150] argued that the inclusion of the proportion of features left out of the feature subset can be included in the fitness function to improve performance. The results of that study are, however, limited and not proven to be statistically significant. In order to avoid extra design choices to determine the relative importance of classification performance and the number of features, the fitness function used contained only a performance measure for classification.

6.2.4.4.1 Performance metric: Sokolova *et al.* [130] discuss different performance measures used in classification and indicate that the empirical evaluation of algorithms and classifiers is a matter of on-going debate. Classification accuracy using cross validation is considered an acceptable performance metric in general. Specific situations such as the medical domain could call for other measures, i.e. sensitivity or ROC analysis, that put more emphasis on failure avoidance or class discrimination.

In this thesis classification accuracy was used as the performance metric. One reason for this was that some of the benchmark datasets used (see sections 6.2.1.1 and 6.2.1.2) contained problems that have more than two classes. This implies that other frequently used performance measures such as precision, sensitivity, and specificity are not universally applicable, since these can only be calculated for datasets with two classes.

6.2.4.4.2 Cross validation method: The cross validation method used in the experiments presented in this chapter is *k*-fold cross validation, with *k* set to 10 in most cases. This is in line with the recommendations by Salzberg [122]. A side benefit of the repeated calculations is that the standard deviation across multiple calculations can also be determined. This gives insight into the stability of the classification accuracy across the *k* folds. The preference is for a low standard deviation, which indicates stability.

Using *k*-fold cross validation is computationally intensive, however, and needs to be performed for each candidate solution during each iteration of the PSO algorithm. Therefore, the choice is made to keep the number of repetitions of the *k*-fold cross validation small during the main loop of the algorithm. This is a trade-off between stability of the fitness value calculated for a particle position, and the computational effort required. Hence the number of repetitions of the *k*-fold cross validation is set to:

- 10 in determining the best parameters for the classifier (in case of the J48 or *k*-NN classifier);
- 5 in the fitness function used during the main loop of the PSO algorithm; and
- 10 in determining the final classification accuracy of the found feature subset.

The various phases of the wrapper method mentioned here will be explained more fully in section 6.2.5.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

6.2.4.4.3 Fitness function algorithm: The fitness function $f(.)$ combines simple classification accuracy as the performance metric with k-fold cross validation and is described in algorithm 8.

Algorithm 8: Fitness function for use in experiments on FSP

```
Consider an FSP with underlying dataset $D$ and classifier $C$;
Let $X = \{x_1, \ldots, x_m\}$ be a candidate solution of selected attributes determined by the PSO;
Randomly divide $D$ into $k$ disjunct folds $D_i$;
for $i = 1, \ldots, k$ do
    Set training set $L_i = D_i$ and test sets $T_i = D \backslash L_i$;
    Train classifier $C$ on training set $L_i$ using only the attributes in $X$;
    Determine accuracy$(T_i, C, X)$, the accuracy of $C$ on test set $T_i$ using only attributes in $X$;
end
The fitness $f(X)$ of $X$ is equal to the average over all $k$ measures of accuracy$(T_i, C, X)$.
```

6.2.4.5 Initialization

Particles were initialized randomly for each PSO algorithm. For the BPSO, CFBPSO, and PBPSO algorithms, the positions were initialized randomly in $\{0,1\}^n$, while the velocities for each were initialized randomly in $[-1,1]^n$, following [31]. For PBPSO the continuous-valued positions, $\vec{x'}_i(0)$, were initialized as $\vec{0}$, to ensure that no initial bias was included in the discrete-valued positions, $\vec{x}_i(0)$. For the SBPSO algorithm, the positions were randomly initialized, such that each element had a 0.5 chance of being included, and all velocities were initialized as the empty set.

6.2.4.6 Stopping conditions

Due to computational considerations, much fewer fitness function evaluations could be used in the experiments on the FSP than used for MKP in chapter 5. This is because, while the fitness function evaluation of a single candidate solution for the MKP takes very little time (checking the constraints and, if all are met, summing the value of the included weights), for the FSP this involves five cycles of training a classifier and determining the classification accuracy. Both a large number of features or a large number of instances in the training and testing sets can make this a slow process.

For each independent run of an algorithm, the same three stopping conditions were applied:

1. the best fitness function value found in the swarm equaled a classification accuracy of 100%,
2. the best fitness function value found in the swarm had not improved for 50 iterations, or
3. more than 100 iterations had passed.

6.2.4.7 Number of independent runs

PSO is a stochastic optimization algorithm, and thus individual runs of the algorithm can have different results. Hence, multiple independent runs of the algorithms have to be executed and the average performance reported. For all experiments on the FSP, both in tuning and in testing, 30 independent runs were used.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

6.2.5 PSO wrapper algorithm for FSP

All the components from the previous parts of this section combined do not yet fully describe the experiments run in order to test and compare the PSO algorithms on the FSP. The final part is the flow of the PSO wrapper method, which is described in this section.

The main construction of the PSO wrapper is very basic and is described in algorithm 9. It returns the feature subset found by the PSO search and the average classification accuracy using that feature subset. The experiments started with the initialization of the classifier and particle swarm, followed by the main PSO wrapper loop, and a final accuracy calculation. These three steps are described in separate algorithms below. After these descriptions, some observations are made about the use of pseudo-random number sequences in the numerical experiments on the FSP.

Algorithm 9: PSO wrapper approach to solve FSP: overview

```
Initialize classifier and swarm according to algorithm 10;
Run main loop of PSO algorithm according to algorithm 11;
Determine final accuracy and according to algorithm 12;
Return values:
    · best position found $\widehat{Y}$;
    · $f_{\text{final}}(\widehat{Y})$;
    · standard deviation of 10 independent final classification accuracy calculations;
```

6.2.5.1 Initialization of PSO wrapper

Algorithm 10 describes in detail the initialization at the start of the PSO wrapper method.

Algorithm 10: PSO wrapper approach to solve FSP: initialization

```
Define classifier $C'$ as classifier $C$ with best parameters:
    · from parameter grid $G$;
    · “best“ defined as highest classification accuracy;
        · using all $n$ features;
        · by 10-fold cross validation;
        · accuracy calculation repeated 10 times;

Set fitness calculator $f_{C'}$ to:
    · average of 5 independent calculations;
    · of average classification accuracy by classifier $C'$;
    · using 10-fold cross validation;

Set $N$ equal to the number of particles in the swarm;
for $i = 1, \ldots, N$ do
    Initialize $V_i$ according to algorithm’s velocity initialisation strategy;
    Initialize $X_i :=$ random subset of $U$;
    calculate $f_{C'}(X_i)$;
    Initialize $f_{C'}(Y_i) := -\infty$;
end
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

This description assumes that a parametrized classifier (J48 or k-NN) is used and the PSO search is performed by the SBPSO. If the GNB classifier is used instead, the steps to tune the classifier C resulting in C' are skipped, and the classifier C' is just equal to the original GNB classifier C. If another PSO algorithm is used, the initialization of the particle swarm is not a random subset of U, but instead (though equivalently) a random vector in $\{0,1\}^n$, where n is the number of features in the dataset.

6.2.5.2 Main loop of PSO wrapper

After initialization, the main loop of the PSO wrapper is executed. This loop is described in algorithm 11. The only link to the FSP was via the fitness function f, for which each evaluation meant determining the average classification accuracy using 5-fold cross validation. The main loop returned $\widehat{Y}$, the feature subset with the highest registered fitness.

Algorithm 11: PSO wrapper approach to solve FSP: main loop

```
while all stopping conditions are false do
    for $i = 1,\dots,N$ do
        if $f_{C'}(X_i) > f_{C'}(Y_i)$ then
            $Y_i := X_i$;
        end
        if $f_{C'}(X_i) > f_{C'}(\widehat{Y}_i)$ then
            $\widehat{Y}_i := X_i$;
        end
    end
    for $i = 1,\dots,N$ do
        Update $V_i$ according to PSO's velocity update equation;
        Update $X_i$ according to PSO's position update equation;
        Calculate fitness $f_{C'}(X_i)$ for particle $i$;
    end
end
Initialize $f_{C'}(\widehat{Y}) := -\infty$;
for $i = 1,\dots,N$ do
    if $f_{C'}(\widehat{Y}_i) > f_{C'}(\widehat{Y})$ then
        $\widehat{Y} := \widehat{Y}_i$;
    end
end
Return $\widehat{Y}$;
```

6.2.5.3 Final classification accuracy calculation

As described in the previous sections, during the main loop of the PSO wrapper fitness function evaluations were made using a classifier that was tuned (in case of the J48 and k-NN classifiers) using all features. The classification accuracy during the main run was determined using 5-fold cross validation. Both choices meant that just reporting the classification accuracy recorded during the PSO search is not the best representative depiction of accuracy resulting from the found feature subset. Therefore, after

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the main loop had completed, the classifier C was again tuned according to the process described in section 6.2.3.3, but now using *only the features selected* in $\widehat{Y}$. This tuning resulted in the classifier C_{final}. In case of the GNB classifier, C_{final} equaled the original C classifier. Using this re-tuned classifier, a final fitness function evaluation was made as the average of 10 repeated accuracy calculations using 10-fold cross validation.

The final recorded outcome of a single run of the PSO wrapper method consisted of the best position found, $\widehat{Y}$, the final fitness function evaluation, $f_{\text{final}}(\widehat{Y})$, and the standard deviation across the 10 repeated accuracy calculations using 10-fold cross validation. Algorithm 12 describes this final calculation process in detail.

Algorithm 12: PSO wrapper approach to solve FSP: final classification

```
Define classifier $C_{\text{final}}$ as classifier $C$ with best parameters:
    · from parameter grid $G$;
    · "best“ defined as highest classification accuracy:
        · using only the features in $\widehat{Y}$;
        · by 10-fold cross validation;
        · accuracy calculation repeated 10 times;

Set final fitness calculator $f_{\text{final}}$ to:
    · average of 10 independent calculations;
    · of average classification accuracy by $C_{\text{final}}$;
    · using 10-fold cross validation;

Re-calculate fitness of $\widehat{Y}$ as $f_{\text{final}}(\widehat{Y})$;
Determine standard deviation across 10 independent final classification accuracy calculations;

Return values:
    · best position found $\widehat{Y}$;
    · $f_{\text{final}}(\widehat{Y})$;
    · standard deviation of 10 independent final classification accuracy calculations;
```

6.2.5.4 Use of pseudo-random numbers in cross validation and classification

Special care was taken that the pseudo-random numbers used in the cross validation splits of the dataset for the final classification accuracy always started at the same point. In this setup, the splits in the dataset still varied between the 10 independent calculations that underly $f_{\text{final}}(\widehat{Y})$ and these 10 calculations can still be considered to be independent. This ensured the most fair comparison and full replicability.

This same care was used for the pseudo-random numbers used by the classifiers themselves (for example, to break ties when using the k-Nearest Neighbor classifier). The final classification accuracy calculation always started the random sequence at the same point and this meant that, if the exact same features were selected and the same classifier was used for two independent runs of the PSO, the exact same average classification accuracy and standard deviation would result. Again, the 10 calculations that underly the determination of $f_{\text{final}}(\widehat{Y})$ were still independent of each other, as they all used a different part of the same pseudo random-number sequence.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

It is important to understand that *only* if the features selected are exactly the same for two independent runs of a PSO algorithm (or for that matter between two different PSO algorithms) using the same classifier, the resulting average classification accuracy was the same. If one feature differed, then the series of pseudo-random numbers was used in a slightly different order, resulting in a different classification accuracy.

Also it is important to stress that this special way of calculating the final accuracy did not affect the actual PSO algorithm nor the features selected - it only affected the final accuracy recorded and used to compare the PSO algorithms. The special calculation affected the comparison in such a way that, if the exact same features were selected, the same accuracy resulted: hence the only way PSO algorithms distinguished themselves in accuracy, was by the selected features and as little as possible by the randomness inherent in the classification accuracy calculation.

Finally, it is noted that the Mersenne Twister algorithm [95] was used to generate the pseudo-random number sequences in the experiments in this chapter. Only for selecting which PSO parameter combinations to consider in the PSO tuning process, was a different pseudo-random sequence used. In that case, the parameter combinations spanning the PSO parameter space were generated using a Sobol pseudo-random number sequence as described in section 6.5.

6.3 Exhaustive search to test the fitness function

This section describes the exhaustive search performed to determine if the setup of a PSO wrapper approach using the three selected classifiers works sufficiently well to compare PSO algorithms. Particularly, whether the fitness measurement by the cross validated classification accuracy on different feature subsets allows for statistically significant differences in performance by the PSO algorithms.

The task that the PSO algorithms have in solving the FSP is to find an optimal subset of features that allows for the most accurate classification by a classifier, as described in section 6.2.5. However, due to the stochastic element present in the classifiers (for example to break ties), the classification accuracy using a particular feature subset is “noisy”: repeated measurements will tend to show slight differences in classification accuracy. If the difference in classification accuracy between different feature subsets is small compared to the uncertainty in the classification accuracy measurement for a single feature subset, there is no statistically significant difference in how optimal the different subsets are and the FSP may not help to differentiate between the performance of different PSO algorithms.

To test whether the setup of fitness function works sufficiently well, an exhaustive search was performed of the fitness values across all feature subsets of a number of dataset-classifier pairs. The next section describes how this search was set up, followed by the results and conclusions.

6.3.1 Experimental method

The description of the experimental method for the exhaustive search to test the fitness function is divided into three parts: which datasets were selected, how the exhaustive search was organized, and how the classifiers were set up and tuned.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

6.3.1.1 Dataset selection

In an exhaustive search on a dataset containing n features, 2^n different sets of selected features need to be tested. For large numbers of n this becomes prohibitively expensive in terms of the number of computations. This is exactly the reason that feature selection is used on large datasets. For the investigation described in this section, the number of features was limited to 10, meaning at most $2^{10} = 1024$ different subsets needed to be tested for a dataset. The 11 datasets thus selected are listed below in table 6.5 in section 6.3.1.3. Note that in the main FSP experiments all these 11 datasets were used during the *testing* phase and none were used during the PSO parameter tuning.

6.3.1.2 Organization of exhaustive search

For the 11 datasets and the three classifiers used in the FSP experiments, the fitness value (average classification accuracy measured using 10-fold cross validation) was determined for all possible feature subsets. The fitness value was calculated independently 10 times and the average accuracy and the standard deviation of these 10 calculations were recorded.

A fair comparison of the accuracy between different subsets of selected features requires that all other circumstances are as equal as possible. Two such important circumstances are the splits of the dataset during the 10-fold cross validation and the parameters used by the classifiers themselves. In the exhaustive search, a separate pseudo-random numbers sequence was used in the 10-fold cross validation to determine the random splits of the dataset into the training set and the testing set. This meant that for each of the 10 independent calculations of the classification accuracy for a given subset, a different 10-fold split of the dataset was used. But the same 10 different 10-fold splits were used for the other feature subsets in that dataset.

6.3.1.3 Classifier tuning

The J48 and k-NN classifiers require parameters to be chosen for the classifier to work: l and γ for the J48 classifier and k for the k-NN classifier. For both classifiers these parameters were set beforehand and the same parameters were used on all subsets of features tested in the exhaustive search. The tuning method used was the same as described in section 6.2.3.3. For the GNB classifier no parameters had to be chosen. The resulting classifier parameters per dataset are listed in table 6.5.

The fact that the classifier parameters were chosen from a grid as the best performing parameter combination using all features in classification, may have introduced a bias that favored large subsets. This potential problem is investigated in more detail in section 6.4.

6.3.2 Results

This section describes the results of the exhaustive search of the fitness values across all possible feature subsets. For each pair of dataset and classifier, the classification accuracy was determined exhaustively for all feature subsets. This resulted in an average accuracy and the standard deviation of the 10 independent calculations of that accuracy. A summary of these results is given in table 6.6.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.5: Classifier parameters and metrics for the J48 and *k*-NN classifiers used in the exhaustive search on the fitness function.

		J48		*k*-NN	
Dataset	# features	l	γ	k	metric
iris	4	18	0.275	5	Euclidean
corral	6	2	0.050	1	Euclidean
liver	6	26	0.275	3	Euclidean
monk-1	6	34	0.275	1	Euclidean
monk-2	6	2	0.425	5	Euclidean
pima	8	2	0.425	19	Euclidean
breasttissue	9	2	0.125	1	Euclidean
glass	9	2	0.275	1	Euclidean
tic-tac-toe	9	18	0.125	9	Euclidean
parity5-5	10	2	0.350	9	Euclidean
vowel	10	2	0.350	9	Euclidean

It was then determined how many of the feature subsets yielded an accuracy of within 1.645 standard deviations (the 95% confidence level assuming a normal distribution) of the best found accuracy. Also included in the column “% best” is the percentage of all feature subsets in datasets that were in this way indistinguishable from the one with the highest accuracy found.

Table 6.6: Number of subsets in exhaustive search that are within 95% confidence interval from best recorded accuracy.

			GNB		J48		*k*-NN	
Dataset	# features	# subsets	# best	% best	# best	% best	# best	% best
iris	4	16	**1**	6.3 %	8	50.0 %	4	25.0 %
corral	6	64	10	15.6 %	2	3.1 %	**1**	1.6 %
liver	6	64	25	39.1 %	9	14.1 %	4	6.3 %
monk-1	6	64	10	15.6 %	32	50.0 %	5	7.8 %
monk-2	6	64	64	100.0 %	**1**	1.6 %	**1**	1.6 %
pima	8	256	2	0.8 %	56	21.9 %	**1**	0.4 %
breasttissue	9	512	9	1.8 %	22	4.3 %	75	14.6 %
glass	9	512	28	5.5 %	28	5.5 %	3	0.6 %
tic-tac-toe	9	512	**1**	0.2 %	**1**	0.2 %	**1**	0.2 %
parity5-5	10	1024	11	1.1 %	15	1.5 %	6	0.6 %
vowel	10	1024	**1**	0.1 %	78	7.6 %	9	0.9 %

The ideal situation is where one feature subset yields a classification accuracy that is statistically superior to that of all other feature subsets. Table 6.6 shows that this was an uncommon situation that occurred for only nine of the 33 datasets:

- For GNB, only three (iris, tic-tac-toe, and vowel) out of the 11 datasets had a single feature subset for which the classification accuracy was better in a statistically significant manner than all other feature subsets.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- For J48, only two (monk-2 and tic-tac-toe) out of the 11 datasets had a single feature subset for which the classification accuracy was better in a statistically significant manner than all other feature subsets.
- For *k*-NN, only four (corral, monk-2, pima, and tic-tac-toe) out of the 11 datasets had a single feature subset for which the classification accuracy was better in a statistically significant manner than all other feature subsets.

Also it is clear that for some combination of classifier and dataset, feature selection itself had no benefit: using the GNB classifier on the monk-2 dataset failed completely: all 64 feature subsets had the exact same classification accuracy which was the same as that of randomly selecting an instance form the training set and applying that class label to any new instance to be classified. Only the fact that the class distribution for monk-2 is uneven, led to an accuracy above 50%, because the GNB classifier was unable to capture the concept underlying this artificial dataset. Using a wrapper method with the GNB classifier, the FSP is not a well-posed problem and it is impossible to differentiate in the quality of different PSO algorithms trying to solve it.

Other combinations of dataset and problem also led to the FSP being a poorly posed problem, though not as bad as the monk-2 and GNB classifier case: for the liver and GNB, iris and J48, and monk-1 and J48 each, almost 40% or more of the feature subsets led to statistically the same accuracy as the optimum. It is difficult, however, to say at what percentage of equivalent feature subsets the FSP should be considered inadequate for use in the PSO comparison. Other combinations like iris and *k*-NN or pima and J48 were also at least borderline in their suitability.

Another thing that stood out, was that there was no clear pattern across dataset or classifier: if a combination of one classifier and one dataset performed badly in the manner described above, that same dataset may have combined with another classifier to form a perfect FSP. The monk-2 dataset failed completely with the GNB classifier, but combined with the J48 classifier, the FSP became a problem with only one, statistically discernible optimum. Also, the three classifiers were affected by this problem in roughly equal measure based on this limited investigation of eleven datasets, although one could claim that the *k*-NN classifier was least affected.

The next three subsections give the detailed results of the investigation per classifier. In each case a table lists the 11 datasets investigated, the number of features of the dataset, the number of different combinations of feature subsets investigated (# combo’s), and the average accuracy and the standard deviation over 10 independent accuracy calculations for two specific feature subsets: all features and the feature subset yielding the highest classification accuracy. The number of features selected in this best feature subset is included, as well as a bit-string indicating which features were selected.

6.3.2.1 GNB classifier

Table 6.7 gives the results of the exhaustive search for the best subset of features using the GNB classifier. Using the best subset of features for each dataset, the average accuracy for the GNB classifier across the 11 datasets was 66.5%. This meant a roughly 10% relative improvement in accuracy over the 60.2% average accuracy based on all features.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.7: Exhaustive search for feature subset yielding highest accuracy using the GNB classifier.

			All features		Best set of features			
Dataset	# features	# combo's	Accuracy	(Stdev)	Accuracy	(Stdev)	# Features	Bit-string
iris	4	16	94.1 %	(0.4 %)	96.6 %	(0.2 %)	2	(0011)
corral	6	64	79.4 %	(3.7 %)	85.0 %	(2.8 %)	4	(111100)
liver	6	64	58.4 %	(0.4 %)	60.2 %	(1.2 %)	4	(110011)
monk-1	6	64	62.0 %	(1.1 %)	65.1 %	(1.0 %)	1	(000010)
monk-2	6	64	67.1 %	(0.0 %)	67.1 %	(0.0 %)	0	(000000)
pima	8	256	63.7 %	(0.8 %)	76.4 %	(0.4 %)	2	(01000100)
breasttissue	9	512	57.5 %	(1.1 %)	67.5 %	(1.7 %)	5	(110100101)
glass	9	512	35.5 %	(1.3 %)	47.4 %	(1.2 %)	2	(010100000)
tic-tac-toe	9	512	74.5 %	(0.3 %)	74.5 %	(0.3 %)	9	(111111111)
parity5-5	10	1024	37.7 %	(1.6 %)	46.2 %	(1.3 %)	0	(0000000000)
vowel	10	1024	31.7 %	(0.5 %)	45.1 %	(0.5 %)	4	(1101100000)
average			60.2 %		66.5 %			

Note that from datasets monk-2 and parity5-5, the highest accuracy for the GNB classifier came from using no features at all. In this case the GNB classifier randomly assigned a class to every instance based on the frequency of each class in the training set. For the monk-2 dataset the best accuracy using no features was equal to that using all features and actually the accuracy using the GNB classifier was the same for each feature subset. This indicates that the accuracy of the GNB classifier on this dataset was completely unaffected by feature selection.

The classification accuracy for the vowel dataset was only 45.1% even for the best subset of features selected. This accuracy clearly outperformed a random classification because this dataset contains 11 classes, meaning a random classification accuracy would be close to 11%.

6.3.2.2 J48 classifier

Table 6.8 lists the results of the exhaustive search for the best subset of features using the J48 classifier. Using the best subset of features for each dataset, the average accuracy for the J48 classifier across the 11 datasets was 83.5%. This meant only a 5% relative improvement in accuracy over the 80.0% average accuracy based on all features. On this sample of smaller datasets, the J48 classifier seemed little improved by feature selection.

Note that using the best subset of features, the J48 classifier was not able to obtain a 100% classification accuracy on any of the datasets. Only for the monk-2 and tic-tac-toe artificial datasets did feature selection not improve the classification accuracy over that of using all features, as all features are required to correctly describe the concept embedded in those two datasets.

6.3.2.3 *k*-NN classifier

Table 6.9 lists the results of the exhaustive search for the best subset of features using the *k*-NN classifier. Using the best subset of features for each dataset, the average accuracy for the *k*-NN classifier across

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.8: Exhaustive search for feature subset yielding highest accuracy using the J48 classifier.

			All features		Best set of features			
Dataset	# features	# combo's	Accuracy	(Stdev)	Accuracy	(Stdev)	# Features	Bit-string
iris	4	16	93.3 %	(0.9 %)	94.7 %	(0.2 %)	1	(0001)
corral	6	64	85.5 %	(5.5 %)	98.7 %	(2.2 %)	4	(111100)
liver	6	64	68.1 %	(2.0 %)	69.8 %	(1.8 %)	3	(001011)
monk-1	6	64	75.0 %	(0.0 %)	75.0 %	(0.0 %)	1	(000010)
monk-2	6	64	90.1 %	(1.0 %)	90.1 %	(1.0 %)	6	(111111)
pima	8	256	73.5 %	(1.1 %)	74.8 %	(0.9 %)	4	(01100101)
breasttissue	9	512	67.0 %	(3.0 %)	72.2 %	(1.8 %)	5	(100010111)
glass	9	512	68.3 %	(2.2 %)	74.1 %	(2.0 %)	6	(111000111)
tic-tac-toe	9	512	92.8 %	(0.5 %)	92.8 %	(0.5 %)	9	(111111111)
parity5-5	10	1024	86.9 %	(3.7 %)	97.1 %	(0.9 %)	6	(1111100001)
vowel	10	1024	79.1 %	(0.9 %)	79.5 %	(0.8 %)	8	(1111101101)
average			80.0 %		83.5 %			

the 11 datasets was 87.9%. This meant a 13% relative improvement in accuracy over the 77.6% average accuracy based on all features, the biggest improvement of the three classifiers. This makes sense, because the *k*-NN classifier itself (using an Euclidean distance metric with all features scaled to the same range of [0, 1]) can not apply different importance to various features like the GNB and J48 classifiers are able to do. The latter two classifiers perform some sort of feature ranking when training the classifier, thus the benefit of explicit feature selection should intuitively be less on these two classifiers.

Table 6.9: Exhaustive search for feature subset yielding highest accuracy using the *k*-NN classifier.

			All features		Best set of features			
Dataset	# features	# combo's	Accuracy	(Stdev)	Accuracy	(Stdev)	# Features	Bit-string
iris	4	16	95.6 %	(0.5 %)	96.7 %	(0.6 %)	2	(0011)
corral	6	64	93.4 %	(2.1 %)	100.0 %	(0.0 %)	4	(111100)
liver	6	64	62.9 %	(1.1 %)	68.3 %	(1.4 %)	3	(100011)
monk-1	6	64	67.1 %	(0.6 %)	100.0 %	(0.0 %)	3	(110010)
monk-2	6	64	82.7 %	(2.1 %)	82.7 %	(2.1 %)	6	(111111)
pima	8	256	74.2 %	(0.5 %)	78.4 %	(0.6 %)	6	(01101111)
breasttissue	9	512	69.4 %	(0.7 %)	72.3 %	(1.8 %)	4	(110010010)
glass	9	512	70.7 %	(0.6 %)	79.1 %	(1.0 %)	6	(111001110)
tic-tac-toe	9	512	90.3 %	(0.4 %)	90.3 %	(0.4 %)	9	(111111111)
parity5-5	10	1024	48.8 %	(1.6 %)	100.0 %	(0.0 %)	5	(1111100000)
vowel	10	1024	98.9 %	(0.2 %)	99.1 %	(0.3 %)	9	(1111111110)
average			77.6 %		87.9 %			

Note that using the best subset of features, the *k*-NN classifier was able to obtain a 100% classification accuracy on the corral, monk-1 and parity5-5 datasets. Only for the monk-2 and tic-tac-toe artificial datasets did feature selection not improve the classification accuracy over that of using all features, as all

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

features were required to correctly describe the concept embedded in these two datasets.

6.3.3 Conclusions

The question this section investigated was if the design of the fitness function and PSO wrapper method works well enough. The focus of the investigation was on the question if the classifiers yielded enough difference in the fitness values of different feature subsets compared to the variability resulting from the classification accuracy calculation. The result of the investigation were mixed: in some cases the setup worked well, in other cases it worked very poorly. This difference in the results was driven by the combination of the dataset underlying the FSP and the classifier used.

Some combinations of classifier and dataset clearly yielded a poorly posed FSP for solving by the PSO wrapper method. For these ill-posed FSPs, there was no sense of an optimum to be found by the PSO algorithm. The random fluctuations in the classification accuracy will lead to some feature subset FS_1 having the highest fitness in one run of the PSO. But in another run, some other feature subset FS_2 is likely to yield the highest fitness and be the set of features selected by the wrapper method. The quality of the PSO algorithm in guiding the search then becomes immaterial.

Within the limited set of datasets investigated, poor performance seemed to occur more frequently in artificial datasets with very few features. The datasets used in the main experiments on the FSP in this chapter, but which did not form part of the investigation in this section, all contained more or even many more features. The problem of an ill-posed FSP seems less likely to occur in those larger datasets.

In theory, it would be best to exclude all dataset-classifier combinations that work poorly from the main experiments in this chapter as these combinations do not help to compare PSO algorithms, but instead only cloud the comparison. However, without performing an exhaustive search like that outlined in this section, it is not possible to identify these situations a-priori. For small benchmark problems such a pre-selection can be done, but with real life problems, this will be impossible or at the least negate the purpose of using a non-brute force method like PSO to perform the feature selection. If the problem is small enough that a brute force approach is feasible, that brute force approach is to be preferred as it is guaranteed to always match or outperform a stochastic search. Thus it was decided to keep all combinations selected in section 6.2.1 in the main experiments, but to be aware of this issue when analyzing the results.

6.4 Exhaustive search of classifier parameter space

This section describes the setup and results of a second exhaustive search performed on a number of smaller FSP datasets. This exhaustive investigation is performed to determine if the method chosen to tune the parameterized classifiers (J48 and *k*-NN) works sufficiently well. As described in section 6.2.3.3, the classifiers with parameters need to be tuned on a given dataset in order to allow the classifier to perform an adequate classification. Such adequate classification in turn is required for the PSO to be able to successfully select features as described in section 6.2.5.

The classifier tuning method can be summarized as finding the best parameter (combination) from a limited set of choices, where the best parameter is the one that leads to the highest classification accuracy.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The crucial part of the tuning process for this investigation is that the classification is performed using *all* the dataset's features. Recall that the tuned classifier forms part of the fitness function used during the PSO search in the space of feature subsets. Therefore, during the PSO search, the classifier will repeatedly determine the classification accuracy on the dataset, but usually with *only a subset* of the features selected. The question this section needs to answer is whether tuning the classifier using all features leads to a classifier that works adequately when only a subset of features is used in classification.

The proposed tuning method choses a single parameter combination from a grid of possible combinations. This investigation tests how well the single classifier parameters chosen by the tuning method perform across *all* feature subsets in a dataset. This performance is compared to that of the other possible choices of the classifier parameters from the grid of possible combinations. Parameter combinations outside of the grid are not considered in this investigation.

This section first outlines the experimental setup for the exhaustive investigation of classifier parameters, followed by listing and discussing the results of the experiments and what conclusions are drawn from the results.

6.4.1 Experimental method

The description of the experimental method for the exhaustive search of the classifier parameter space is divided into three parts: which datasets were chosen, how the search itself was organized, and how the results per dataset were combined into a single measure for each parameter: the probability score.

6.4.1.1 Dataset selection

Similarly as discussed in the section 6.3.1.1, computational restrictions mean that only datasets with a relatively small number of features can be used in this exhaustive investigation. Since multiple classifier settings are tested for each feature subset, some of the larger datasets that could still be investigated in section 6.3 are now also considered to be too large. Therefore, the limit was set at datasets of at most nine features, resulting in a selection of nine datasets shown in table 6.10. All these datasets are part of the set of FSPs used in testing, none of the nine datasets are used in the tuning of the PSO parameters.

Table 6.10: Datasets used in exhaustive search of classifier parameter space

dataset	# instances	# classes	# attributes	# numerical	# nominal
iris	150	3	4	4	0
corral	64	2	6	0	6
glass	214	7	9	9	0
liver	345	2	6	6	0
monk-1	432	2	6	6	0
monk-2	432	2	6	6	0
pima	768	2	8	8	0
breasttissue	106	6	9	9	0
tic-tac-toe	958	2	9	9	2

This selection of datasets is a mixture of three artificially constructed datasets (monk-1, monk-2 and

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

tic-tac-toe) and six real-world datasets. The number of instances range from 64 for the iris dataset to 958 for the tic-tac-toe endgame dataset. The glass and breasttissue datasets differ from the other seven datasets in that they contain seven and six different classes respectively, while the other datasets have only two or three (for the iris dataset) classes.

6.4.1.2 Organization of exhaustive search

Experiments were conducted for both classifiers that are parameterized: the J48 decision tree with parameters l and γ and the k-NN with parameter k. The GNB classifier has no parameter and thus was not included in the experiments to determine the influence of classifier parameters.

The grids of parameters of the J48 and k-NN classifiers tested were the same as those used in the classifier tuning in the main experiments as described in section 6.2.3.3. The 49 parameter combinations tested for the J48 classifier were those resulting from the seven choices each for parameters l and γ that were listed before in table 6.3. The 10 different parameter values for k in the k-NN classifier were listed before in table 6.4.

For each dataset, an exhaustive search was conducted across all a classifier's possible parameter combinations. For each parameter combination, all feature subsets in the dataset were evaluated by computing the classification accuracy using *a single* 10-fold cross validation.

A fair comparison of the accuracy between different subsets of selected features requires that all other circumstances are as equal as possible, which includes the splits of the dataset during the 10-fold cross validation. To ensure this, a separate process was used for generating pseudo-random numbers to determine the random splits of the dataset into the training set and the testing set in the 10-fold cross validation.

The goal of the exhaustive search is to determine the quality of the classification for each of the classifier's parameter combinations across *all* feature subsets. Using this measure of quality, the performance of the parameter combination resulting from the chosen classifier tuning method can be compared to all other combinations. The quality measure used is described in the next section.

6.4.1.3 Probability score calculation

Many different measures can be constructed to combine the classification accuracies on all feature subsets to see how well each parameter combination works. This section describes one such measure, labeled the *probability score*. The probability score is built using three main building blocks:

Reverse ranking: Instead of comparing the raw classification accuracy, the accuracies for different parameter combinations on a single feature subset are reverse ranked with the highest accuracy receiving the highest and best rank.

Selection: Besides the reverse ranking, the raw classification accuracy plays another role. The accuracies for different parameter combinations on a single feature subset are compared to the highest accuracy found for that feature subset. Only parameter combinations that have an accuracy that is close enough to the best accuracy for that feature subset are considered "good".

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Probability score: The probability score combines the results of the reverse ranking and the selection. An intermediate score is calculated for a parameter combination by summing across all different feature subsets the rank obtained, *but only for those situations where the parameter combination is considered "good"*. The intermediate scores are scaled linearly to form the probability scores, such that the sum of scores across all classifier parameters equals one. The resulting probability score can be loosely interpreted as the probability that the parameter combination is the best one to use for classification on the dataset *if it is not known which features from the dataset will be used in the classification*. This situation of not knowing which features will be used, is exactly the situation present at the start of the PSO search in the wrapper method of solving the FSP.

The remainder of this subsection describes in detail the steps and calculations that go into determining the probability score for each classifier parameter combination:

1. Let D be the dataset underlying the FSP with a total of N features. Then D has an exhaustive list of I feature subsets FS_i, where $I = 2^N$.

2. Let C be the classifier used, which is parameterized. In total J different parameter combinations are considered for C, leading to J classifiers $C(j)$.

3. For each combination of feature subset FS_i and classifier $C(j)$ the average classification accuracy, $acc_{10-\text{CV}}(D,FS_i,C(j))$, is determined using 10-fold cross validation. At the same time, the standard deviation of the classification accuracies across those 10 folds is recorded and labeled $SD_{10-\text{CV}}(D,FS_i,C(j))$.

4. Then for each feature subset, FS_i, the best scoring classifier is determined, namely that classifier $C(\bar{j}_{FS_i})$ with parameter indexed $\bar{j}_{FS_i}$ that yields the highest accuracy of all $C(j)$ on D using only the features in FS_i:
$$\bar{j}_{FS_i} = \underset{j}{\operatorname{argmax}}\{acc(D,FS_i,C(j)\}, \tag{6.1}$$
with corresponding maximum accuracy
$$\begin{aligned} acc_{\max}(D,FS_i) &= acc_{10-\text{CV}}(D,FS_i,C(\bar{j}_{FS_i})) \\ &= \max_j\{acc_{10-\text{CV}}(D,FS_i,C(j))\}, \end{aligned} \tag{6.2}$$
and standard deviation $SD_{10-\text{CV}}(D,FS_i,C(\bar{j}_{FS_i}))$.

5. The reverse ranks, $r(D,j,FS_i)$, are determined separately for each feature subset FS_i by ranking the classification accuracies achieved by each of the parameters using that subset of features. The parameter that achieves the highest classification accuracy receives the highest rank:
$$r(D,j,FS_i) = \text{rank}(acc_{10-\text{CV}}(D,FS_i,C(j)) \mid acc_{10-\text{CV}}(D,FS_i,C(m),m=1,\ldots,J) \tag{6.3}$$
So for the parameter indexed $\bar{j}_{FS_i}$, which achieved the highest classification accuracy on FS_i, the resulting reverse rank, $r(D,\bar{j}_{FS_i},FS_i)$, equals J. For the worst performing parameter indexed j_-, the resulting reverse rank, $r(D,j_-,FS_i)$, equals 1. In the case of ties, reverse ranks are averaged over all tying parameter combinations.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

6. Next, assuming a normal distribution of the measurements of the accuracy and using a confidence level of 95%, the *critical level*, $\text{crit}(D, FS_i)$, is determined according to

$$\begin{aligned}\text{crit}(D,FS_i) &= acc_{\max}(D,FS_i) - \Phi(95\%)\, SD_{10-\text{CV}}(D,FS_i,C(\bar{j}_{FS_i})) \\ &= acc_{\max}(D,FS_i) - 1.645\, SD_{10-\text{CV}}(D,FS_i,C(\bar{j}_{FS_i})) \qquad (6.4)\end{aligned}$$

7. Any classifier $C(j)$ having achieved a classification accuracy on FS_i which is above this critical level is considered a "good enough" parameter for classifier C on dataset D using only features FS_i. In the next step of the calculation, such good enough parameters result in a 1 using the indicator function in the following way,

$$\mathbb{1}\{acc_{10-\text{CV}}(D,FS_i,C(j)) \geq \text{crit}(D,FS_i)\}, \qquad (6.5)$$

   while parameters whose classification accuracy on FS_i is below $\text{crit}(D,FS_i)$ result in a 0 in formula (6.5).

8. Next, $\text{score}(D, j)$ is determined as the sum of the reverse ranks $r(D, j, FS_i)$ across all feature subsets for which parameter j is considered good enough:

$$\text{score}(D, j) = \sum_{i=1}^{I} r(D, j, FS_i) * \mathbb{1}\{acc_{10-\text{CV}}(D,FS_i,C(j)) \geq \text{crit}(D,FS_i)\} \qquad (6.6)$$

9. As a final step, the scores are normalized so they equal one when summed across all parameters j. This final result is labeled the *probability score* of parameter j on the dataset D using classifier C:

$$\text{probability score}(D, j) = \frac{\text{score}(D, j)}{\sum_{m=1}^{J} \text{score}(D, m)} \qquad (6.7)$$

6.4.2 Results

This section contains the results of the probability score calculation for the J48 and k-NN classifiers across the nine selected datasets. Results for each of the two classifiers are listed and discussed in turn, by comparing the optimal parameter (the one that achieved the highest probability score) from the exhaustive search with that resulting from the chosen classifier tuning method from section 6.2.3.3. Note that rankings of the probability scores used in this section follow the normal convention of the best result receiving rank 1.

6.4.2.1 J48 classifier

For the J48 classifier, seven different choices for parameter l and seven different choices for parameter γ led to a total of 49 parameter combinations tested on nine datasets of up to nine features. Table 6.11 shows a summary of the results of the probability calculations on those nine datasets. The table shows the name of the dataset and the number of features in the left-most two columns. The next three columns grouped under "Best" show the value for l and γ that achieved the highest probability score across all feature subsets, and the probability score labeled "score" for this best parameter combination. The final four columns grouped under "Chosen" show the l and γ parameter that resulted from the classifier tuning

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

process, described in section 6.2.3.3, which used all features. The column labeled “score” indicates the probability score of this chosen parameter combination and column labeled “rank” is the rank this pair had out of the 49 combinations tested. To clarify: the “2” listed in this column for dataset glass means that the parameter combination chosen in the tuning of the J48 classifier rank second best out of the 49 combinations based on probability scores.

Table 6.11: Best and chosen parameter values found in exhaustive search for the J48 classifier.

		Best			Chosen			
Dataset	# features	l	γ	Score	l	γ	Score	Rank
iris	4	2	0.050	3.05%	18	0.275	2.50%	15
corral	6	2	0.125	4.92%	2	0.050	3.83%	7
liver	6	42	0.125	3.27%	26	0.275	2.25%	17
monk-1	6	2	0.050	2.70%	34	0.275	2.11%	22
monk-2	6	2	0.275	2.40%	2	0.425	1.89%	39
pima	8	34	0.200	2.89%	2	0.425	0.85%	48
breasttissue	9	2	0.050	9.71%	2	0.125	9.51%	2
glass	9	2	0.050	8.42%	2	0.125	7.30%	2
tic-tac-toe	9	2	0.350	4.53%	18	0.125	2.06%	19

In general, i.e. across the nine datasets, the performance of the chosen parameter combinations for the J48 classifier of l and γ was reasonable. However, large differences in performance between the datasets can be seen:

- For six out of the nine datasets investigated, the probability score of the chosen parameter combination was at least 75% of the highest probability score found. It is acknowledged that this 75% threshold was chosen somewhat arbitrarily. For the liver, pima, and tic-tac-toe datasets the threshold was not met, with the ratios of chosen probability score to best probability scores of 69%, 29%, and 45% respectively.
- For the pima dataset the probability score of the chosen combination was only 0.85%, which is less than half of the average probability score (2.04%) across all 49 combinations. Also for the monk-2 dataset, the probability score of the chosen combination was less than this average level.
- When looking at the rank of the chosen parameter combinations’ probability scores, large differences can be seen across the nine datasets. For the pima dataset, the chosen parameters performed second-to-worst, and for monk-2 the chosen parameter ranked only 39th out of 49. The other seven datasets all ranked in the top half of the 49 parameter combinations, but only three (corral, breasttissue, and glass) ranked in the top 25%. For the datasets breasttissue and glass, the chosen parameters performed very well: second best out of the 49 combinations. Across all nine datasets the average rank was 19th out of 49, equivalent to the 61% percentile where a higher percentage indicates a better rank.

A summary of the same results per parameter combination for the J48 classifier can be seen in table 6.12. For each pair of l and γ this table contains the average rank across the nine datasets. For each

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

dataset, all 49 parameter combinations are ranked on probability score, with the highest - and thus the best - score receiving the lowest rank. The bottom row gives another average, this time for a single value of l across all seven choices for γ in that column. The right-most column shows something similar: the average rank for a single value of γ across seven choices for l.

Table 6.12: Detailed results of the exhaustive search for J48 parameter values for each dataset investigated. Numbers indicate average rank of the parameter combination out of 49 possibilities across all nine datasets. Lower ranks mean better performance and are colored more darkly.

γ / l	2	10	18	26	34	42	50	average
0.050	10.2	15.9	19.9	21.1	23.4	24.3	28.2	20.4
0.125	14.2	17.8	20.6	21.7	23.9	26.3	30.9	22.2
0.200	13.4	17.9	21.6	19.6	23.8	26.5	30.4	21.9
0.275	15.6	22.0	23.2	22.0	24.8	27.8	31.3	23.8
0.350	18.9	25.2	25.5	23.5	28.1	28.7	32.1	26.0
0.425	24.6	29.0	27.2	28.3	31.1	32.4	33.2	29.4
0.500	26.6	30.2	29.2	30.4	32.6	34.6	35.4	31.3
average	17.6	22.6	23.9	23.8	26.8	28.7	31.6	

It is clear that small values of both l and γ performed best, with performance (as measured by the average rank) becoming less as either l or γ grows larger. Looking at the individual datasets, however, this behavior was not universal: for the liver and pima datasets for example, the optimal values of l are 34 and 42, respectively, and the top-left corner pair of $l = 2$ and $\gamma = 0.050$ ranked only 16th and 39th out of 49 for those two datasets.

The results from table 6.12 can also be compared to that of the chosen classifier tuning method, which scored an average rank of 19.0 across the nine datasets. Table 6.12 contains only eight entries that are *smaller* and thus better than this rank of 19.0, meaning that the chosen classifier tuning method ranked just within the 20% best parameter combinations.

More detailed results for the J48 classifier, showing the probability score for each combination of l and γ for all nine datasets, are listed in appendix C in section C.1.

6.4.2.2 *k*-NN classifier

For the k-NN classifier, 10 different choices for parameter k were tested in the classifier tuning process, ranging from 1 to 19 in steps of 2. Table 6.13 shows a summary of the results of the probability calculations on the nine datasets with few features. The table shows the name of the dataset and the number of features in the left-most two columns. The next two columns grouped under “Best” shows the value for k that was deemed to work best at classification across all possible feature subsets, and the probability score labeled “score” for this best parameter combination. The final three columns grouped under “Chosen” show the value for k chosen by the classifier tuning process described in section 6.2.3.3 using all features. The column labeled “score” indicates the probability score of this chosen parameter combination, followed by the rank this parameter achieved out of the 10 values of k tested.

In general, i.e. across the nine datasets, the performance of the chosen value of k for the k-NN

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.13: Best and chosen parameter values found in exhaustive search for the *k*-NN classifier.

		Best		Chosen		
Dataset	# features	*k*	prob.	*k*	prob.	Rank
iris	4	13	14.4%	5	7.9%	8
corral	6	19	13.1%	1	9.7%	6
liver	6	19	12.9%	3	5.0%	9
monk-1	6	1	13.8%	1	13.8%	1
monk-2	6	19	29.1%	5	2.8%	8
pima	8	19	21.2%	19	21.2%	1
breasttissue	9	1	33.4%	1	33.4%	1
glass	9	1	24.4%	1	24.4%	1
tic-tac-toe	9	17	15.8%	9	11.3%	6

classifier showed a clear split between datasets on which the chosen approach worked well versus one on which it worked poorly:

- For only four out of the nine datasets investigated, the probability score of the chosen parameter combination was at least 75% of the highest probability score found, but in all those cases (i.e. monk-1, pima, breasttissue, and glass) the chosen parameter value was also the best possible choice. Across the other five datasets, the ratio of the score for the chosen value of *k* versus highest probability scores per dataset ranged from 39% to 74% with an average of 50%.
- Of the five datasets that failed the 75% probability score ratio threshold, for only one (i.e. tic-tac-toe) did the chosen value for *k* lead to a probability score above 10%, which is the average across the 10 possible values for *k*. Across the four "good" datasets, the average probability score was 23%, more than twice the average score.
- When looking at the rank of the chosen parameter's probability scores, the clearest differences can be seen across the nine datasets. As stated before, for four datasets the best value of *k* was chosen in the classifier tuning, while for the other five the average rank was 7.4 out of 10 where 1 means best.

When the actual values of *k* that perform well or poorly are investigated, one sees that the best *k*-value was either high (19 was the highest possible value, which occurred four times, with one occurrence each of 13 and 17) or low (the remaining three datasets all had $k = 1$ as the optimal parameter value for the classifier). The chosen tuning process, however, seemed to tend towards lower values for *k* (the average across all nine datasets was 5) with the exception being the pima dataset for which the chosen (and also best) *k*-value was 19. Excluding the pima dataset, the average chosen *k*-value across the other eight datasets was 3. This dichotomy between datasets on which small values of *k* performed better versus those on which larger values of *k* worked best can also be seen in table 6.14. This table shows the distribution across the 10 values for *k* of the probability score using the *k*-NN classifier. The cells in the table are colored in grey-scale according to the value in the cell, with a darker hue indicating a higher value in the cell.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.14: Detailed results of the exhaustive search for *k*-NN parameter values for each dataset investigated. Higher probability scores are colored more darkly.

dataset	chosen *k*	1	3	5	7	9	11	13	15	17	19
iris	5	0.9%	7.4%	7.9%	12.6%	10.3%	11.8%	14.4%	10.5%	11.7%	12.5%
corral	1	9.7%	8.3%	5.5%	6.6%	9.2%	10.6%	12.8%	13.0%	11.2%	13.1%
liver	3	4.6%	5.0%	10.6%	10.7%	10.7%	10.4%	11.2%	12.0%	12.0%	12.9%
monk-1	1	13.8%	13.0%	12.7%	9.3%	9.3%	7.4%	9.9%	8.8%	7.9%	7.9%
monk-2	5	1.3%	2.5%	2.8%	3.7%	6.0%	7.6%	9.9%	13.6%	23.5%	29.1%
pima	19	0.0%	0.5%	3.0%	3.9%	6.9%	11.0%	14.9%	17.7%	20.8%	21.2%
breasttissue	1	33.4%	17.4%	15.3%	10.4%	6.5%	5.8%	5.2%	3.3%	1.7%	0.9%
glass	1	24.4%	17.6%	15.1%	10.5%	8.3%	6.2%	4.9%	4.7%	4.4%	3.9%
tic-tac-toe	9	0.5%	1.7%	4.1%	9.0%	11.3%	13.1%	13.7%	15.5%	15.8%	15.3%
average rank	4.4	6.6	6.6	6.4	5.6	5.8	6.0	4.2	4.6	4.7	4.2

On the bottom row the average rank of 4.4 for the chosen classifier tuning method is compared to the average rank for each of the parameter values separately, with a lower rank meaning better performance. For simple methods, only the choice of using $k = 19$ on all nine datasets yielded a better performance than the chosen classifier tuning method. The choice of $k = 1$ which is popular in literature was actually the worst performing parameter value.

6.4.3 Conclusions

For the J48 classifier, the investigation on the nine smaller datasets did *not* indicate a serious problem with the procedure used to tune the classifier before its use in a PSO's fitness function for solving the FSP. The procedure resulted in reasonable parameters in most cases, although it led to bad classifier parameters on some datasets, as exemplified by the poor results on the monk-2 and pima datasets. No discernible pattern could be seen that indicated poorer performance of the chosen classifier tuning method for datasets with a larger number of features.

Table 6.12 showed that small values for both l and γ worked well on the nine datasets investigated, with the pair $l = 2$ and $\gamma = 0.050$ ranking as best. No distinct pattern was discerned to see if this outperformance by smaller parameter values extended to larger datasets. The chosen classifier tuning method performed better than 80% of the "simple" choices of using one parameter combination across all nine datasets, and its performance thus is deemed adequate.

For the *k*-NN classifier in general, the dichotomy between datasets for which small *k* values worked well and those for which large values of *k* worked well is troubling, since the chosen classifier tuning method seems to favor smaller values for *k*. The number of datasets investigated is too small for any statistically significant results, but the chosen classifier tuning method was able to correctly pick a larger value for *k* on two (pima, tic-tac-toe) of the five datasets which require a large value of *k* to work well.

There is no evidence that the chosen classifier tuning method leads to unacceptable results across all

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

datasets, although in some cases, e.g. the J48 classifier on the pima dataset, the results of the investigation for the chosen parameters were very poor. The results showed that there was a large variability between the optimal classifier parameters between datasets, especially for the k-NN classifier. A simple choice such as the “always use $k = 1$ for the k-NN classifier” which has been widely used in literature was actually the worst performing choice out of those investigated: the results in table 6.14 show that this approach yielded an average rank for $k = 1$ across the nine datasets of 6.6 out of 10, with 1 being the best rank. The chosen classifier tuning method had an average rank of 4.4 out of 10, outscoring all but the parameter value $k = 19$.

6.5 PSO parameter tuning

This section describes the method used to tune the various PSO algorithms prior to their use in the main numerical experiments on the FSP and the resulting tuning results. Most studies from literature that use a PSO algorithm to solve the FSP did *not* tune the PSO algorithm before the experiments; see also section 3.4.4. Instead, these studies used parameter values from previous studies, parameter values which have been proven successful on very different problems than the domain of feature selection. Although having parameters that work “well enough” on most problem domains is preferable, this approach might not be the best choice for the FSP domain.

In the experiments using PSO algorithms described in this chapter all PSO algorithms are tuned. Section 6.5.1 describes the process used to tune the BPSO, PBPSO, CFBPSO, and SBPSO algorithms. Section 6.5.2 lists the resulting parameters for each of these four PSO algorithms.

6.5.1 Parameter tuning process

All four PSO algorithms were tuned in the same manner to ensure a fair comparison of test results. The parameter tuning process is broadly the same as that used for the MKP, which was described in section 5.3.1.

For each of the 12 combinations of a PSO algorithm and a classifier, the same process was used to tune the algorithm’s parameters, although the number of control parameters differed: BPSO has four parameters, PBPSO and CFBPSO each have six, and SBPSO has five parameters. Each algorithm-classifier pair was tuned once on each of the eight tuning datasets listed in table 6.2 in section 6.2.1.2.

Table 6.15 lists the ranges of possible parameter values used in this tuning process. The Cartesian product of the parameter value ranges for one algorithm forms the parameter space for that algorithm. For each of the four PSO algorithms, 128 parameter combinations were generated that span each algorithm’s parameter space. Only static control parameters were considered. In order to generate the parameter combinations in a manner that ensures that the parameter space was covered well, sequences of Sobol pseudo-random numbers were used according to the method proposed by Franken [36].

For each triplet of (i) PSO algorithm, (ii) classifier, and (iii) dataset the same procedure was followed: All 128 parameter combinations were used in turn as settings for the PSO algorithm under consideration, and the full process described in section 6.2.5 was followed, resulting in three outcomes: the best position found, $\widehat{Y}$, the final fitness function evaluation, $f_{\text{final}}(\widehat{Y})$, and the standard deviation of the 10 repeated

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.15: Parameter ranges used in tuning the four PSO algorithms on the FSP

algorithm	ω	c_1	c_2	$V_{\max}$		
BPSO	[0.50, 0.99]	[0.00, 5.00]	[0.00, 5.00]	[1.00, 10.00]		
algorithm	ω	c_1	c_2	$V_{\max}$	R	p_{mut}
PBPSO	[0.50, 0.99]	[0.00, 5.00]	[0.00, 5.00]	[1.00, 10.00]	[1.0, 100.0]	[0.00, 0.50]
algorithm	ω	c_1	c_2	$V_{\max}$	N_{constant}	p_{replace}
CBFPSO	[0.50, 0.99]	[0.00, 5.00]	[0.00, 5.00]	[1.00, 10.00]	$\{1,\ldots,5\}$	[0.00, 0.25]
algorithm	c_1	c_2	c_3	c_4	k	
SBPSO	[0.00, 1.00]	[0.00, 1.00]	[0.50, 5.00]	[0.50, 5.00]	$\{1,\ldots,5\}$	

accuracy calculations using 10-fold cross validation.

Using the ranking and averaging method introduced in the experiments on the MKP, which was described in section 5.3.1, these 128 final fitness values for all eight datasets were taken together to yield the best parameter combination for each algorithm-classifier pair.

6.5.2 Tuning results

The results of the PSO parameter tuning are given in table 6.16. Results are grouped per PSO algorithm in units of three lines for the three classifiers used.

For the BPSO, the tuning results showed stability across the three classifiers. For the GNB and k-NN classifiers the resulting PSO parameters were the same, and for the J48 classifier only the c_2 parameter, the social attraction, showed an important deviation with a value just under 2 compared to a value of 4.4 for the other two classifiers.

For the PBPSO, the tuning results were very different for each of the three classifiers. The momentum, ω, ranged from 0.687 for the GNB to 0.9709 for the J48. Interesting is also the balance between c_1, attraction to the personal best, and c_2, attraction to the neighborhood best: for the GNB classifier the personal best had the highest attraction, for J48 it was the neighborhood best, while for the k-NN classifiers both had a roughly equal strength of attraction. The values of the transformation parameter, R, and the mutation probability, p_{replace}, did not show any clear pattern.

For the CFBPSO, tuning results showed stability across the three classifiers. This time the GNB and J48 classifiers had the same tuned parameters. For the k-NN classifier the value of ω matched that of the other two classifiers, but the personal and social attraction had reverse importance: c_2, the attraction to the neighborhood best, was larger than c_1, the attraction to the personal best. The catfish parameters N_{constant} and p_{replace} did not show any clear pattern.

For the SBPSO, the tuning results were very different for each of the three classifiers. All three tuned values for c_2 were high within the possible range of $[0,1]$, while values for c_3 and c_4 tended towards the low or middle part of their respective ranges, $[0.5,5.0]$. Values for parameter c_1, the attraction to the personal best, showed three differing outcomes which varied across the whole range of $[0,1]$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.16: Tuned PSO parameters for FSPs

algorithm	classifier	ω	c_1	c_2	$V_{\max}$		
BPSO	GNB	0.9479	4.0234	4.4141	7.5391		
BPSO	J48	0.9709	4.4141	1.9922	5.1484		
BPSO	*k*-NN	0.9479	4.0234	4.4141	7.5391		
algorithm	classifier	ω	c_1	c_2	$V_{\max}$	R	p_{mut}
PBPSO	GNB	0.6187	1.4453	4.3359	3.7422	41.9922	0.0039
PBPSO	J48	0.9709	4.4141	1.9922	5.1484	38.8984	0.0508
PBPSO	*k*-NN	0.8828	3.2813	3.5938	2.9688	4.0938	0.0469
algorithm	classifier	ω	c_1	c_2	$V_{\max}$	N_{constant}	p_{replace}
CFBPSO	GNB	0.9709	4.4141	1.9922	5.1484	3	0.0254
CFBPSO	J48	0.9709	4.4141	1.9922	5.1484	3	0.0254
CFBPSO	*k*-NN	0.9785	2.4609	4.1016	9.2266	1	0.0762
algorithm	classifier	c_1	c_2	c_3	c_4	k	
SBPSO	GNB	0.3672	0.9141	1.5898	1.3086	2	
SBPSO	J48	0.9609	0.8828	2.2930	2.5742	2	
SBPSO	*k*-NN	0.4766	0.9922	1.9414	2.3633	3	

Detailed tuning results with outcomes per dataset for each of the classifiers can be found in section C.2 in appendix C.

6.6 Experimental results

This section gives an overview of the results of the numerical experiments using the FSP to compare the new and tuned SBPSO algorithm to tuned versions of the BPSO, PBPSO, and CFBPSO algorithms. The detailed results are not included here for reasons of conciseness, but can be found in appendix E. The results consist of three pieces of information: the first is the classification accuracy achieved by the PSO algorithms using each of the three classifiers GNB, J48, and *k*-NN. The accuracy is given as the average and standard deviation across 30 independent runs of the PSO wrapper algorithms. The second piece of information is the ranking of the four PSO algorithms on the 30 testing datasets, which formed the basis of the statistical comparison to determine which algorithm performed best. The third piece of information is the number of features selected by the PSO wrappers, given as the average and standard deviation over the 30 independent runs.

Before the results of the experiments are discussed in separate sections for each of the three classifiers used, section 6.6.1 describes an additional set of statistical tests performed to see *which datasets* actually showed a significant differentiation between the four PSO algorithms for each of the three classifiers. The PSO algorithms' performance on all datasets is compared with the performance on only those datasets that showed a significant difference in performance.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

After the discussion of the results per classifier, the results are combined across the three classifiers in section 6.6.5. Lastly, conclusions are drawn on the performance of the SBPSO on the FSP in section 6.7.

6.6.1 F-test to identify differentiating datasets

The main comparison of the results of the PSO runs on the FSP follows the statistical analysis used on the MKP and described in section 5.5 and appendix E. Section 6.3.3 of the exhaustive investigations of the fitness function for the FSP indicated, however, that for some datasets the chosen classifiers may not be able to indicate a significant difference in accuracy. Including these datasets into the overall comparison may dilute any true differences in performance such that the Iman-Davenport test (ID test) on the ranks yields no significant result. In order to try and correct for this problem, a second set of statistical comparisons was made using *only* the datasets that truly yielded a difference in performance between the PSO algorithms, based on the variability of the recorded fitness function values.

Whether difference in performance of the PSO algorithms on a single dataset was significant is determined statistically. Per dataset, the average and standard deviations of the classification accuracy across the 30 independent runs of the four PSO algorithms were compared using a one-way analysis of variance (ANOVA) F-test. This test was used to determine if the variance seen in the independent classification accuracies for each PSO algorithm was small enough compared to the variance between the average classification accuracies of the four PSO algorithms. Using a significance level of $\alpha = 5\%$, this test determined if the dataset led to statistically significant differences between the four PSO algorithms.

The results of the various F-tests are given in table 6.17 below. The F-statistic and p-value are listed. A simple label reading `TRUE` or `FALSE` indicates if the results on that classifier and dataset show a statistically significant difference using a $\alpha = 5\%$ confidence level. Note that four different PSO algorithms are compared using 30 runs for each algorithm, so the corresponding critical level for the F-test at $\alpha = 5\%$ is $F(4-1,(4-1)*(30-1)) = F(3,87) = 2.7094$.

To perform the F-test per dataset and per classifier, the average and standard deviation across the 30 classification accuracy results for each dataset and PSO algorithm were needed. These detailed results are listed in the tables in appendix C, namely in table C.16 for the GNB, table C.19 for the J48 classifier, and in table C.22 for the k-NN classifier.

The results of the $30 \times 3 = 90$ F-tests in table 6.17 show that only on a small number of datasets a statistically significant difference in performance was found at a confidence level of $\alpha = 5\%$:

- For the GNB classifier, only nine out of 30 datasets showed a significant difference under the F-test, i.e. arrhythmia, audiology, dermatology, horse-colic, movement-libras, musk-1, parkinsons, sonar, and soybean.

- For the J48 classifier, only five out of 30 datasets showed a significant difference under the F-test, i.e. hill-valley, movement-libras, musk-1, parity5-5, and sonar.

- For the k-NN classifier, only nine out of 30 datasets showed a significant difference under the F-test, i.e. arrhythmia, audiology, german, hill-valley, horse-colic, ionosphere, movement-libras, musk-1, and spectf.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.17: Datasets with significant differences on the one-way ANOVA F-test

	GNB classifier			J48 classifier			k-NN classifier		
Datasets	F-stat.	p-value	Signif.	F-stat.	p-value	Signif.	F-stat.	p-value	Signif.
arrhythmia	15.22	0.0000	**TRUE**	1.66	0.1812	FALSE	33.65	0.0000	**TRUE**
audiology	18.91	0.0000	**TRUE**	1.03	0.3852	FALSE	4.88	0.0035	**TRUE**
australian	0.40	0.7533	FALSE	0.12	0.9461	FALSE	1.56	0.2045	FALSE
bands	2.59	0.0580	FALSE	0.40	0.7520	FALSE	0.50	0.6841	FALSE
breasttissue	0.44	0.7264	FALSE	1.79	0.1546	FALSE	0.34	0.7979	FALSE
corral	0.70	0.5547	FALSE	0.08	0.9699	FALSE	0.42	0.7424	FALSE
crx	0.19	0.9005	FALSE	0.69	0.5628	FALSE	0.26	0.8541	FALSE
dermatology	6.78	0.0004	**TRUE**	1.61	0.1921	FALSE	1.74	0.1657	FALSE
german	1.24	0.3010	FALSE	0.26	0.8530	FALSE	3.32	0.0236	**TRUE**
glass	0.81	0.4915	FALSE	0.18	0.9103	FALSE	0.15	0.9313	FALSE
hill-valley	1.26	0.2931	FALSE	80.65	0.0000	**TRUE**	7.96	0.0001	**TRUE**
horse-colic	13.73	0.0000	**TRUE**	0.12	0.9454	FALSE	4.24	0.0076	**TRUE**
ionosphere	2.07	0.1095	FALSE	0.23	0.8752	FALSE	7.41	0.0002	**TRUE**
iris	1.18	0.3208	FALSE	1.31	0.2755	FALSE	0.07	0.9736	FALSE
liver	0.29	0.8304	FALSE	0.13	0.9402	FALSE	0.92	0.4361	FALSE
monk-1	0.37	0.7700	FALSE	0.80	0.5000	FALSE			FALSE
monk-2	0.06	0.9809	FALSE	0.08	0.9695	FALSE	0.45	0.7193	FALSE
movement-libras	87.88	0.0000	**TRUE**	7.52	0.0002	**TRUE**	6.34	0.0006	**TRUE**
musk-1	179.37	0.0000	**TRUE**	2.96	0.0368	**TRUE**	11.64	0.0000	**TRUE**
parity5-5	0.06	0.9802	FALSE	3.73	0.0143	**TRUE**	0.52	0.6705	FALSE
parkinsons	9.73	0.0000	**TRUE**	0.43	0.7294	FALSE	1.46	0.2312	FALSE
pima	0.35	0.7873	FALSE	0.09	0.9628	FALSE	1.03	0.3837	FALSE
sonar	11.24	0.0000	**TRUE**	3.10	0.0308	**TRUE**	1.23	0.3043	FALSE
soybean	4.21	0.0079	**TRUE**	2.49	0.0657	FALSE	0.05	0.9829	FALSE
spectf	0.39	0.7585	FALSE	1.17	0.3242	FALSE	3.07	0.0318	**TRUE**
tic-tac-toe	1.52	0.2141	FALSE	0.87	0.4580	FALSE	0.64	0.5889	FALSE
vehicle	1.04	0.3805	FALSE	0.49	0.6932	FALSE	1.03	0.3822	FALSE
vote	0.69	0.5578	FALSE	0.26	0.8530	FALSE	0.73	0.5381	FALSE
vowel	1.11	0.3505	FALSE	0.47	0.7019	FALSE	0.78	0.5095	FALSE
wdbc	1.81	0.1514	FALSE	0.88	0.4522	FALSE	0.43	0.7350	FALSE
# Significant			9			5			9

Note that these results were not sensitive to the chosen confidence level: using a confidence level of $\alpha = 1\%$, the number of selected datasets was 7, 2, and 7 for the GNB, J48, and k-NN classifiers respectively. Using a confidence level of $\alpha = 10\%$, the number of selected datasets was 10, 6, and 9 for the GNB, J48, and k-NN classifiers respectively.

In general, the comparison of PSO algorithms in this thesis uses the non-parametric ID test, as it removes the need to assume a specific distribution in the fitness values that are compared. In this specific case the use of the simpler F-test is deemed defensible: the standard deviation across the independent calculations of the classification accuracy using 10-fold cross validation include the variability due to random noise in the cross validation splits and braking of ties. Although this random noise need not be normally distributed, its distribution will likely be sufficiently well-behaved for the F-test to be usable.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Note that a one-way ANOVA F-test as performed here does *not* require normally distributed inputs.

In the main result sections below, further ID-tests are conducted to compare the performance of the four PSO algorithms, both using all 30 testing datasets and using only the datasets selected by the F-test in table 6.17.

6.6.2 GNB classifier

Table 6.18 contains a summary of the statistical test results from comparing the four PSO algorithms on the 30 testing datasets using the GNB classifier. Detailed results showing the actual classification accuracies achieved, as well as the respective ranks of the four PSO algorithms per datasets, can be found in appendix C in table C.16.

Table 6.18: Overview of statistical results on the FSP using the GNB classifier

GNB	All datasets				Selected datasets			
	BPSO	CFBPSO	PBPSO	SBPSO	BPSO	CFBPSO	PBPSO	SBPSO
Dataset	Rank	Rank	Rank	Rank	Rank	Rank	Rank	Rank
average rank	(2.67)	**(1.97)**	(3.10)	**(2.27)**	**(2.56)**	**(1.78)**	(3.78)	**(1.89)**
rank of rank	(3)	(1)	(4)	(2)	(3)	(1)	(4)	(2)
Z-score	2.1000		3.4000	0.9000	0.8520		2.1909	0.1217
p-value	0.0179		0.0003	0.1841	0.1971		0.0142	0.4516
Holm α	0.0250		0.0500	0.0167	0.0250		0.0500	0.0167
# ranked 1	3	13	4	10	1	4	0	4
# ranked 2	9	8	4	9	2	3	1	3
# ranked 3	13	6	7	4	6	2	0	1
# ranked 4	5	3	15	7	0	0	8	1
# datasets	30	30	30	30	9	9	9	9

Based on the ranks for all 30 datasets, the CFBPSO performed best with an average rank of 1.97. The ID-test indicated that the average ranks led to a statistically significant difference in performance. Further Nemenyi post-hoc tests showed that the BPSO and PBPSO underperformed with respect to the other two PSOs. However, no significant difference in performance was detected between the CFBPSO and the SBPSO.

The outperformance of the CFBPSO and SBPSO was by no means universal because the CFBPSO achieved either a first or second rank in only 21 out of 30 cases, while for the SBPSO this was true in 19 out of 30 cases. Thus, on one third of the datasets tested, the two best performing algorithms ranked as worst or second-worst. Note the difference with the results of comparing PSO algorithms on the MKP, where the SBPSO was the best performing algorithm in almost all cases, and never worse than second best.

Using the ranks on only the nine datasets that showed a statistically significant difference on the F-test, the CFBPSO still performed best with an average rank of 1.78. The ID-test indicated that a statistically significant difference in performance existed between the four PSO algorithms. Further Nemenyi post-hoc tests showed that the PBPSO underperformed, but no significant difference in performance was detected between the CFBPSO, the BPSO, and the SBPSO. So on the smaller number of datasets that

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

"pass" the F-test using the GNB classifier, the ID-test and Nemenyi post-hoc tests were less powerful: even though the difference in average rank between the CFBPSO and BPSO was larger on the nine selected datasets, this difference was no longer statistically significant.

The GNB classifier failed on the monk-2 dataset, as all four PSO algorithms achieved the same classification accuracy (67.13%) with a standard deviation of zero. This was because the classifier was unable to correctly represent the concept behind the dataset's class labels. The number of features selected can be seen in table C.18 in appendix C and this ranged around 3 with a standard deviation of 1.3 features. This is almost exactly what is to be expected from randomly choosing the features in each of the 30 independent runs from the six total number of features in the dataset. Although not on the same level of failure, the GNB classifier was also not very successful on the monk-1 dataset: only one feature was selected in all instances, resulting in the concept underlying the dataset's class label being only partially represented by the classifier using that one feature.

The GNB classifier also failed on the parity5-5 dataset, as all four PSO algorithms achieved almost the same classification accuracy (46.3%) while zero features were selected (see table C.18). That classification accuracy was the result of simply choosing the most common class label. The parity concept is something the GNB classifier was unable to represent.

The CFBPSO performed well on the ionosphere dataset, while the other three PSO algorithms struggled. What was even more remarkable, is that the standard deviation of the classification accuracy for the CFBPSO on this dataset was quite high (2.45%), while for the other three PSO algorithms this number was even above 9%. This result can be explained by looking at the average number of features selected. For the BPSO, PBPSO, and SBPSO the average number of features selected was five out of 34, with a standard deviation ranging from 3.5 to 6.2. For CFBPSO, only 2.9 features were selected on average, with a standard deviation of 1.6. The reset feature embedded in the CFBPSO meant that many times during the CFBPSO's run, particles get reset to the empty set of features, making it easier for the algorithm to find small subsets of features from a relatively large (in this case 34) set of features. Because of the large variation in the classification accuracy across the 30 independent runs for each of the four PSO algorithms, the F-test did not indicate a significant difference in performance on the ionosphere dataset, even though the CFPSO outperformed the other algorithms by at least 2.42% in classification accuracy.

Other datasets that showed a large difference in the classification accuracy achieved by the four PSO algorithms were soybean, musk-1, and movement-libras. On the soybean database the CFBPSO outperformed the SBPSO by 1.70%. In contrast, the SBPSO outperformed the CFBPSO by a wide margin on the musk-1 (by 6.04%) and the movement-libras (by 1.24%) datasets. For the other 26 datasets, the difference in classification accuracy between CFBPSO and SBPSO is less than 0.4%.

For seven different datasets (breasttissue, iris, monk-1, parity5-5, pima, tic-tac-toe, vowel) using the GNB classifier, all four PSO algorithms were able to select the same number of features in each of the 30 independent runs, leading to a standard deviation of zero in the number of features (see table C.18). Note that these were all small datasets with at most 10 features. For these simple problems, the FSP could be solved well by all PSOs resulting in a single best scoring set of features. These datasets did not help differentiate in the relative performance of the four PSO algorithms, because the ranking of the PSO algorithms on these datasets was determined by the noise in the classification accuracy instead of a true

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

difference in the quality of the features selected.

6.6.3 J48 classifier

Table 6.19 contains a summary of the statistical test results from comparing the four PSO algorithms on the 30 testing datasets using the J48 classifier. Detailed results can be found in appendix C in table C.19.

Table 6.19: Overview of statistical results on the FSP using the J48 classifier

J48	All datasets				Selected datasets			
	BPSO	CFPSO	PBPSO	SBPSO	BPSO	CFPSO	PBPSO	SBPSO
Dataset	Rank	Rank	Rank	Rank	Rank	Rank	Rank	Rank
average rank	**(2.37)**	**(2.23)**	(3.10)	**(2.30)**	**(2.40)**	**(1.40)**	(3.20)	**(3.00)**
rank of rank	(3)	(1)	(4)	(2)	(2)	(1)	(4)	(3)
Z-score	0.4000		2.6000	0.2000	1.0954		1.9718	1.7527
p-value	0.3446		0.0047	0.4207	0.1367		0.0243	0.0398
Holm α	0.0250		0.0500	0.0167	0.0167		0.0500	0.0250
# ranked 1	7	10	2	11	1	4	0	0
# ranked 2	10	8	6	6	2	0	1	2
# ranked 3	8	7	9	6	1	1	2	1
# ranked 4	5	5	13	7	1	0	2	2
# datasets	30	30	30	30	5	5	5	5

Using the ranks on all 30 datasets, the CFBPSO performed best with an average rank of 2.23. The ID-test indicated that the average ranks meant that a statistically significant difference in performance existed. Further Nemenyi post-hoc tests showed that the PBPSO underperformed the other three PSOs. However, no significant difference in performance was detected between the CFBPSO, the BPSO, and the SBPSO.

While CFBPSO achieved the best average rank, the difference with the BPSO and the SBPSO was very small (2.23 versus 2.37 and 2.30) and the SBPSO actually performed best on the highest number of datasets (11) compared to 10 for CFBPSO and seven for BPSO. Even the clearly underperforming PBPSO algorithm was able to achieve the best classification for two of the datasets, showing that the relative performance was inconsistent across the 30 testing datasets.

Based on the F-test results per dataset, only five out of the 30 testing datasets showed a statistically significant difference using the J48 classifier. This small number of selected datasets meant that, even though the ID-test showed a statistical significant difference among the four PSO algorithms on this small set, the post-hoc Nemenyi tests only showed significant under-performance by the PBPSO. Even the relatively poor average rank of 3.00 achieved by the SBPSO, could not be distinguished from the best average rank of 1.40 achieved by the CFBPSO. By consequence, the BPSO also scored equally, in statistical sense, as the CFBPSO en SBPSO.

On the monk-1 dataset, very little difference could be seen between the four PSO algorithms, each with an average accuracy of 75.01% and a very low standard deviation in accuracy. Looking at the number of features selected in table C.21, however, the average of 3.5 features with a standard deviation of 1.1 features was again very close to what is to be expected from randomly choosing the features in each

© University of Pretoria

of the 30 independent runs out of the six total features. The J48 classifier was thus unable to correctly portray the concept behind the monk-1 dataset, while the partial success reported was not dependent on the features selected.

Interestingly, the J48 classifier *was* able to correctly portray the monk-2 dataset, by selecting all six features in all cases. The variability in the average classification accuracy between the four PSO algorithms was thus fully caused by the noise in the final fitness calculation. The different relative performance which resulted in the CFBPSO achieving the rank of 1 and BPSO the rank of 4 can thus be attributed to randomness and was not indicative of a true difference in performance by the PSO in solving the FSP.

For the tic-tac-toe dataset the same applies as was noted for the monk-2 dataset in the previous paragraph: all runs for each of the PSO algorithms resulted in selecting all nine features, meaning there was no difference in performance on the FSP. The variability in the average classification accuracy between the four PSO algorithms was thus fully caused by noise in the final fitness calculation.

The J48 classifier failed on the hill-valley dataset for all PSOs as the classification accuracy was no more than 50% while the dataset has only two roughly equally occurring classes. CFBPSO scored almost 3% better than the other PSO algorithms because the reset mechanism allowed each of the 30 runs to find the empty set of features. This resulted in a higher classification accuracy than the other feature subsets, but the empty subset clearly is not the solution of the FSP. The J48 classifier was not able to capture the concept behind the hill-valley dataset.

On the movement-libras dataset, the SBPSO performed quite poorly with the lowest classification accuracy 1.6% lower than the CFBPSO. The SBPSO selected on average 53 features, which was more than 10 more features than the other three algorithms. Also, the standard deviation in the number of features was much higher for the SBPSO, indicating that the SBPSO was not able to find a reasonable solution in some of the 30 runs. Other than this and the hill-valley datasets, the difference in accuracy between CFBPSO and SBPSO was less than 0.9%.

6.6.4 *k*-NN classifier

Table 6.20 contains a summary of the statistical test results from comparing the four PSO algorithms on the 30 testing datasets using the *k*-NN classifier. Detailed results can be found in appendix C in table C.22.

Using the ranks on all 30 datasets, the SBPSO performed the best with an average rank of 1.68. The ID-test indicated that the average ranks meant that a statistically significant difference in performance existed. Further Nemenyi post-hoc tests showed that the SBPSO outperformed all other three PSO algorithms. The outperformance by the SBPSO was quite strong: it performed best on 20 out of 30 datasets. while the CFBPSO, which ranked second, performed best on 7 of the remaining datasets. Still, the SBPSO did perform worst on three datasets (i.e. corral, parkinsons, and vowel).

Based on the F-test results per dataset, only nine out of the 30 testing datasets showed a statistically significant difference using the *k*-NN classifier. This small number of selected datasets meant that, even though the ID-test showed a statistical significant difference among the nine PSO algorithms on this small set, the post-hoc Nemenyi tests were less powerful than on the full set of 30 datasets: No statistical

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.20: Overview of statistical results on the FSP using the *k*-NN classifier

k-NN	All datasets				Selected datasets			
	BPSO	CFPSO	PBPSO	SBPSO	BPSO	CFPSO	PBPSO	SBPSO
Dataset	Rank	Rank	Rank	Rank	Rank	Rank	Rank	Rank
average rank	(2.75)	(2.48)	(3.08)	**(1.68)**	(3.33)	(2.89)	**(2.78)**	**(1.00)**
rank of rank	(3)	(2)	(4)	(1)	(4)	(3)	(2)	(1)
Z-score	3.2000	2.4000	4.2000		2.5560	2.0692	1.9475	
p-value	0.0007	0.0082	0.0000		0.0053	0.0193	0.0257	
Holm α	0.0250	0.0167	0.0500		0.0500	0.0250	0.0167	
# ranked 1	2	7	0	20	0	0	0	9
# ranked 2	12.5	8.5	7.5	2.5	2	4	3	0
# ranked 3	6.5	7.5	12.5	4.5	2	2	5	0
# ranked 4	9	7	10	3	5	3	1	0
# datasets	30	30	30	30	9	9	9	9

difference could be seen between the SBPSO and the CFBPSO on the selected datasets, even though SBPSO performed best on each of the nine datasets - only for a confidence level of 7.5% can statistical outperformance be seen.

The largest difference in classification accuracy was seen on the arrhythmia dataset, where SBPSO outperformed the three other PSO algorithms by at least 3.83%. The reason behind this was likely that the SBPSO was able to find better solutions using fewer features, evidenced by the average number of features selected of 65.6 out of 279 possible features versus an average of 113 across the other three PSO algorithms. For the SBPSO, the variance in the number of features selected was much higher (standard deviation of 43.4), with the actual size of the feature subsets ranging from 16 to 183 features. The smaller subsets performed best, achieving classification accuracies up to 74%. The large number of features means that the space of feature subsets contains $9.7 \cdot 10^{83}$ points, of which only a very small portion could be searched during the PSO runs, making this a very hard problem. This result is discussed further in section 6.7.

The FSP based on the monk-1 dataset was solved perfectly by all PSOs, resulting in an average classification accuracy of 100%. As evidenced by the fact that the average number of features was between 3 and 4, the *k*-NN classifier was able to perfectly solve the classification problem on more than one feature subset: as long as features 1, 2, and 5 were included, adding one more feature did not reduce the classification from perfect using the 1-NN classifier. Effectively, this means that, out of the 64 possible feature subsets, four different subsets yield a perfect classification, making the problem too easy to allow for differentiation in the performance of the four PSO algorithms on the FSP based on this dataset.

On the parity5-5 dataset the CFBPSO performed better than the other PSO algorithms (by at least 1.67%) and was close to perfect, while achieving a much lower standard deviation in the accuracy. No difference could be seen in the number of features selected compared to the other PSOs, which at 6.8 was also higher than the five required for correct classification. The good accuracy score was caused by the fact that CFPBSO found the five right features in all 30 runs, but often still included additional

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

features as well. The k-NN classifier evidently was able to deal quite well with the redundant features. For each of the other three algorithms, exactly one out of the 30 runs failed completely and the PSOs did not select all five required features, resulting in a classification accuracy of around 50% which is equivalent to random classification. Again it was the reset mechanism that helped the CFBPSO, as every time a particle gets reset, it has a 50% chance to get reset to the subset of all features. In that case, all the five important features are included and the k-NN classifier with $k = 1$ achieves a classification accuracy close to 100%, attracting other particles in the swarm.

6.6.5 All three classifiers combined

This section combines the results per classifier into a single comparison of the four PSO algorithms. Because combining the classification accuracies for different classifiers is not useful for the purpose of testing performance on the FSP, only ranks are used in this section.

Table 6.21 contains an overview of the rankings achieved on each of the 30 testing datasets by the four PSO algorithms using all three classifiers. Each cell in the table contains the ranking achieved using the GNB, J48, and k-NN classifier respectively, separated by a colon (':'). In those cases where the F-test indicated significant differences for a dataset-classifier combination, the resulting ranks are indicated in bold. At the bottom of the table, the average rank across all 90 dataset-classifier combinations is stated, as well as the average rank across the datasets selected using the F-tests. The ID-test showed statistical significant differences in both cases, and those PSO algorithms that performed best or were indistinguishable from the best are indicated in bold.

As expected, the results of the experiments on the FSP differed according to which classifier was used: a different classifier meant different rankings of the four PSO algorithms on the datasets. Only 23 out of the 120 combinations (11%) of dataset and PSO algorithm showed results *unaffected* by the choice of classifier, meaning that the PSO algorithm obtained the same rank out of four for the dataset using each of the three classifiers. Such consistent outcomes happened three times for the BPSO (on the corral, ionosphere, and vehicle datasets), two times for the PBPSO (on the bands and dermatology datasets), and four times each for the CFBPSO (on the dermatology, liver, parity5-5, and tic-tac-toe datasets) and the SBPSO (on the horse-colic, parity5-5, tic-tac-toe, and vowel datasets). Only in three cases did such a consistent outcome mean one algorithm performed best using all three classifiers: the SBPSO on horse-colic and the tic-tac-toe datasets, and the CFBPSO on the parity5-5 dataset. In four cases the consistent outcome meant a consistently worst outcome: PBPSO on the bands and dermatology datasets, CFBPSO on the liver dataset, and SBPSO on the vowel dataset.

Although the individual ranks on single datasets varied across classifiers, in contrast the outcomes of the statistical tests to compare the PSO algorithms combing the classifiers were fairly consistent: using all 30 datasets, the CFBPSO and the SBPSO were the two best performing algorithms for all three classifiers, and only for the k-NN classifier was there a statistical significant difference with the SBPSO outperforming the other three PSOs. Using the J48 classifier, the BPSO algorithm also was statistically indistinguishable from these two algorithms, but for the other two classifiers the BPSO underperformed in a statistically significant manner. The PBPSO underperformed using each of the three classifiers. If only those datasets selected by the F-test were used, then for each of the three classifiers the lone

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 6.21: Detailed rankings of the four PSO algorithms on the FSP combining all three classifiers

Dataset	BPSO	CFBPSO	PBPSO	SBPSO
arrhythmia	**3 : 2 : 4**	**1 : 1 : 2**	**4 : 3 : 3**	**2 : 4 : 1**
audiology	**2 : 1 : 4**	**1 : 2 : 2**	**4 : 4 : 3**	**3 : 3 : 1**
australian	4 : 1 : 3	2 : 3 : 2	3 : 2 : 4	1 : 4 : 1
bands	1 : 2 : 2	2 : 1 : 3	4 : 4 : 4	3 : 3 : 1
breasttissue	3 : 3 : 2	4 : 1 : 4	2 : 2 : 3	1 : 4 : 1
corral	3 : 3 : 3	2 : 2 : 1	1 : 4 : 2	4 : 1 : 4
crx	2 : 4 : 3	1 : 3 : 2	3 : 2 : 4	4 : 1 : 1
dermatology	**1 : 1 : 2**	**3 : 3 : 3**	**4 : 4 : 4**	**2 : 2 : 1**
german	2 : 1 : **4**	1 : 4 : **2**	3 : 2 : **3**	4 : 3 : **1**
glass	3 : 2 : 2	1 : 4 : 1	4 : 3 : 4	2 : 1 : 3
hill-valley	2 : **3 : 3**	3 : **1 : 4**	4 : **4 : 2**	1 : **2 : 1**
horse-colic	**3 : 4 : 2**	**2 : 2 : 4**	**4 : 3 : 3**	**1 : 1 : 1**
ionosphere	3 : 3 : **3**	1 : 1 : **4**	4 : 4 : **2**	2 : 2 : **1**
iris	4 : 3 : 4	3 : 4 : 1	1 : 1 : 2	2 : 2 : 3
liver	3 : 3 : 2	4 : 4 : 4	1 : 2 : 3	2 : 1 : 1
monk-1	4 : 2 : x	1 : 4 : x	2 : 3 : x	3 : 1 : x
monk-2	1 : 1 : 4	2 : 2 : 1	3 : 4 : 3	4 : 3 : 2
movement-libras	**3 : 2 : 4**	**2 : 1 : 3**	**4 : 3 : 2**	**1 : 4 : 1**
musk-1	**3 : 2 : 2**	**2 : 1 : 3**	**4 : 3 : 4**	**1 : 4 : 1**
parity5-5	4 : **4** : 2	1 : **1** : 1	2 : **2** : 4	3 : **3** : 3
parkinsons	**3 : 2 : 2**	**1 : 3 : 1**	**4 : 4 : 3**	**2 : 1 : 4**
pima	2 : 3 : 4	4 : 1 : 3	3 : 4 : 2	1 : 2 : 1
sonar	**2 : 1 : 2**	**3 : 3 : 4**	**4 : 4 : 3**	**1 : 2 : 1**
soybean	**3 : 2 : 2**	**1 : 3 : 3**	**2 : 4 : 4**	**4 : 1 : 1**
spectf	3 : 2 : **4**	1 : 3 : **2**	4 : 4 : **3**	2 : 1 : **1**
tic-tac-toe	3 : 4 : 3	2 : 2 : 2	4 : 3 : 4	1 : 1 : 1
vehicle	2 : 2 : 2	1 : 1 : 4	3 : 4 : 3	4 : 3 : 1
vote	4 : 1 : 1	1 : 2 : 3	3 : 3 : 4	2 : 4 : 2
vowel	2 : 3 : 1	3 : 2 : 2	1 : 1 : 3	4 : 4 : 4
wdbc	2 : 4 : 4	3 : 2 : 1	4 : 3 : 2	1 : 1 : 3
average rank	(2.60)	**(2.23)**	(3.09)	**(2.08)**
average rank selected	(2.83)	**(2.13)**	(3.26)	**(1.78)**

statistical conclusion that could be drawn was that the PBPSO algorithm underperformed.

Only in two cases did the F-test indicate that a dataset yielded statistically significant differences for each of the three classifiers used: the movement-libras and musk-1 datasets. These datasets also showed the same behavior in that the SBPSO performed best using the GNB and k-NN classifiers, while it performed worst using the J48 classifier. For five datasets (arrhythmia, audiology, hill-valley, horse-colic, sonar) two out of three classifiers yielded a significant outcome of the F-test, while for a further seven datasets (dermatology, german, ionosphere, parity5-5, parkinsons, soybean, spectf) one out of three classifiers was able to do so. This means that for the remaining 16 datasets, none of the three classifiers yielded a large enough difference in performance to yield a significant result on the F-test.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The details underlying the statistical comparison of the four PSO algorithms using all three classifiers, and some overview statistics on these rankings, are shown in table 6.22.

Table 6.22: Overview of statistical results on the FSP combining all three classifiers

All classifiers	All datasets				Selected datasets			
	BPSO	CFBPSO	PBPSO	SBPSO	BPSO	CFBPSO	PBPSO	SBPSO
Measure	Rank	Rank	Rank	Rank	Rank	Rank	Rank	Rank
average rank	(2.60)	(**2.23**)	(3.09)	(**2.08**)	(2.83)	(**2.13**)	(3.26)	(**1.78**)
Rank of rank	(3)	(2)	(4)	(1)	(3)	(2)	(4)	(1)
Z-score	2.7135	0.7794	5.2828		2.7410	0.9137	3.8831	
p-value	0.0033	0.2179	0.0000		0.0031	0.1804	0.0001	
Holm α	0.0250	0.0167	0.0500		0.0250	0.0167	0.0500	
# ranked 1	3 : 7 : 2	13 : 10 : 7	4 : 2 : 0	10 : 11 : 20	1 : 1 : 0	4 : 4 : 0	0 : 0 : 0	4 : 0 : 9
# ranked 2	9 : 10 : 12	8 : 8 : 8	4 : 6 : 7	9 : 6 : 2	2 : 2 : 2	3 : 0 : 4	1 : 1 : 3	3 : 2 : 0
# ranked 3	13 : 8 : 6	6 : 7 : 7	7 : 9 : 12	4 : 6 : 4	6 : 1 : 2	2 : 1 : 2	0 : 2 : 5	1 : 1 : 0
# ranked 4	5 : 5 : 9	3 : 5 : 7	15 : 13 : 10	7 : 7 : 3	0 : 1 : 5	0 : 0 : 3	8 : 2 : 1	1 : 2 : 0
# ranked 1	12	30	6	41	2	8	0	13
# ranked 2	31	24	17	17	6	7	5	5
# ranked 3	27	20	28	14	9	5	7	2
# ranked 4	19	15	38	17	6	3	11	3
# datasets	90	90	90	90	23	23	23	23

The SBPSO achieved the best average rank, both on all 90 datasets (average rank of 2.08), as on the 23 selected datasets (average rank of 1.78). The outcome of the ID-test and Nemenyi post-hoc tests on the combined results in table 6.21 showed that in both cases the difference in average rank with the CFBPSO was too small to indicate a significant difference in performance between the two, while the BPSO and PBPSO were shown to underperform. Table 6.22 also contains two summaries of the number of times a PSO algorithm achieved a particular rank, shown in two blocks of four lines each labeled "# ranked". The top-most block shows a count of the ranks split by classifier in the order of GNB, J48, and *k*-NN classifier separated by colons. The bottom-most block sums these counts to a single total for the PSO algorithm.

These summaries show that the SBPSO was able to achieve the best average rank on all 30 datasets, mainly because it achieved the best rank on 41 out of 90 dataset-classifier pairs. The CFBPSO achieved the best rank on 30 dataset-classifier pairs. The BPSO and PBPSO achieved the best rank on 12 and six dataset-classifier pairs, respectively. The top ranks achieved by the SBPSO came predominately (20 from 41) from pairs using the *k*-NN classifier, while performance on the GNB and J48 classifier combined was roughly equal to that of the CFBPSO. These results indicate that the SBPSO showed more variability in performance across the three classifiers. This observation is investigated further in table 6.23.

The different average ranks across a single classifier are repeated in table 6.23 from the previous sections. The average rank labeled "All" is the same as that shown at the top in table 6.22. The row labeled "standard deviation" shows the standard deviation across the three different average ranks for each of the three classifiers. Now shown together, one can see some patterns emerging:

© University of Pretoria

Table 6.23: Overview of average rank and variability across classifiers on the FSP

Measure	BPSO	CFBPSO	PBPSO	SBPSO
average rank GNB	(2.67)	**(1.97)**	(3.10)	**(2.27)**
average rank J48	**(2.37)**	**(2.23)**	(3.10)	**(2.30)**
average rank *k*-NN	(2.75)	(2.48)	(3.08)	**(1.68)**
average rank All	(2.60)	**(2.23)**	(3.09)	**(2.08)**
standard deviation	0.20	0.26	0.01	0.35

- The under-performance of the PBPSO algorithm was consistent across the three classifiers.
- For the BPSO classifier, the performance was slightly better for the J48 classifier than the other two, but overall still reasonably equal.
- For CFBPSO the performance was unequal across classifiers where the best performance came using the GNB classifier and the worst using the *k*-NN classifier.
- For SBPSO the performance was most unequal across classifiers, evidenced by the highest standard deviation of the three average ranks per classifier. The best performance came using the *k*-NN classifier, while performance for the GNB and J48 classifiers was less.

6.7 Conclusions

This chapter had two objectives. The most important objective was to determine whether it is possible to successfully apply the SBPSO algorithm to the FSP. Here success meant yielding solutions of sufficient quality, but not necessarily solutions on par with the best approaches for solving the FSP. The investigation of SBPSO's efficiency while solving the FSP (the number of iterations, fitness function evaluations, or flops) was explicitly set out of scope. The second objective was to compare the performance of the SBPSO algorithm on the FSP to that of three other PSO algorithms known from literature. These two objectives are discussed in reverse order, followed by some additional findings relating to the experimental setup and its possible impact.

6.7.1 Comparing the SBPSO to other PSO algorithms

The SBPSO was compared to three other PSO algorithms from literature: the BPSO, the PBPSO, and the CFBPSO. The process to compare the algorithms was kept similar to that used successfully in chapter 5 on the MKP. A difference in the process was that, instead of using three different swarm topologies, in the experiments on the FSP the Von Neumann topology was used for all the PSO algorithms and in all experiments. Instead, three different *classifiers* were used to prevent the choice of a single classifier from introducing a bias in the results. These three classifiers used were the GNB, the J48, and the *k*-NN classifier.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

6.7.1.1 CFBPSO and SBPSO performed best

In short, the results of the numerical experiments conducted on the FSP in this chapter can be described as follows: the CFBPSO algorithm performed best when combined with either the GNB or J48 classifier, but this performance was not significantly better than that of the SBPSO. Using the k-NN classifier, the SBPSO algorithm performed best and in this case the outperformance of the other three algorithms *was* statistically significant.

When an F-test as described in section 6.6.1 was used to determine which datasets showed a significant difference in performance between the four PSO algorithms, only 23 out of a possible 90 combinations of dataset and classifier showed such a statistically significant difference. Using only this smaller set of datasets to compare the PSO algorithms, the statistical ID-tests to compare the PSO algorithms became less powerful. In this case, the SBPSO and the CFBPSO could not be distinguished in performance for any of the three classifiers. Also, using the GNB or the J48 classifier, the BPSO algorithm could not be shown to have a significantly different performance than the CFBPSO or the SBPSO. The PBPSO algorithm was shown to have significantly underperformed the other PSO algorithms for each of three classifiers.

In contrast to how clearly the experiments on the MKP showed outperformance by the SBPSO, the results on the FSP showed a much less clear result. One could perhaps see this as an indication that the CFBPSO, which was not used in the experiments on the MKP, was a better algorithm than the MBPSO it replaced in the line-up of PSO algorithms. Without testing the CFBPSO on the MKP, this cannot be decided. A more significant finding, however, is the fact that the relative performance seen was much less consistent on the FSP than on the MKP. While CFBPSO and SBPSO achieved the most first place ranks on the FSP, both algorithms performed worst out of the four PSO algorithms on some datasets for each of the three classifiers. On the MKP, the SBPSO never performed worse than second out of four and performed best in the overwhelming majority of cases. Even if the CFBPSO was a better algorithm to compare the SBPSO to, this does not explain the contrast of the SBPSO having performed quite poorly in some cases on the FSP, while it was so consistent on the MKP. This inconsistency is discussed further below.

6.7.1.2 Impact of choice of classifier on SBPSO

The SBPSO outperformed all other PSO algorithms on the k-NN in a statistically significant manner. On the GNB and J48 classifiers, the CFBPSO performed best, but not in a statistically significant manner. A possible explanation for this difference in performance lies in the combination of (i) how the SBPSO shows stability in the features selected, and (ii) the observation that feature selection by the PSO is more directly linked with classification accuracy when using the k-NN classifier.

Firstly, the SBPSO allows for stability in the features selected during a run of the algorithm. The SBPSO behaves with more “crispness” than the other PSO algorithms when the search has not yet converged: in the SBPSO, the randomness in which features are in- or excluded from a particle’s position is *less* than for the other PSO’s. For those algorithms, which are all equal to or based on the binary PSO, the process by which a particle’s velocity is converted to a position is the same. This process means that

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

early in the search a high degree of randomness exists in the particle's position: although some features have a velocity close to 0 or 1 and are stable, those with a velocity around 0.5 will tend to flip between in- and exclusion for consecutive iterations. In contrast, the feature subsets in the SBPSO's particles are able to better retain or exclude specific features and thus are more stable: whereas in a BPSO based algorithm the exact features selected vary strongly until the velocity per dimension (per feature) converges, the SBPSO is able to reduce this variance earlier in the search. If a feature is added to a particle's position in one iteration for the SBPSO, the chance of that feature being excluded again in the next iteration is, on average, lower than for PSO algorithms using a transformation of the velocity to a position.

An example of how the stability of features selected is likely to have caused good performance using the *k*-NN classifier was on the arrhythmia dataset, as previously discussed in section 6.6.4. The fact that the arrhythmia dataset has a large number of features, as well as that only a relatively small number of particles and iterations were used, meant that the search was still full in exploration mode and the swarm had not yet converged when the runs ended. The non-SBPSO algorithms would still have shown a lot of randomness in the swarm positions. Note that the good performance by SBPSO on the arrhythmia dataset was only for the *k*-NN classifier: when the SBPSO combined with the GNB or J48 classifier, it was not able to select small feature subsets (which in this case was good) in a similar manner. Using the *k*-NN classifier, SBPSO selected on average 66 features ranging from 16 to 183, while for the GNB and J48 classifiers these average number of features were 138 and 160 respectively.

The feature selection by the PSO algorithms in the wrapper was more closely linked to classification accuracy when using the *k*-NN classifier than when using one of the other two classifiers. This results from the fact that the GNB and J48 classifiers did *not* use all features equally in the classification task and the in- or exclusion of a single feature can lead to little to no impact on the classification accuracy using one of these two classifiers. In contrast, for the *k*-NN classifier each feature used directly impacts the distance between instances and thus was much more likely to influence the accuracy classification. Hence, the crispness that SBPSO exhibited when combined with the *k*-NN classifier, existed to a much smaller extent - if at all - when combined with the GNB and J48 classifiers. When the features selected are less closely linked with the fitness value, steering the search through the parameter subspace using a PSO algorithm becomes much harder.

6.7.2 Was the SBPSO successfully applied to the FSP?

The most important objective for this chapter was to see if the SBPSO could be successfully applied to the FSP yielding quality solutions. At a first glance, this appears to be the case: the setup of using a wrapper method combined with the SBPSO produced classification better than that using all features on most datasets for all three classifiers. Exceptions were datasets in which all features were required in classification (for example the *k*-NN classifier using all nine features in tic-tac-toe), since feature selection cannot lead to improvement in those cases. Other exceptions were combinations of classifier and dataset where the classifier was unable to represent the classification concept, for example the GNB classifier on the monk-2 dataset: here the classification accuracy was solely driven by the class distribution without regards of which features were used. In those combinations of classifier and dataset, the FSP was not solvable by the SBPSO or any of the other PSO algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

It is unclear, however, how well SBPSO's solutions were in an *absolute* sense: were the selected features the optimal ones for a given classifier? Might another algorithm have found better feature subsets? Difference in setup make it difficult to compare average classification accuracies directly to that of other studies. The fairest measure of the quality was to compare the SBPSO against other algorithms in the same setup, and here SBPSO proved to work on par or better than the PSO algorithms it was compared to. If, however, this experimental setup was flawed such that all four PSO algorithms failed to find optimal solutions, this comparison would not bring such a failure to light. Some of the results discussed in section 6.6 point to a potential problem that casts doubt on how successful the SBPSO was in solving the FSP.

6.7.2.1 Noisy fitness values

The potential problem mentioned in the previous paragraph is that the fitness function contains random fluctuations or noise. The fitness value is set equal to the average classification accuracy using a classifier and *k*-fold cross-validation. A first source of noise comes from the cross validation process: different folds used in training and testing will likely lead to different classifications and hence to a different classification accuracy. Care was taken to use a separate pseudo-random number sequence to determine the cross validation folds, ensuring full replicability of the results. Still, fitness function evaluations using the same set of features yielded slightly different results if they occurred at different times or for different particles.

A second source of noise comes from the classifier itself, which may use pseudo-random numbers to break ties in otherwise equal classification outcomes. Again, these pseudo random numbers were controlled to ensure full reproducibility of the results, but subsequent classifications using the same cross validation folds and the same features can still produce different tie breaks and hence a different classification accuracy.

The random fluctuations in the fitness function evaluation may have aversely affected the PSO search, especially if the random fluctuations exceeded the difference between the average classification accuracies achieved on different feature subsets: if a particle, representing one particular set of features selected, achieved a fitness value that scored *above* average due to a random fluctuation, this may have resulted in a false optimum. This and other particles in the swarm then became attracted to this false optimum, potentially misdirecting the PSO search. Whether this actually occurred in the experiments requires a more in depth investigation that was set out of scope for this thesis. For such an investigation, the distribution of classification accuracies for each feature subset in the dataset would need to be determined and then all particles in the swarm need to be tracked through all iterations.

The exhaustive investigation in section 6.3 constituted a less detailed analysis of the noisiness problem. Results of that investigation showed that for some of the smaller datasets, the random fluctuations in classification accuracy exceeded the difference in average classification accuracy between features subsets. It was unclear, however, whether this truly caused sufficient problems with the PSO search to make it fail. Also, since the exhaustive search could only be performed for small datasets, it is not known if the larger test datasets may have been affected.

All in all, what can be concluded is that the noise in the fitness calculations affects the relative

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

rankings of the PSO algorithms and weakens the comparison between the algorithms. To determine whether the SBPSO (and other PSOs) can truly solve the FSP in the chosen wrapper setup requires additional investigation.

6.7.3 Problems with the experimental setup

A number of choices made in the experimental setup turned out to be less optimal for solving the FSP using a PSO wrapper approach, or they turned out to partially impair the comparison of the SBPSO to other PSO algorithms. These choices are discussed in turn.

6.7.3.1 PSO parameter tuning

The PSO tuning method that was successful for the MKP was assumed to also work sufficiently well for tuning the PSO on the FSP. In particular, the choice of having separate groups of datasets for the tuning and testing process meant that the selection of representative datasets for tuning was very important. Although this approach was recommended by Salzberg [122], two possible problems surfaced. The first problem was whether a single combination of PSO parameters could perform well enough on all FSPs being considered. The second problem was whether the tuning method used was able to find such a suitable parameter combination.

The results of the experiments on the FSP showed a large variability in the performance of the PSO algorithms, evidenced by the fact that the SBPSO performed best on some datasets, but also ranked last on a substantial number of datasets. During the PSO tuning the same variability was present, as no clear outperformance could be seen by one or even by a few similar sets of parameters. Parameters that worked well on one dataset, performed very badly on other datasets in the group of tuning problems. One conclusion to be drawn is that different datasets for FSP in general have less in common than the different benchmark problems for MKP have in common. This raises the question if it is possible to tune a PSO algorithm such that it performs sufficiently well on all FSP using a single set of parameters. The results of the experiments did not answer this question.

The second problem was whether the tuning method used worked sufficiently well. The FSP, especially for the larger datasets, required a lot of computational effort to perform a single fitness function evaluation. By running a wrapper method to solve the FSP in the tuning process, and by choosing to evaluate 128 different parameter combinations spanning the parameter space for each of the four PSO algorithms, many fitness function evaluations were made in tuning. This put a restriction on the number and size of the tuning datasets. As a result, only eight datasets were included in the tuning dataset and, even though those datasets with the smallest number of features were excluded from tuning, six out of the tuning datasets had between 10 and 20 features, with only two (lung-cancer and promoter) having 56 and 57 features respectively. The risk looms large that the group of tuning datasets was too small and did not reflect the whole range of different FSPs. Perhaps it would have been better to have used a larger number of tuning datasets, and to have reduced the number of parameter combinations used in tuning.

© University of Pretoria

6.7.3.2 Classifier parameter tuning

The parameterized classifiers J48 and k-NN were tuned at the start of each PSO run, by taking that parameter combination which yielded the highest classification accuracy from a limited set of possible combinations. In this classification all features were used. During the PSO run, classification accuracies were determined across the many subsets of features using these tuned classifiers.

The fact that the parameters were chosen from a limited grid meant that optimal parameters may not have been considered in the classifier tuning process at all and that therefore optimal tuning was made impossible. Further computational effort could have been employed to search across a larger grid of possible values, or to use a search algorithm to iteratively adjust and improve parameters from a range of values. If the classifier parameter tuning was indeed impactful - it was not possible to tell from the results in this chapter - such impact would be most prominent for the γ parameter of the J48 classifier: this was the only truly continuous parameter for which only seven different parameter values were considered.

However, the fact that *all features* were used to determine the classification accuracy in the classifier tuning was likely more impactful. The investigations in section 6.4 showed that different feature subsets achieve optimal classification accuracy using different classifier parameters. The tuning method used for the classifier is premised on the assumption that parameters which work well using all features, also work adequately on all other feature subsets. It is possible that this choice introduced a bias that favored large subsets over small subsets with few features chosen. It is unclear if this bias affected the four PSO algorithms differently and thus caused a bias in the comparative results.

6.7.3.3 Classifiers cannot represent all classification concepts

The results on the experiments in this chapter showed that for some datasets, some classifiers were not able to represent the concept that underlies the dataset's class label. This effect was more pronounced for artificial datasets like, for example, monk-1, monk-2, parity5-5, and tic-tac-toe. A clear example of this was how the GNB classifier failed on the monk-2 and tic-tac-toe datasets, as indicated in section 6.3.2.1: the GNB classifier yielded the same classification accuracy as random classification. In this combination of classifier and dataset, feature selection did not improve the classification accuracy and thus yielded no information on which PSO algorithm performed better on the FSP.

Only an exhaustive search like that performed in sections 6.3 and 6.4 is able to conclusively identify all combinations of classifier and dataset where feature selection adds no value. Ideally, such cases should be removed from the comparison of the PSO algorithms, because these cases do not help distinguish in the performance of the PSO algorithms. For datasets with a large number of features, however, it is not feasible to identify such situations beforehand.

6.7.4 Data pre-processing

Data normalization is important for the k-NN classifier as it ensures that all attributes are assigned the same a priori importance in the classification process. Because the classification accuracy of the k-NN classifier is sensitive to scaling of the attributes, the classifier is not scale invariant. The accuracy of the GNB and J48 classifiers is unaffected by data normalization, as both classifiers are scale invariant.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

6.7.5 The next chapter

This chapter concludes the experimental analysis of the SBPSO. The next chapter will summarize all results from this thesis to draw final conclusions with regards to the objectives set in the preface. Also, possible future avenues of research regarding the SBPSO will be suggested.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Part IV

Conclusions and future work

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 7

Conclusions and future work

This chapter summarizes the major findings of the examined work, and provides a set of suggested future work that can be investigated as a result.

7.1 Conclusions

The main objective of the work described in this thesis was to develop and investigate a functioning, generic, set-based PSO algorithm that can solve discrete optimization problems (DOPs). To reach this goal, various sub-objectives were identified, which are discussed in turn below together with the main findings.

7.1.1 Review of existing set-based PSO algorithms

The first sub-objective was to determine whether a functioning, generic, set-based PSO algorithm already exists. For this a review of the appropriate literature was conducted, focusing on existing discrete and set-based PSO algorithms. It was concluded that such an algorithm is not yet available, as the reviewed algorithms all lacked at least one of the attributes of (i) functioning such that its use leads to good results in solving DOPs, (ii) being generically applicable to all DOP and instead of being problem specific, or (iii) not being truly set based:

- The SetPSO proposed by Neethling and Engelbrecht [102] was shown in [79] to perform badly on the MKP and hence it does not fulfill the criterion of being an algorithm that is truly functioning on this DOP.
- The algorithms proposed by Correa *et al.* [24], Bock and Hettenhausen [10] and Veenhuis [143] are not generic but each contains problem specific elements.
- In the algorithms proposed by Chen *et al.* [17], Wu *et al.* [149], and Khan and Engelbrecht [65] the candidate solution is represented by a particle position with a fixed size and which thus can not be called a true set.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.1.2 Constructing the SBPSO

Further sub-objectives were to determine the basic components that make up a PSO algorithm in order to determine what components should be present to make the new algorithm a PSO algorithm, and which additional components are required to make the new set-based algorithm a functioning algorithm. Then, these were to be combined to give a mathematical formulation of the SBPSO.

The review of the existing PSO literature had identified the components of a PSO algorithm as:

- a swarm of particles which each have a position and a velocity, whereby the position is updated by adding the velocity to the current position.
- a velocity update equation that describes the evolution of a particle's velocity. In its canonical form this contains three components:
  - a cognitive component that describes the attraction of the particle to the best position in the search space found by that particle previously,
  - a social component that describes the attraction of the particle to the best position in the search space found by any particle in its neighborhood, and
  - an inertia component that causes the velocity to retain part of the direction it currently has.
- the swarm topology.

To construct a set-based PSO, first a set-based equivalent was defined for the concepts of particle position and velocity. Operators were designed to allow for the basic operations required in the position and velocity update equations. The cognitive and social components of the velocity update equation were successfully translated to a set-based setting. It proved that the concept of inertia did not translate to a set-based setting. Hence generic mechanisms to add and remove elements were proposed which allow the SBPSO algorithm to search the entire search space, regardless of the initial particle positions in the swarm. A specific implementation of these generic mechanisms was proposed using k-tournament selection, allowing for "smart" additions and deletions.

The SBPSO was thus shown to be a generic, set-based PSO algorithm. The only part of the main objective still left to prove was whether it was also a functioning algorithm.

7.1.3 Test the SBPSO and compare it to other PSOs

For SBPSO to be a functioning algorithm, the algorithm needed to be shown to work in practice. This objective was met by testing the SBPSO on two different DOPs, namely the MKP and the FSP, and to compare the performance (in terms of quality of the solution found) of the SBPSO against discrete PSO algorithms from literature. For this comparison, PSO algorithms were chosen which had been applied to these two DOPs before: the BPSO, MBPSO, and PBPSO in the case of the MKP, and the BPSO, CFBPSO, and PBPSO in the case of the FSP.

Chapter 2 for the MKP and chapter 3 for the FSP argued that both problems were valid test beds for a discrete PSO algorithm: a review of literature showed both to be non-trivial, NP-complete problems which had been the subject of experiments involving discrete PSO algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

For the MKP a simple fitness function was available in either (i) the knapsack's value to be maximized or (ii) the percentage shortfall of the knapsack's value to the known optimum (or a known bound for this optimum) to be minimized. The second option was used in all experiments. The only design problem with regards to the fitness function for the MKP was how to incorporate the MKP's weight constraints. For this purpose, a penalty function approach was used, in which all solutions that broke at least one constraint were deemed ineligible and assigned an infinite penalty.

The experiments in chapter 5 showed that SBPSO could be successfully applied to the MKP yielding high quality solutions and SBPSO outperformed the three PSO algorithms it was compared to, BPSO, MBPSO, and PBPSO, by a considerable and statistically significant margin. PBPSO yielded better results than SBPSO in a small number of cases (for large MKPs with $m = 5$ constraints when a ring or Von Neumann topology was used for both PBPSO and SBPSO). The problems on which PBPSO outperformed, were likely caused by PBPSO having been better attuned to those specific MKPs than the SBPSO algorithm. In the remaining, more than 95% of all cases, the SBPSO algorithm was superior, regardless of problem set or topology, to the other three algorithms. The SBPSO was therefore successfully applied to the MKP.

For the FSP, in contrast, constructing the fitness function meant many choices. Three different classifiers were used, and the classification accuracy was determined using repeated 10-fold cross validation of the FSP's underlying dataset. Due to the inherent random fluctuations of the cross validation and classification, the resulting fitness function was noisy: repeated fitness function evaluations on the same position were likely to yield slightly different fitness values.

The experiments in chapter 6 showed a different picture than those on the MKP and applying the SBPSO to the FSP could not be deemed a similar success: Using the k-NN classifier, the SBPSO algorithm performed best and for this classifier the outperformance of the other three algorithms was statistically significant. Using either the GNB or J48 classifier, the CFBPSO algorithm performed best, but this performance was not significantly better than that of the SBPSO. A possible explanation for this difference in performance across classifiers may lie in the combination of (i) how the SBPSO in theory is more stable in which features are selected, and (ii) the observation that feature selection by the PSO is more directly linked with classification accuracy when using the k-NN classifier.

7.1.4 Using k-tournament selection in the SBPSO

The SBPSO algorithm used in the experiments in chapters 5 and 6 contained an operator to add elements to a particle's position using a k-tournament selection. A run of the SBPSO algorithm thus required more objective function evaluations than the other PSO algorithms it was compared to. One could argue that because of this fact, SBPSO had an advantage over the other three PSO algorithms in the experiments. The sensitivity analysis on the SBPSO also showed an improvement in performance for increasing values of k, while the value $k = 1$ (i.e. without the tournament selection) clearly underperformed.

Two findings from the tuning process, however, provided arguments that the influence of the tournament selection may have been small. First, the average value of k across the six different times SBPSO was tuned (for three topologies each on two sets of MKPs) was insufficiently large to reject the statistical hypothesis of an above average value. Second, control parameter k was the least important of SBPSO's

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

parameters in explaining performance on the MKP, which seems to contradict the idea that larger tournaments provided a significant benefit to SBPSO. Further analysis is needed to see if the effect of the added objective function evaluations on SBPSO's performance was significant.

7.1.5 Investigate SBPSO's control parameters

A third objective to be addressed in this thesis was to investigate what parameter values work well for SBPSO. For this purpose a detailed investigation and sensitivity analysis was conducted using the MKP. The tuning process compared performance results on a subset of MKPs for many parameter combinations. These parameter combinations were generated using Sobol pseudo-random numbers and spanned the whole parameter space. This allowed for a detailed sensitivity analysis of SBPSO's parameters on the MKP, indicating which values for SBPSO's control parameters led to good results.

The results from section 5.6.2 are repeated here for clarity:

c_1 (attraction to the personal best): For control parameter c_1 the sensitivity analysis very clearly showed that higher values lead to better results with $0.8 \leq c_1 < 1.0$ leading to the best results, regardless of which topology was used. A small drop-off was seen for parameter combinations with $0.9 \leq c_1 < 1.0$ compared to $0.8 \leq c_1 < 0.9$, indicating that the optimal value for c_1 lies above 0.8 but still some distance below the theoretical maximum of 1.0.

c_2 (attraction to the neighborhood best): For control parameter c_2 the area of good results was less clearly marked than for c_1. The best results were achieved by SBPSO using parameter combinations with $0.5 \leq c_2 < 1.0$ for lbest SBPSO and Von Neumann SBPSO, equivalent to half the parameter space for c_2. For gbest SBPSO in contrast, $0.3 \leq c_2 < 0.6$ yielded the best results.

c_3 (the maximum number of elements to add to the solution set using k-tournament selection): For control parameter c_3 lbest SBPSO and Von Neumann SBPSO achieved the best results for values $1.5 \leq c_3 < 2.5$, but all values of c_3 between 1.0 and 3.0 looked adequate. For gbest SBPSO, the area of the parameter space that yielded good results was more evenly spread at slightly higher values with $1.5 \leq c_3 < 3.5$ yielding the best results.

c_4 (the maximum number of elements to remove from the solution set randomly): For control parameter c_4 all values $1.5 \leq c_4 < 5.0$ showed adequate results, regardless of which topology SBPSO was paired with. The best results were found for parameter combinations where c_4 had values in either the range $2.0 \leq c_3 < 2.5$ or $3.5 \leq c_3 < 4.0$. It is not clear why two distinct peaks showed and parameter combinations with $2.5 \leq c_4 < 3.5$ underperformed: this will require further investigation.

k (the size of the tournament used to select elements to add to the solution set): For control parameter k, in general performance increased for higher values of k regardless of which topology was used. For lbest SBPSO and Von Neumann SBPSO parameter combinations with $k = 1$ (which means excluding the tournament selection completely) underperformed, while the performance increase for higher values of k topped off at $k = 6$. For gbest SBPSO a different pattern was seen, with parameter combinations with $k = 1$ not underperforming, but values $k \geq 7$ outperforming.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Besides identifying which parameter values yielded the best results on the MKP, a further analysis was performed to determine which of SBPSO's five control parameters were the most important to tune well. The relative importance of SBPSO's five control parameters was approximately the same regardless of which topology was used: parameters c_1 and c_3 were most important to the overall performance of the SBPSO algorithm. These were followed in rank of importance by c_4. The parameters c_2 and k were least influential on the overall performance of the SBPSO algorithm.

The sensitivity analysis showed that the SBPSO parameters c_3 and c_4, linked to the addition and removal of elements other than by attraction to a personal or neighborhood best, clearly influenced the algorithm's performance on the MKP. So besides the theoretical conclusion in chapter 4 that such mechanisms would be required for the SBPSO algorithm to function well, this was also validated in practice for the MKP.

7.1.6 Difference in SBPSO's performance on MKP and FSP

Two things stood out in the performance of the SBPSO in the experiments on the MKP and the FSP: firstly, the fact that the SBPSO clearly outperformed the other three PSO algorithms on the MKP, while the results were much closer on the FSP with SBPSO outperforming the other three algorithms only when the k-NN classifier was used. A caveat in this regard should be that the CFBPSO algorithm, which performed on par with the SBPSO on the FSP, was not used in the experiments on the MKP. Potentially, the CFBPSO may also perform well on the MKP and the outperformance by the SBPSO on the MKP was due to a poor selection of PSO algorithms to compare it to.

A second fact that stood out, was that the results on the MKP were much more consistent: the SBPSO performed best on 95% of the problems, with the PBPSO, BPSO, and MBPSO generally following in that order. On the FSP there is no such consistency in the relative performance of the four PSO algorithms across classifiers or datasets: each PSO algorithm performed both best *and* worst out of the set of four PSOs on a number of datasets regardless of which classifier was used.

The main determinant for the difference in SBPSO's performance on the two problems and the consistency of the results lies in the noisiness of the fitness function. On the MKP there is zero noise in the MKP itself and zero noise in the fitness function used. In this situation the SBPSO was able to show outperformance over the other algorithms.

On the FSP, however, the fitness function was noisy. A first source of noise comes from the cross validation process: different folds used in training and testing likely lead to different classifications and hence a different classification accuracy. A second source of noise came from the classifier itself, which used pseudo-random numbers to break ties in otherwise equal classification outcomes. The exhaustive investigation in section 6.3 showed that, for some of the smaller datasets, the noise in the classification accuracy exceeded the difference in average classification accuracy between subsets of features. In such cases, the PSO may have "constructed" a false optimum which steered the search away from potentially better sets of features. It is unclear whether this noise truly caused large enough problems with the PSO search to make it fail, nor if FSPs based on larger datasets were also thus affected. The inconsistency in the results of the experiments on the FSP indicate that this problem may indeed have occurred: the noise in the fitness function would have affected the outcomes for all four different PSO algorithms equally,

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

and a high enough level of noise is the best explanation for the large inconsistency in performance on the FSP.

In the final conclusion, the SBPSO was shown to be a generic, functioning, set-based PSO algorithm. The SBPSO performed very well on the MKP. In the experiments on the FSP, however, the results were mixed: the SBPSO performed best out of four PSO algorithms when used together with the *k*-NN classifier. The SBPSO performed only second best behind the CFBPSO when used together with either the GNB or the J48 classifier.

7.1.6.1 Why apply SBPSO to both the MKP and the FSP

SBPSO as constructed is a generic approach, which is capable of solving multiple problems and which would require minimal change (e.g., only to the objective function) for its application to a new domain. As such, it is mainly of interest academically. For practitioners, the argument of general applicability holds less weight: for almost all problems, approaches which incorporate domain specific knowledge will yield better results. However, two reasons why a generic approach may still be of interest outside of academia are mentioned here. Firstly, on newer problems for which the deeper structure is not yet known and no problem specific measures have been developed, a good generic approach may still be able to yield reasonable results. This same argument holds even if problem specific methods exits, but the exact type of problem is not recognized by the practitioner. Secondly, implementing a generic approach can be more cost effective than implementing many problem specific algorithms with limited scope. A generic approach may thus be used as a easily available first stab at the problem. And if the resulting solution is good enough in practice, no further resources (development effort or computation time) need to be spent in finding a better solution.

In this thesis, a generic algorithm was applied to both the MKP and the FSP, optimization problems from very different domains. For neither domain, the state-of-the-art was improved upon. These problems were selected for exactly the reason that they are very *dissimilar*. Applying the SBPSO to very different domains gives a first indication of how generically applicable the SBPSO is. In practice, when selecting an algorithm to help solve a new type of problem, a logical option would be to try an algorithm that was successful on a similar problem. It is unlikely that a practitioner would instead select an algorithm from a very dissimilar domain.

7.2 Future work

This section contains some ideas for future research on the new SBPSO algorithm. The simplest area for further research on the SBPSO would be to simply apply it to discrete optimization problems other than the MKP and the FSP. Without a clear idea of *why* to pick specific problems, however, this would not be the most efficient way to increase knowledge about the algorithm's strengths and weaknesses. Therefore other ideas are proposed in separate sections below.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.2.1 Test the CFBPSO on the MKP and compare it to the SBPSO

In chapter 5 the SBPSO was tested on the MKP and compared to the BPSO, MBPSO, and PBPSO algorithms. The SBPSO clearly outperformed the other three and the MBPSO clearly underperformed. In chapter 6, the SBPSO was tested on the FSP and compared to the CFBPSO, BPSO, and PBPSO algorithms. The CFBPSO and SBPSO performed best on the FSP, with the PBPSO underperforming. These outcomes naturally invite the tuning and testing of the CFBPSO on the MKP to see if the outperformance of the SBPSO in chapter 5 was true outperformance, or whether it was at least partially caused by the selection of PSO algorithms for comparison.

7.2.2 Investigate adding and removing elements in the SBPSO

An interesting further study into the working of the SBPSO algorithm would be on the impact of the mechanisms to add and remove elements to a position other than via attraction to personal or neighborhood bests. In this thesis, addition was done using a *k*-tournament selection while removal was done randomly from a specific subset of elements. A logical first step would be to test if the SBPSO still works if the parts of the update equations that deal with addition and removal of elements are changed to be less computationally intensive. For example, the tournament selection can be dropped. A second step could be to make these mechanisms dynamic, such that the number of random additions and removals changes during search, either in a predetermined way based on the number of iterations or by dynamically adapting the mechanism based on a measure of swarm diversity.

7.2.3 Investigate the impact of noisy environments on the SBPSO

The previous chapter describing the experiments on the FSP identified the problem of a noisy fitness function potentially impacting the search for all PSO algorithms due to attraction to a false optimum. Actually, this problem not only affects PSO algorithms but any search algorithm which uses a gradient to guide the search. In order to test if the SBPSO performed adequately on the FSP in an absolute sense, one possibility is to compare SBPSO's results to the least intelligent search algorithm, random search, using the same number of fitness function evaluations. The SBPSO uses a number of fitness function evaluations roughly equal to the number of particles in the swarm times the number of iterations. For random search, the same number of fitness function evaluations can be performed by repeatedly selecting a feature subset at random by giving each feature a chance of 0.5 to be included. The random search has no gradient or direction, so the search is not impacted by false optima.

In this way, it can be tested if the attraction to a false optimum was sufficiently detrimental such that it hurt SBPSO's search power. A sufficiently large feature subspace would be required to ensure that each algorithm can only search part of the sub-space. As a second step, this analysis can be repeated on a number of different datasets of increasing size: any trend seen in outperformance or underperformance by the SBPSO is expected to increase if the size of the feature subspace grows. If this trend is strong enough and enough datasets are used, it will show even amongst the large variation across different FSPs.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.2.4 Investigate how well the SBPSO works in dynamic environments

This thesis only studied the SBPSO on problems that had static environments: the search space did not change over time. In real life, many optimization problems exist that do change over time. Some of these problems are DOPs. The changes in the search space can be gradual over time, or more pronounced in discrete shifts. Studying the SBPSO in such dynamic environments could yield further insights into the SBPSO's strengths and weaknesses. This further study could investigate if the SBPSO needs to be adjusted, and if so, whether small adjustments to the velocity update equation suffice or whether explicit mechanisms need to be added to deal with the environment changing over time.

© University of Pretoria

Bibliography

[1] Abraham, A., Liu, H., Zhang, W., and Chang, T.-G. (2006). Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm. In Gabrys, B., Howlett, R., and Jain, L., editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 4252 of *Lecture Notes in Computer Science*, pages 500–507. Springer, Berlin/Heidelberg.

[2] Almuallim, H. and Dietterich, T. G. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1):279–305.

[3] Amaldi, E. and Kann, V. (1998). On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science*, 209(1-2):237–260.

[4] Averbakh, I. (1994). Probabilistic properties of the dual structure of the multidimensional knapsack problem and fast statistically efficient algorithms. *Mathematical Programming*, 65(1-3):311–330.

[5] Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository.

[6] Bayes, M. and Price, M. (1763). An Essay towards Solving a Problem in the Doctrine of Chances. *Philosophical Transactions*, 53:370–418.

[7] Benameur, L., Alami, J., and El Imrani, A. (2009). A new discrete particle swarm model for the frequency assignment problem. In *Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications*, pages 139–144, Piscataway, NJ. IEEE Press.

[8] Berkelaar, M., Eikland, K., and Notebaert, P. (2006). lpsolve version 5.5.

[9] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Confernece*, pages 11–18, San Francisco. Morgan Kaufmann.

[10] Bock, J. and Hettenhausen, J. (2012). Discrete particle swarm optimisation for ontology alignment. *Information Sciences*, 192(0):152–173.

[11] Bolón-Canedo, V., Sánchez-Maroño, N., and Alonso-Betanzos, A. (2013). A review of feature selection methods on synthetic data. *Knowledge and information systems*, 34(3):483–519.

[12] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT ’92, pages 144–152, New York, NY, USA. ACM.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[13] Boussier, S., Vasquez, M., Vimont, Y., Hanafi, S., and Michelon, P. (2010). A multi-level search strategy for the 0–1 Multidimensional Knapsack Problem. *Discrete Applied Mathematics*, 158(2):97–109.

[14] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

[15] Cervante, L., Xue, B., Zhang, M., and Shang, L. (2012). Binary particle swarm optimisation for feature selection: A filter based approach. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1–8.

[16] Chandrasekaran, S., Ponnambalam, S., Suresh, R., and Vijayakumar, N. (2006). A Hybrid Discrete Particle Swarm Optimization Algorithm to Solve Flow Shop Scheduling Problems. In *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, pages 1–6, Piscataway, NJ. IEEE Press.

[17] Chen, W.-N., Zhang, J., Chung, H., Zhong, W.-L., Wu, W.-G., and Shi, Y. (2010). A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems. *IEEE Transactions on Evolutionary Computation*, 14(2):278–300.

[18] Chu, P. and Beasley, J. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 4:63–86.

[19] Chuang, L.-Y., Chang, H.-W., Tu, C.-J., and Yang, C.-H. (2008a). Improved binary PSO for feature selection using gene expression data. *Computational Biology and Chemistry*, 32(1):29–38.

[20] Chuang, L.-Y., Tsai, S.-W., and Yang, C.-H. (2008b). Catfish particle swarm optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 1–5.

[21] Chuang, L.-Y., Tsai, S.-W., and Yang, C.-H. (2011). Improved binary particle swarm optimization using catfish effect for feature selection. *Expert Systems with Applications*, 38(10):12699–12707.

[22] Claessens, S., Kose, M. A., and Terrones, M. E. (2012). How do business and financial cycles interact? *Journal of International Economics*, 87(1):178–190.

[23] Clerc, M. (2004). Discrete Particle Swarm Optimization Illustrated by the Traveling Salesman Problem. In Onwubolu, G. and Babu, B., editors, *New Optimization Techniques in Engineering*, pages 219–239. Springer, Berlin/Heidelberg.

[24] Correa, E., Freitas, A., and Johnson, C. (2006). A New Discrete Particle Swarm Optimization Algorithm Applied to Attribute Selection in a Bioinformatics Data Set. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 35–42, New York, NY. ACM Press.

[25] Cover, T. M. and Van Campenhout, J. M. (1977). On the Possible Orderings in the Measurement Selection Problem. *Systems, Man and Cybernetics, IEEE Transactions on*, 7(9):657–661.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[26] Deep, K. and Bansal, J. (2008). A Socio-Cognitive Particle Swarm Optimization for Multi-Dimensional Knapsack Problem. In *Proceedings of the First International Conference on Emerging Trends in Engineering and Technology*, pages 355–360.

[27] Díaz-Uriarte, R. and De Andres, S. A. (2006). Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7(1):3.

[28] Du, J.-X., Huang, D.-S., Zhang, J., and Wang, X.-F. (2005). Shape matching using fuzzy discrete particle swarm optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 405–408, Piscataway, NJ. IEEE Press.

[29] Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

[30] Eberhart, R. C., Kennedy, J., and Shi, Y. (2001). *Swarm intelligence*. Morgan Kaufmann series in evolutionary computation. Elsevier, Amsterdam.

[31] Eberhart, R. C. and Shi, Y. (2001). Particle swarm optimization: developments, applications and resources. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 81–86, Piscataway, NJ. IEEE Press.

[32] Eberhart, R. C., Simpson, P. K., and Dobbins, R. W. (1996). *Computational Intelligence PC tools*. AP Professional, Boston, MA.

[33] Esposito, F., Malerba, D., Semeraro, G., and Kay, J. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491.

[34] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874.

[35] Fix, E. and Hodges, J. J. L. (2006). Discriminatory Analysis Nonparametric Discrimination: Consistency Properties. Technical report, Defense Technical Information Center.

[36] Franken, N. (2009). Visual exploration of algorithm parameter space. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 389–398, Piscataway, NJ. IEEE Press.

[37] Friedman, M. (1937). The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, 32(200):675–701.

[38] Furey, T. S., Cristianini, N., Duffy, N., Bednarski, D. W., Schummer, M., and Haussler, D. (2000). Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914.

[39] Gao, F., Cui, G., Zhao, Q., and Liu, H. (2006). Application of Improved Discrete Particle Swarm Algorithm in Partner Selection of Virtual Enterprise. *International Journal of Computer Science and Network Security*, 6(3A):208–212.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[40] García, A., Pastor, R., and Corominas, A. (2006). Solving the Response Time Variability Problem by means of metaheuristics. *Frontiers in artificial intelligence and applications*, 146:187–196.

[41] Garcia, S., Derrac, J., Cano, J. R., and Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435.

[42] Gavish, B. and Pirkul, H. (1986). Computer and Database Location in Distributed Computer Systems. *IEEE Transactions on Computers*, 35(7):583–590.

[43] Gens, G. and Levner, E. (1980). Complexity of approximation algorithms for combinatorial problems: a survey. *Special Interest Group on Algorithms and Computation Theory News*, 12:52–65.

[44] Gherboudj, A., Labed, S., and Chikhi, S. (2012). A New Hybrid Binary Particle Swarm Optimization Algorithm for Multidimensional Knapsack Problem. In Wyld, D. C., Zizka, J., and Nagamalai, D., editors, *Advances in Computer Science, Engineering & Applications*, volume 166 of *Advances in Intelligent and Soft Computing*, pages 489–498. Berlin/Heidelberg: Springer.

[45] Gomez, Y., Bello, R., Puris, A., Garcia, M. M., and Nowe, A. (2008). Two Step Swarm Intelligence to Solve the Feature Selection Problem. *Journal of Universal Computer Science*, 14(15):2582–2596.

[46] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

[47] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422.

[48] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18.

[49] Hall, M. A. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato.

[50] Han, J., Kamber, M., and Pei, J. (2006). *Data mining, Southeast Asia edition: Concepts and techniques*. Morgan kaufmann.

[51] Hanafi, S. and Wilbaut, C. (2011). Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Annals of Operations Research*, 183(1):125–142.

[52] Hembecker, F., Lopes, H. S., and Godoy, J. W. (2007). Particle Swarm Optimization for the Multidimensional Knapsack Problem. In *Proceedings of the International Conference on Adaptive and Natural Computing Algorithms, Part I*, pages 358–365, Berlin/Heidelberg. Springer.

[53] Holm, S. (1979). A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics*, 6(2):pp. 65–70.

© University of Pretoria

[54] Hunt, E. B., Marin, J., and Stone, P. J. (1966). *Experiments in induction*. Boston, MA: Academic Press, New York, NY.

[55] Iman, R. and Davenport, J. (1980). Approximations of the critical region of the Friedman statistic. *Communications in Statistics Part A - Theory and Methods*, 9(6):571–595.

[56] Inza, I., Larrañaga, P., Blanco, R., and Cerrolaza, A. J. (2004). Filter versus wrapper gene selection approaches in DNA microarray domains. *Artificial Intelligence in Medicine*, 31(2):91–103.

[57] Jiang, S., Pang, G., Wu, M., and Kuang, L. (2012). An improved k-nearest neighbor algorithm for text categorization. *Expert Systems with Applications*, 39(1):1503–1509.

[58] Jirapech-Umpai, T. and Aitken, S. (2005). Feature selection and classification for microarray data analysis: Evolutionary methods for identifying predictive genes. *BMC Bioinformatics*, 6(1):148.

[59] John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant Features and the Subset Selection Problem. In *Proceedings of the International Conference on Machine Learning*, pages 121–129, New Brunswick, NJ. Burlington, MA: Morgan Kaufmann.

[60] John, G. H. and Langley, P. (1995). *Estimating Continuous Distributions in Bayesian Classifiers*, volume 1, pages 338–345. Burlington, MA: Morgan Kaufmann.

[61] Kellerer, H. (1999). A Polynomial Time Approximation Scheme for the Multiple Knapsack Problem. In Hochbaum, D., Jansen, K., Rolim, J., and Sinclair, A., editors, *Randomization, Approximation, and Combinatorial Optimization. Algorithms and Techniques*, volume 1671 of *Lecture Notes in Computer Science*, pages 51–62. Berlin/Heidelberg: Springer.

[62] Kennedy, J. and Eberhart, R. (1997). A Discrete Binary Version of the Particle Swarm Algorithm. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, volume 5, pages 4101–4109, Piscataway, NJ. IEEE Press.

[63] Kennedy, J. and Eberhart, R. C. (1995). Particle Swarm Optimisation. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, NJ. IEEE Press.

[64] Kennedy, J. and Mendes, R. (2002). Population structure and particle swarm performance. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 2, pages 1671–1676, Piscataway, NJ. IEEE Press.

[65] Khan, S. and Engelbrecht, A. (2010). A fuzzy particle swarm optimization algorithm for computer communication network topology design. *Applied Intelligence*, 36:1–17.

[66] Khanesar, M., Teshnehlab, M., and Shoorehdeli, M. (2007). A Novel Binary Particle Swarm Optimization. In *Proceedings of the Mediterranean Conference on Control and Automation*, Piscataway, NJ. IEEE Press.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[67] Khuri, S., Bäck, T., and Heitkötter, J. (1994). The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the ACM Symposium on Applied Computing*, pages 188–193, New York, NY. ACM Press.

[68] Kira, K. and Rendell, L. A. (1992). A practical approach to feature selection. In *Proceedings of the Ninth International Workshop on Machine learning*, pages 249–256.

[69] Kittler, J. *et al.* (1978). Feature set search algorithms. *Pattern recognition and signal processing*, pages 41–60.

[70] Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.

[71] Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In *Proceedings of the International Conference on Machine Learning*. Stanford InfoLab.

[72] Kong, M. and Tian, P. (2006). Apply the Particle Swarm Optimization to the Multidimensional Knapsack Problem. In Rutkowski, L., Tadeusiewicz, R., Zadeh, L., and Zurada, J., editors, *Proceedings of the International Conference on Artificial Intelligence and Soft Computing*, volume 4029 of *Lecture Notes in Computer Science*, pages 1140–1149. Springer, Berlin/Heidelberg.

[73] Kong, M., Tian, P., and Kao, Y. (2008). A new ant colony optimization algorithm for the multidimensional knapsack problem. *Computers & Operations Research*, 35(8):2672–2683.

[74] Kononenko, I. (1994). Estimating attributes: analysis and extensions of RELIEF. In *Proceedings of the European Conference on Machine Learning*, pages 171–182. Springer.

[75] Kothari, V., Anuradha, J., Shah, S., and Mittal, P. (2012). A Survey on Particle Swarm Optimization in Feature Selection. In Krishna, P., Babu, M., and Ariwa, E., editors, *Global Trends in Information Systems and Software Applications*, volume 270 of *Communications in Computer and Information Science*, pages 192–201. Springer Berlin Heidelberg.

[76] Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. In *Proceedings of the Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdams. Amsterdam: IOS Press.

[77] Kotsiantis, S. B., Kanellopoulos, D., and Pintelas, P. (2006). Data Preprocessing for Supervised Learning. *International Journal of Computer Science*, 1(2):111–117.

[78] Labed, S., Gherboudj, A., and Chikhi, S. (2011). A Modified Hybrid Particle Swarm Optimization Algorithm for Multidimensional Knapsack Problem. *International Journal of Computer Applications*, 34(2):11–16.

[79] Langeveld, J. and Engelbrecht, A. P. (2011). A Generic Set-Based Particle Swarm Optimization Algorithm. In *Proceedings of the International Conference on Swarm Intelligence*, Cergy, France. EISTI.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[80] Langeveld, J. and Engelbrecht, A. P. (2012). Set-based particle swarm optimization applied to the multidimensional knapsack problem. *Swarm Intelligence*, 6(4):1–46.

[81] Langley, P. and Sage, S. (1994). Induction of Selective Bayesian Classifiers. In de Mántaras, R. L. and Poole, D., editors, *Conference on Uncertainty in Artificial Intelligence*, pages 399–406. Morgan Kaufmann.

[82] Layeb, A. (2011). A novel quantum inspired cuckoo search for knapsack problems. *International Journal of Bio-Inspired Computation*, 3(5):297–305.

[83] LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal Brain Damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann.

[84] Li, Y., Kim, J.-B., and Zhang, L. (2011). Corporate tax avoidance and stock price crash risk: Firm-level analysis. *Journal of Financial Economics*, 100(3):639–662.

[85] Liang, J. J., Qin, A. K., Suganthan, P. N., and Baskar, S. (2006). Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation*, 10(3):281–295.

[86] Liu, B., Wang, L., and Jin, Y.-H. (2007). An Effective PSO-Based Memetic Algorithm for Flow Shop Scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(1):18–27.

[87] Liu, H. and Abraham, A. (2007). An Hybrid Fuzzy Variable Neighborhood Particle Swarm Optimization Algorithm for Solving Quadratic Assignment Problems. *Journal of Universal Computer Science*, 13(9):1309–1331.

[88] Liu, H., Abraham, A., and Hassanien, A. E. (2010). Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. *Future Generation Computer Systems*, 26(8):1336–1343.

[89] Lorie, J. H. and Savage, L. J. (1955). Three Problems in Rationing Capital. *The Journal of Business*, 28:229.

[90] Loulou, R. and Michaelides, E. (1979). New Greedy-like Heuristics for the Multidimensional 0-1 Knapsack Problem. *Operations Research*, 27(6).

[91] Ma, C.-X., Qian, L., Wang, L., Menhas, M. I., and Fei, M.-R. (2010). Determination of the PID controller parameters by Modified Binary Particle Swarm Optimization algorithm. In *Proceedings of the Chinese Control and Decision Conference*, pages 2689–2694, Piscataway, NJ. IEEE Press.

[92] Ma, S. and Huang, J. (2005). Regularized ROC method for disease classification and biomarker selection with microarray data. *Bioinformatics*, 21(24):4356–4362.

[93] Marill, T. and Green, D. M. (1963). On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory*, 9(1):11–17.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[94] Matheus, C. J. and Rendell, L. A. (1989). Constructive Induction On Decision Trees. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, volume 89, pages 645–650.

[95] Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30.

[96] Menhas, M. I., Wang, L., Fei, M.-R., and Ma, C.-X. (2011). Coordinated controller tuning of a boiler turbine unit with new binary particle swarm optimization algorithm. *International Journal of Automation and Computing*, 8:185–192.

[97] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[98] Miyahara, K. and Pazzani, M. J. (2000). Collaborative Filtering with the Simple Bayesian Classifier. In *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*, pages 679–689. Springer.

[99] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. MIT press.

[100] Mosier, C. I. (1951). The need and means of cross validation. I. Problems and designs of cross-validation. *Educational and Psychological Measurement*.

[101] Narendra, P. and Fukunaga, K. (1977). A Branch and Bound Algorithm for Feature Subset Selection. *IEEE Transactions on Computers*, C-26(9):917–922.

[102] Neethling, C. and Engelbrecht, A. (2006). Determining RNA secondary structure using set-based particle swarm optimization. In Yen, G., Lucas, S., Fogel, G., Kendall, G., Salomon, R., Zhang, B.-T., Coello, C., and Runarsson, T., editors, *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1670–1677, Piscataway. NJ. IEEE Press.

[103] Nemenyi, P. (1963). *Distribution-free multiple comparisons*. PhD thesis, Princeton University, Princeton, NJ, USA.

[104] Nemhauser, G. L. and Ullmann, Z. (1969). Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505.

[105] Oh, I.-S., Lee, J.-S., and Moon, B. R. (2004). Hybrid Genetic Algorithms for Feature Selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1424–1437.

[106] Oliver, J. J. and Hand, D. J. (2014). On pruning and averaging decision trees. In *Proceedings of the 12th International Conference on Machine Learning*, pages 430–437.

[107] Olsen, A. L. (1994). Penalty functions and the knapsack problem. In *Proceedings of the First IEEE Conference on Computational Intelligence*, pages 554–558. IEEE.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[108] Omar, N., Jusoh, F., Ibrahim, R., and Othman, M. (2013). Review of feature selection for solving classification problems. *Journal of Information System Research and Innovation*, 3:64–70.

[109] Pacheco, J. A., Casado, S., Núñez, L., and Gómez, O. (2006). Analysis of new variable selection methods for discriminant analysis. *Computational Statistics & Data Analysis*, 51(3):1463–1478.

[110] Pampara, G., Franken, N., and Engelbrecht, A. (2005). Combining particle swarm optimisation with angle modulation to solve binary problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 89–96, Piscataway, NJ. IEEE Press.

[111] Pang, W., Wang, K.-P., Zhou, C.-G., and Dong, L.-J. (2004a). Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In *Proceedings of the IEEE International Conference on Computer and Information Technology*, pages 796–800, Piscataway, NJ. IEEE Press.

[112] Pang, W., Wang, K.-P., Zhou, C.-G., Dong, L.-J., Liu, M., Zhang, H.-Y., and Wang, J.-Y. (2004b). Modified particle swarm optimization based on space transformation for solving traveling salesman problem. In *Proceedings of the International Conference on Machine Learning and Cybernetics*, volume 4, pages 2342–2346, Piscataway, NJ. IEEE Press.

[113] Perkins, S., Lacker, K., and Theiler, J. (2003). Grafting: Fast, incremental feature selection by gradient descent in function space. *The Journal of Machine Learning Research*, 3:1333–1356.

[114] Powers, D. M. (2007). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63.

[115] Puchinger, J., Raidl, G. R., and Pferschy, U. (2010). The Multidimensional Knapsack Problem: Structure and Algorithms. *INFORMS Journal on Computing*, 22:250–265.

[116] Purohit, A., Chaudhari, N. S., and Tiwari, A. (2010). Construction of classifier with feature selection based on genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1–5. IEEE.

[117] Quinlan, R. J. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

[118] Quinlan, R. J. (1993). *C4.5: Programs for Machine Learning*. Burlington, MA: Morgan Kaufmann, 1 edition.

[119] Quinlan, R. J. (1996). Improved use of continuous attributes in C4. 5. *Arxiv preprint cs/9603103*.

[120] Rish, I. (2001). An empirical study of the naive Bayes classifier. In *Proceedings of the International Joint Conference on Artificial Intelligence: Workshop on Empirical Methods in Artificial Intelligence*, pages 41–46.

[121] Saeys, Y., Inza, I., and Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[122] Salzberg, S. L. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328.

[123] Sandin, I., Andrade, G., Viegas, F., Madeira, D., da Rocha, L. C., Salles, T., and Gonçalves, M. A. (2012). Aggressive and effective feature selection using genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE.

[124] Shen, B., Yao, M., and Yi, W. (2006). Heuristic Information Based Improved Fuzzy Discrete PSO Method for Solving TSP. In *Proceedings of the Pacific Rim International Conference on Artificial intelligence*, pages 859–863, Berlin/Heidelberg. Springer.

[125] Shen, Q., Jiang, J.-H., Jiao, C.-X., Shen, G.-l., and Yu, R.-Q. (2004). Modified particle swarm optimization algorithm for variable selection in MLR and PLS modeling: QSAR studies of antagonism of angiotensin II antagonists. *European Journal of Pharmaceutical Sciences*, 22(2-3):145–152.

[126] Shi, Y. and Eberhart, R. C. (1998). A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 69–73, Piscataway, NJ. IEEE Press.

[127] Shi, Y. and Eberhart, R. C. (2001). Fuzzy adaptive particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 101–106, Piscataway, NJ. IEEE Press.

[128] Sigillito, V. G., Wing, S. P., Hutton, L. V., and Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10(3):262–266.

[129] Simon, H. A. (1983). *Why Should Machines Learn?*, pages 25–37. Symbolic Computation. Springer Berlin Heidelberg.

[130] Sokolova, M., Japkowicz, N., and Szpakowicz, S. (2006). Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation. In *Proceedings of Advances in Artificial Intelligence*, pages 1015–1021. Springer.

[131] Somol, P., Novovicova, J., and Pudil, P. (2010). Efficient Feature Subset Selection and Subset Size Optimization. In *Pattern Recognition, Recent Advances*, chapter 4, pages 75–97. InTech.

[132] Soyster, A., Lev, B., and Slivka, W. (1978). Zero-one programming with many variables and few constraints. *European Journal of Operational Research*, 2(3):195–201.

[133] Stoppiglia, H., Dreyfus, G., Dubois, R., and Oussar, Y. (2003). Ranking a Random Feature for Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1399–1414.

[134] Tabakhi, S., Moradi, P., and Akhlaghian, F. (2014). An unsupervised feature selection algorithm based on ant colony optimization. *Engineering Applications of AI*, 32:112–123.

© University of Pretoria

[135] Tasgetiren, M. F., Sevkli, M., Liang, Y.-C., and Gencyilmaz, G. (2004). Particle Swarm Optimization Algorithm for Permutation Flowshop Sequencing Problem. In Dorigo, M., Birattari, M., Blum, C., M.Gambardella, L., Mondada, F., and Stützle, T., editors, *Ant Colony, Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 366–385. Springer, Berlin/Heidelberg.

[136] Thomas, J. G., Olson, J. M., Tapscott, S. J., and Zhao, L. P. (2001). An efficient and robust statistical modeling approach to discover differentially expressed genes using genomic expression profiles. *Genome Research*, 11(7):1227–1236.

[137] Tu, C.-J., Chuang, L.-Y., Chang, J.-Y., Yang, C.-H., *et al.* (2008). Feature selection using PSO-SVM. *IAENG International Journal of Computer Science*, 33(1):111–116.

[138] Tusher, V. G., Tibshirani, R., and Chu, G. (2001). Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Sciences*, 98(9):5116–5121.

[139] Unler, A. and Murat, A. (2010). A discrete particle swarm optimization method for feature selection in binary classification problems. *European Journal of Operational Research*, 206(3):528–539.

[140] Vafaie, H. and De Jong, K. (1992). Genetic algorithms as a tool for feature selection in machine learning. In *Proceedings on the 4th International Conference on Tools with Artificial Intelligence, 1992*, pages 200–203.

[141] Vasquez, M. and Hao, J.-K. (2001). A hybrid approach for the 0–1 multidimensional knapsack problem. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 328–333.

[142] Vasquez, M. and Vimont, Y. (2005). Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81.

[143] Veenhuis, C. (2008). A Set-Based Particle Swarm Optimization Method. In Rudolph, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors, *Proceedings of the Parallel Problem Solving from Nature Conference*, volume 5199 of *Lecture Notes in Computer Science*, pages 971–980. Springer, berlin/Heidelberg.

[144] Vimont, Y., Boussier, S., and Vasquez, M. (2008). Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem. *Journal of Combinatorial Optimization*, 15(2):165–178.

[145] Wang, K.-P., Huang, L., Zhou, C.-G., and Pang, W. (2003). Particle swarm optimization for traveling salesman problem. In *Proceedings of the International Conference on Machine Learning and Cybernetics*, volume 3, pages 1583–1585, Piscataway, NJ. IEEE Computer Society.

[146] Wang, L., Wang, X., Fu, J., and Zhen, L. (2008). A Novel Probability Binary Particle Swarm Optimization Algorithm and Its Application. *Journal of Software*, 3(9):28–35.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[147] Watkins, C. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, England.

[148] Weston, J., Elisseeff, A., Schölkopf, B., and Tipping, M. E. (2003). Use of the Zero-Norm with Linear Models and Kernel Methods. *Journal of Machine Learning Research*, 3:1439–1461.

[149] Wu, Z., Ni, Z., Gu, L., and Liu, X. (2010). A Revised Discrete Particle Swarm Optimization for Cloud Workflow Scheduling. In *Proceedings of the International Conference on Computational Intelligence and Security*, pages 184–188, Piscataway, NJ. IEEE Press.

[150] Xue, B., Zhang, M., and Browne, W. N. (2012). New fitness functions in binary particle swarm optimisation for feature selection. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1–8.

[151] Yan, Z. and Yuan, C. (2004). Ant Colony Optimization for Feature Selection in Face Recognition. In Zhang, D. and Jain, A. K., editors, *Proceedings of the International Conference on Biometric Authentication*, volume 3072 of *Lecture Notes in Computer Science*, pages 221–226. Springer.

[152] Yang, C.-S., Chuang, L.-Y., Li, J.-C., and Yang, C.-H. (2008). Chaotic maps in binary particle swarm optimization for feature selection. In *Proceedings of the IEEE Conference on Soft Computing in Industrial Applications*, pages 107–112.

[153] Yang, S., Wang, M., and Jiao, L. (2004). A quantum particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 320–324, Piscataway, NJ. IEEE Press.

[154] Yu, L. and Liu, H. (2003). Feature selection for high-dimensional data: a fast correlation-based filter solution. In *Proceedings of the 20th International Conference on Machine Learning*, volume 3, pages 856–863.

[155] Yun, C., Oh, B., Yang, J., and Nang, J. (2011). Feature Subset Selection Based on Bio-Inspired Algorithms. *Journal of Information Science and Engineering*, 27(5):1667–1686.

[156] Zhang, C., Sun, J., Wang, Y., and Yang, Q. (2007). An Improved Discrete Particle Swarm Optimization Algorithm for TSP. In *Proceedings of the IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, pages 35–38, Piscataway, NJ. IEEE Computer Society.

[157] Zhen, L., Wang, L., Wang, X., and Huang, Z. (2008). A Novel PSO-Inspired Probability-based Binary Optimization Algorithm. In *Proceedings of the International Symposium on Information Science and Engineering*, volume 2, pages 248–251, Oulu. Academy Publisher.

[158] Zhong, W.-L., Zhang, J., and Chen, W.-N. (2007). A novel discrete particle swarm optimization to solve traveling salesman problem. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3283–3287, Piscataway, NJ. IEEE Press.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Part V

Appendices

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Appendix A

Detailed results for small MKPs

This appendix contains the detailed results of the experiments run on the small MKPs. In total 55 such problems were considered, all of are known from the literature and are described by Chu and Beasley [18]. All problems used are available on-line at the Operations Research library[1]. The 55 problems used can be found in the files “mknap1.txt” and “mknap2.txt” and are labeled with that filename and the order in which they can be found the files: the problem labeled “mknap2-3” is thus the third problem found in the file “mknap2.txt”.

Four different algorithms (BPSO, MBPSO, PBPSO, SBPSO) were compared and each algorithm was run using three different topologies for the particle swarm: star (GB), ring (LB), and Von Neumann (VN). Each combination of algorithm and topology was first tuned on the tuning set of 15 problems. The tuned combinations of algorithm and topology were then applied on the test set of 40 problems, where 100 independent runs were simulated for each of these 12 combinations. The exact experimental set-up for both the tuning and the testing process is described in detail in sections 5.2 and 5.3.

This appendix contains detailed results of both the tuning experiments as well as the testing results based on the tuned algorithm topology pairs. An overview of all result tables for the small MKPs in this appendix can be found in table A.1.

The detailed results for the tuning experiments are shown organized by algorithm in four tables (A.2, A.3, A.4, A.5). Shown in all result tables for tuning problems is the average error across the independent runs versus the known optimum of problem followed by a rank and counter shown between brackets separated by a colon (“:”). In case of the results tables for the tuning group the rank shown is the rank of the chosen best parameter combination in the set of 128 parameter combinations for the problem named on that line. This ranking indicates how well the chosen parameter combination is suited to the problem listed compared to the other parameter combinations investigated. The counter that follows the colon is the number of 128 parameter combinations for which are tied for the lowest average error on that problem. In case the known optimum is found and the average error is zero, the counter number indicates how many parameter combination in total were able to find the known optimum in all 30 independent runs.

For the problem “mknap1-4” in table A.2 for example, a 0% average error indicates that the GB

[1]`http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table A.1: Guide to tables in Appendix A

Group	Type of comparison	Description	Table	Statistically best
Tuning	intra algorithm	BPSO	A.11	LB BPSO & VN BPSO
Tuning	intra algorithm	MBPSO	A.12	LB MBPSO & VN MBPSO
Tuning	intra algorithm	PBPSO	A.13	LB PBPSO & VN PBSPO
Tuning	intra algorithm	SBPSO	A.14	(none)
Testing	inter algorithm	Summary	A.6	SBPSO
Testing	intra algorithm	Summary	A.10	Ring & Von Neumann
Testing	inter algorithm	Star topology	A.7	GB SBPSO
Testing	inter algorithm	Ring topology	A.8	LB SBPSO
Testing	inter algorithm	Von Neumann topology	A.9	VN SBPSO & VN PBPSO
Testing	intra algorithm	BPSO	A.11	LB BPSO & VN BPSO
Testing	intra algorithm	MBPSO	A.12	LB MBPSO & VN MBPSO
Testing	intra algorithm	PBPSO	A.13	LB PBPSO & VN PBSPO
Testing	intra algorithm	SBPSO	A.14	(none)

BPSO algorithm using the chosen parameter combination was able to find the optimum solution in all 30 independent runs. The rank of 59.5 indicates a total $2 * 59.5 - 1 = 118$ parameter combinations all achieved the same best average error on problem "mknap1-4", a fact confirmed by the counter number after the colon. For the problem "mknap1-5" in table A.2 for example, the average error for the GB BPSO algorithm is 0.059%, and the rank of 70.5 stems from the fact that it shares the 69th rank with 3 other parameter combinations. The counter number 1 after the colon indicates that a single parameter combination had the best average error (this best average error being 0.005%, a fact not shown in the table).

The result tables for the experiments on the tuned algorithm topology combinations fall into two categories: inter algorithm comparisons and intra algorithm comparisons. The inter algorithm comparisons show the results of the four algorithms for one topology, resulting in three tables (A.7, A.8, A.9). The testing results by topology are summarized in table A.6. The intra algorithm comparisons show the results of all three topologies for one algorithm, resulting in four tables (A.11, A.12, A.13, A.14). The testing results by algorithm are summarized in table A.10. An overview of these result tables as well as the best topology or algorithm for each table with testing results can be found in table A.1. The best in this case is meant as statistically significant outperformance ($\alpha = 0.05$) according to a Iman-Davenport test on the average error and further Nemenyi-tests with Holm-adjusted α. For a detailed description of this statistical test see appendix E.

Shown in the result tables for the experiments are the average error over 100 independent runs versus the known optimum, as well as the success rate (SR) of finding this optimum. Also shown are the number of problems out of 40 for which all 100 runs of the algorithm successfully found the known optimum ("# perfect"), and for which it could not find the optimum in any of the runs ("# failure").

© University of Pretoria

A.1 Detailed tuning results per algorithm

A.1.1 BPSO

Table A.2: Details of the small MKP tuning results for BPSO.

Problem	*n*	*m*	GB BPSO error	GB BPSO (rank : # best)	LB BPSO error	LB BPSO (rank : # best)	VN BPSO error	VN BPSO (rank : # best)
mknap1-4	20	10	0.000 %	(59.5 : 118)	0.000 %	(57 : 113)	0.000 %	(57.5 : 114)
mknap1-5	28	10	0.059 %	(70.5 : 1)	0.000 %	(14 : 27)	0.008 %	(47.5 : 27)
mknap2-10	71	2	5.999 %	(10 : 1)	6.315 %	(6 : 1)	5.761 %	(3 : 1)
mknap2-15	30	5	0.000 %	(36 : 71)	0.000 %	(30 : 59)	0.000 %	(29 : 57)
mknap2-17	40	5	0.011 %	(1 : 1)	0.000 %	(7 : 13)	0.000 %	(6.5 : 12)
mknap2-2	60	30	0.315 %	(11 : 1)	0.199 %	(6 : 1)	0.094 %	(1 : 1)
mknap2-20	50	5	0.144 %	(14 : 1)	0.000 %	(2 : 3)	0.000 %	(3 : 5)
mknap2-26	60	5	0.048 %	(4 : 1)	0.057 %	(5 : 1)	0.008 %	(1 : 1)
mknap2-28	70	5	1.075 %	(22 : 1)	0.478 %	(7 : 1)	0.119 %	(3 : 1)
mknap2-33	80	5	1.992 %	(13 : 1)	1.520 %	(7 : 1)	0.440 %	(2 : 1)
mknap2-39	90	5	3.934 %	(15 : 1)	2.701 %	(7 : 1)	1.051 %	(2 : 1)
mknap2-4	24	2	0.000 %	(48.5 : 96)	0.000 %	(39.5 : 78)	0.000 %	(40 : 79)
mknap2-41	27	4	0.213 %	(4 : 1)	0.257 %	(17.5 : 1)	0.195 %	(8 : 1)
mknap2-45	40	30	0.662 %	(3 : 1)	0.000 %	(1 : 1)	0.155 %	(8 : 3)
mknap2-48	35	4	2.017 %	(86 : 1)	0.415 %	(22 : 1)	0.453 %	(25 : 1)
average			1.098 %	(26.5 : *n.a.*)	0.796 %	(15.2 : *n.a.*)	0.552 %	(15.8 : *n.a.*)
# perfect			3		7		5	
average non-perfect			1.372 %	(21.1 : *n.a.*)	1.493 %	(9.7 : *n.a.*)	0.828 %	(10.1 : *n.a.*)

© University of Pretoria

A.1.2 MBPSO

Table A.3: Details of the small MKP tuning results for MBPSO.

Problem	*n*	*m*	GB MBPSO error	GB MBPSO (rank : # best)	LB MBPSO error	LB MBPSO (rank : # best)	VN MBPSO error	VN MBPSO (rank : # best)
mknap1-4	20	10	0.000 %	(9 : 17)	0.000 %	(13 : 25)	0.000 %	(10.5 : 20)
mknap1-5	28	10	0.073 %	(10 : 2)	0.016 %	(16.5 : 3)	0.013 %	(15 : 2)
mknap2-10	71	2	6.693 %	(1 : 1)	5.690 %	(1 : 1)	5.631 %	(1 : 1)
mknap2-15	30	5	0.000 %	(30.5 : 60)	0.000 %	(26 : 51)	0.000 %	(29 : 57)
mknap2-17	40	5	0.302 %	(14 : 1)	0.011 %	(1 : 1)	0.063 %	(12 : 2)
mknap2-2	60	30	0.642 %	(7 : 1)	0.255 %	(9 : 1)	0.153 %	(1 : 1)
mknap2-20	50	5	0.462 %	(5 : 1)	0.056 %	(4 : 1)	0.007 %	(1 : 1)
mknap2-26	60	5	0.673 %	(29 : 1)	0.083 %	(1 : 1)	0.119 %	(1 : 1)
mknap2-28	70	5	0.576 %	(6 : 1)	0.163 %	(6 : 1)	0.170 %	(11 : 1)
mknap2-33	80	5	1.434 %	(7 : 1)	0.541 %	(12 : 1)	0.306 %	(5 : 1)
mknap2-39	90	5	1.903 %	(14 : 1)	0.684 %	(5 : 1)	0.515 %	(2 : 1)
mknap2-4	24	2	0.022 %	(15 : 1)	0.000 %	(2.5 : 4)	0.008 %	(9.5 : 7)
mknap2-41	27	4	1.022 %	(42.5 : 1)	0.350 %	(3 : 1)	0.403 %	(3.5 : 1)
mknap2-45	40	30	3.767 %	(25 : 1)	0.988 %	(11 : 1)	0.662 %	(6 : 1)
mknap2-48	35	4	1.556 %	(26 : 1)	0.593 %	(12 : 1)	0.343 %	(1 : 1)
average			1.275 %	(16.1 : *n.a.*)	0.629 %	(8.2 : *n.a.*)	0.560 %	(7.2 : *n.a.*)
# perfect			2		3		2	
average non-perfect			1.471 %	(15.5 : *n.a.*)	0.786 %	(6.8 : *n.a.*)	0.646 %	(5.3 : *n.a.*)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.1.3 PBPSO

Table A.4: Details of the small MKP tuning results for PBPSO.

Problem	*n*	*m*	GB PBPSO error	GB PBPSO (rank : # best)	LB PBPSO error	LB PBPSO (rank : # best)	VN PBPSO error	VN PBPSO (rank : # best)
mknap1-4	20	10	0.000 %	(35 : 69)	0.000 %	(36 : 71)	0.000 %	(36 : 71)
mknap1-5	28	10	0.043 %	(28 : 1)	0.003 %	(24.5 : 20)	0.005 %	(28.5 : 19)
mknap2-10	71	2	5.588 %	(2 : 1)	5.484 %	(6 : 1)	5.428 %	(5 : 1)
mknap2-15	30	5	0.000 %	(28 : 55)	0.000 %	(26 : 51)	0.000 %	(25.5 : 50)
mknap2-17	40	5	0.101 %	(17 : 1)	0.000 %	(8.5 : 16)	0.000 %	(8 : 15)
mknap2-2	60	30	0.196 %	(3.5 : 1)	0.092 %	(4 : 1)	0.066 %	(4 : 1)
mknap2-20	50	5	0.041 %	(2 : 1)	0.000 %	(3 : 5)	0.000 %	(4 : 7)
mknap2-26	60	5	0.035 %	(8 : 1)	0.013 %	(8 : 1)	0.005 %	(7 : 1)
mknap2-28	70	5	0.197 %	(7 : 1)	0.043 %	(5 : 1)	0.021 %	(5 : 1)
mknap2-33	80	5	0.500 %	(10 : 1)	0.205 %	(9 : 1)	0.176 %	(8 : 1)
mknap2-39	90	5	0.524 %	(7 : 1)	0.558 %	(9 : 2)	0.498 %	(9 : 1)
mknap2-4	24	2	0.000 %	(29 : 57)	0.000 %	(27 : 53)	0.000 %	(24 : 47)
mknap2-41	27	4	0.428 %	(23 : 1)	0.289 %	(24 : 1)	0.303 %	(22 : 1)
mknap2-45	40	30	1.027 %	(11 : 1)	0.000 %	(2 : 3)	0.168 %	(12 : 2)
mknap2-48	35	4	0.983 %	(18 : 1)	0.120 %	(1 : 1)	0.087 %	(1 : 1)
average			0.644 %	(15.2 : *n.a.*)	0.454 %	(12.9 : *n.a.*)	0.450 %	(13.3 : *n.a.*)
# perfect			3		6		5	
average non-perfect			0.805 %	(11.4 : *n.a.*)	0.756 %	(10.1 : *n.a.*)	0.676 %	(10.2 : *n.a.*)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.1.4 SBPSO

Table A.5: Details of the small MKP tuning results for SBPSO.

Problem	*n*	*m*	GB SBPSO error	GB SBPSO (rank : # best)	LB SBPSO error	LB SBPSO (rank : # best)	VN SBPSO error	VN SBPSO (rank : # best)
mknap1-4	20	10	0.000 %	(29 : 57)	0.000 %	(30 : 59)	0.000 %	(28.5 : 56)
mknap1-5	28	10	0.000 %	(6.5 : 12)	0.000 %	(17 : 33)	0.000 %	(17 : 33)
mknap2-10	71	2	5.199 %	(1 : 1)	5.211 %	(2 : 1)	5.210 %	(1.5 : 2)
mknap2-15	30	5	0.000 %	(35 : 69)	0.000 %	(36 : 71)	0.000 %	(36.5 : 72)
mknap2-17	40	5	0.000 %	(4.5 : 8)	0.000 %	(23.5 : 46)	0.000 %	(23 : 45)
mknap2-2	60	30	0.076 %	(2 : 1)	0.028 %	(2 : 1)	0.021 %	(1 : 1)
mknap2-20	50	5	0.000 %	(4 : 7)	0.000 %	(9 : 17)	0.000 %	(8.5 : 16)
mknap2-26	60	5	0.001 %	(1.5 : 2)	0.000 %	(3.5 : 6)	0.000 %	(3 : 5)
mknap2-28	70	5	0.000 %	(1 : 1)	0.002 %	(4 : 1)	0.002 %	(2.5 : 4)
mknap2-33	80	5	0.002 %	(1 : 1)	0.002 %	(1 : 1)	0.004 %	(1 : 1)
mknap2-39	90	5	0.017 %	(2 : 1)	0.000 %	(1.5 : 2)	0.000 %	(2 : 3)
mknap2-4	24	2	0.000 %	(26.5 : 52)	0.000 %	(26 : 51)	0.000 %	(26 : 51)
mknap2-41	27	4	0.196 %	(1 : 1)	0.213 %	(3 : 1)	0.225 %	(3 : 1)
mknap2-45	40	30	0.000 %	(1 : 1)	0.000 %	(12.5 : 24)	0.000 %	(13 : 25)
mknap2-48	35	4	0.394 %	(8 : 1)	0.081 %	(1 : 1)	0.077 %	(2 : 1)
average			0.392 %	(8.3 : *n.a.*)	0.369 %	(11.5 : *n.a.*)	0.369 %	(11.2 : *n.a.*)
# perfect			8		9		9	
average non-perfect			0.841 %	(2.4 : *n.a.*)	0.923 %	(2.2 : *n.a.*)	0.923 %	(1.8 : *n.a.*)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.2 Summarized testing results per topology

Table A.6: Summary of the small MKP test results per topology. Bold face indicates statistically significant outperformance of one algorithm for that topology.

problem	GB BPSO error	(rank)	GB MBPSO error	(rank)	GB PBPSO error	(rank)	GB SBPSO error	(rank)
average error	1.117 %	(2.80)	1.089 %	(3.56)	0.628 %	(2.45)	**0.444 %**	(1.19)
stdev error	1.913 %		1.592 %		1.625 %		1.640 %	
average SR	42.8 %	(2.81)	29.9 %	(3.38)	51.4 %	(2.50)	82.5 %	(1.31)
stdev SR	41.3 %		34.7 %		35.1 %		31.8 %	
# perfect	5	(2.5)	3	(4)	5	(2.5)	21	(1)
# failure	11	(4)	4	(2)	4	(2)	4	(2)
Z-score	5.58		8.21		4.36			
p-value	0.0000		0.0000		0.0000			
Holm α	0.0250		0.0500		0.0167			
	LB BPSO		LB MBPSO		LB PBPSO		LB SBPSO	
average error	0.841 %	(2.95)	0.639 %	(3.35)	0.521 %	(2.31)	**0.440 %**	(1.39)
stdev error	1.716 %		1.620 %		1.634 %		1.641 %	
average SR	50.3 %	(2.93)	45.7 %	(3.24)	63.4 %	(2.28)	81.9 %	(1.56)
stdev SR	43.3 %		37.4 %		36.8 %		33.2 %	
# perfect	7	(3)	4	(4)	12	(2)	23	(1)
# failure	10	(4)	4	(2)	4	(2)	4	(2)
Z-score	5.40		6.79		3.19			
p-value	0.0000		0.0000		0.0007			
Holm α	0.0250		0.0500		0.0167			
problem	VN BPSO error	(rank)	VN MBPSO error	(rank)	VN PBPSO error	(rank)	VN SBPSO error	(rank)
average error	0.609 %	(2.81)	0.613 %	(3.45)	0.510 %	(2.28)	**0.439 %**	(1.46)
stdev error	1.635 %		1.623 %		1.633 %		1.641 %	
average SR	56.6 %	(2.76)	48.6 %	(3.31)	64.8 %	(2.36)	82.7 %	(1.56)
stdev SR	41.1 %		35.3 %		36.8 %		32.6 %	
# perfect	9	(3)	4	(4)	12	(2)	25	(1)
# failure	6	(4)	4	(2)	4	(2)	4	(2)
Z-score	4.68		6.89		2.84			
p-value	0.0000		0.0000		0.0023			
Holm α	0.0250		0.0500		0.0167			

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.3 Detailed testing results per topology

A.3.1 Star topology

Table A.7: Details of the small MKP test results for the star topology. Bold face indicates statistically significant outperformance of one algorithm for that topology.

			GB BPSO		GB MBPSO		GB PBPSO		GB SBPSO	
problem	n	m	error	(rank)	error	(rank)	error	(rank)	error	(rank)
mknap1-1	6	10	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)
mknap1-2	10	10	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)
mknap1-3	15	10	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)
mknap1-6	39	5	0.262 %	(3)	0.600 %	(4)	0.261 %	(2)	0.070 %	(1)
mknap1-7	50	5	0.572 %	(3)	0.725 %	(4)	0.312 %	(2)	0.074 %	(1)
mknap2-1	60	30	0.228 %	(2)	1.164 %	(4)	0.242 %	(3)	0.021 %	(1)
mknap2-3	28	2	0.046 %	(2)	0.730 %	(4)	0.050 %	(3)	0 %	(1)
mknap2-5	28	2	1.811 %	(4)	1.463 %	(3)	1.000 %	(2)	0.114 %	(1)
mknap2-6	28	2	3.919 %	(3)	3.871 %	(2)	3.937 %	(4)	3.698 %	(1)
mknap2-7	28	2	0.768 %	(2)	1.374 %	(4)	0.853 %	(3)	0.347 %	(1)
mknap2-8	28	2	0.801 %	(2)	2.088 %	(4)	0.851 %	(3)	0.231 %	(1)
mknap2-9	105	2	0.349 %	(2)	0.950 %	(4)	0.367 %	(3)	0.189 %	(1)
mknap2-11	30	5	0.416 %	(3)	0.490 %	(4)	0.411 %	(2)	0.348 %	(1)
mknap2-12	30	5	0.083 %	(2)	0.201 %	(4)	0.168 %	(3)	0.004 %	(1)
mknap2-13	30	5	3.209 %	(2)	3.237 %	(4)	3.209 %	(2)	3.209 %	(2)
mknap2-14	30	5	0.030 %	(2)	0.133 %	(4)	0.082 %	(3)	0 %	(1)
mknap2-16	40	5	0.018 %	(2)	0.134 %	(4)	0.045 %	(3)	0 %	(1)
mknap2-18	40	5	10.44 %	(4)	9.471 %	(3)	9.420 %	(2)	9.407 %	(1)
mknap2-19	40	5	0 %	(1.5)	0.317 %	(4)	0.017 %	(3)	0 %	(1.5)
mknap2-21	50	5	0.042 %	(2)	0.135 %	(4)	0.046 %	(3)	0 %	(1)
mknap2-22	50	5	0.046 %	(2)	0.452 %	(4)	0.048 %	(3)	0 %	(1)
mknap2-23	50	5	0 %	(2)	0.012 %	(4)	0 %	(2)	0 %	(2)
mknap2-24	60	5	0.123 %	(2)	0.509 %	(4)	0.131 %	(3)	0.002 %	(1)
mknap2-25	60	5	0.016 %	(2)	0.351 %	(4)	0.032 %	(3)	0.001 %	(1)
mknap2-27	60	5	0.006 %	(3)	0.285 %	(4)	0 %	(1.5)	0 %	(1.5)
mknap2-29	70	5	0.255 %	(2)	1.243 %	(4)	0.464 %	(3)	0 %	(1)
mknap2-30	70	5	0.031 %	(2)	0.878 %	(4)	0.068 %	(3)	0 %	(1)
mknap2-31	70	5	0.363 %	(3)	0.623 %	(4)	0.350 %	(2)	0.004 %	(1)
mknap2-32	80	5	0.197 %	(3)	0.628 %	(4)	0.030 %	(2)	0 %	(1)
mknap2-34	80	5	0.254 %	(4)	0.198 %	(3)	0.064 %	(2)	0 %	(1)
mknap2-35	80	5	0.946 %	(3)	1.385 %	(4)	0.235 %	(2)	0 %	(1)
mknap2-36	90	5	0.867 %	(4)	0.621 %	(3)	0.116 %	(2)	0 %	(1)
mknap2-37	90	5	0.627 %	(3)	0.803 %	(4)	0.055 %	(2)	0 %	(1)
mknap2-38	90	5	1.864 %	(4)	1.255 %	(3)	0.326 %	(2)	0.014 %	(1)
mknap2-40	90	5	1.854 %	(4)	0.828 %	(3)	0.233 %	(2)	0.001 %	(1)
mknap2-42	34	4	1.668 %	(4)	0.751 %	(3)	0.235 %	(2)	0.005 %	(1)
mknap2-43	29	2	3.330 %	(4)	1.442 %	(3)	0.406 %	(2)	0 %	(1)
mknap2-44	20	10	3.331 %	(4)	1.880 %	(3)	0.269 %	(2)	0 %	(1)
mknap2-46	37	30	3.665 %	(4)	1.778 %	(3)	0.476 %	(2)	0 %	(1)
mknap2-47	28	4	2.259 %	(4)	0.571 %	(3)	0.295 %	(2)	0.007 %	(1)
average			1.117 %	(2.80)	1.089 %	(3.56)	0.628 %	(2.45)	**0.444 %**	(1.19)
# perfect			5	(2.5)	3	(4)	5	(2.5)	21	(1)
# failure			11	(4)	4	(2)	4	(2)	4	(2)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.3.2 Ring topology

Table A.8: Details of the small MKP test results for the ring topology. Bold face indicates statistically significant outperformance of one algorithm for that topology.

			LB BPSO		LB MBPSO		LB PBPSO		LB SBPSO	
problem	n	m	error	(rank)	error	(rank)	error	(rank)	error	(rank)
mknap1-1	6	10	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)
mknap1-2	10	10	0.127 %	(4)	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-3	15	10	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)
mknap1-6	39	5	0.121 %	(3)	0.296 %	(4)	0.119 %	(2)	0.116 %	(1)
mknap1-7	50	5	0.314 %	(3)	0.385 %	(4)	0.114 %	(2)	0.090 %	(1)
mknap2-1	60	30	0.358 %	(4)	0.249 %	(3)	0.029 %	(2)	0.001 %	(1)
mknap2-3	28	2	0 %	(2)	0.047 %	(4)	0 %	(2)	0 %	(2)
mknap2-5	28	2	0.336 %	(3)	0.443 %	(4)	0.145 %	(2)	0.043 %	(1)
mknap2-6	28	2	3.863 %	(4)	3.840 %	(2)	3.846 %	(3)	3.698 %	(1)
mknap2-7	28	2	0.503 %	(2)	0.849 %	(4)	0.644 %	(3)	0.286 %	(1)
mknap2-8	28	2	0.235 %	(2)	0.782 %	(4)	0.265 %	(3)	0.121 %	(1)
mknap2-9	105	2	0.191 %	(1)	0.494 %	(4)	0.207 %	(2)	0.225 %	(3)
mknap2-11	30	5	0.368 %	(2)	0.471 %	(4)	0.405 %	(3)	0.348 %	(1)
mknap2-12	30	5	0.162 %	(3)	0.126 %	(2)	0.236 %	(4)	0 %	(1)
mknap2-13	30	5	3.209 %	(2.5)	3.209 %	(2.5)	3.209 %	(2.5)	3.209 %	(2.5)
mknap2-14	30	5	0.038 %	(2)	0.110 %	(3)	0.228 %	(4)	0 %	(1)
mknap2-16	40	5	0.012 %	(2)	0.093 %	(4)	0.063 %	(3)	0 %	(1)
mknap2-18	40	5	9.449 %	(3)	9.466 %	(4)	9.408 %	(2)	9.407 %	(1)
mknap2-19	40	5	0 %	(2)	0.088 %	(4)	0 %	(2)	0 %	(2)
mknap2-21	50	5	0 %	(2)	0.018 %	(4)	0 %	(2)	0 %	(2)
mknap2-22	50	5	0 %	(2)	0.102 %	(4)	0 %	(2)	0 %	(2)
mknap2-23	50	5	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)
mknap2-24	60	5	0.007 %	(2)	0.118 %	(4)	0.015 %	(3)	0 %	(1)
mknap2-25	60	5	0.001 %	(2)	0.045 %	(4)	0.007 %	(3)	0 %	(1)
mknap2-27	60	5	0 %	(2)	0.027 %	(4)	0 %	(2)	0 %	(2)
mknap2-29	70	5	0.012 %	(2)	0.223 %	(4)	0.040 %	(3)	0 %	(1)
mknap2-30	70	5	0.007 %	(3)	0.139 %	(4)	0 %	(1.5)	0 %	(1.5)
mknap2-31	70	5	0.091 %	(4)	0.051 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-32	80	5	0.179 %	(3)	0.208 %	(4)	0.004 %	(2)	0 %	(1)
mknap2-34	80	5	0.034 %	(4)	0.032 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-35	80	5	0.666 %	(4)	0.395 %	(3)	0.079 %	(2)	0 %	(1)
mknap2-36	90	5	0.359 %	(4)	0.142 %	(3)	0.016 %	(2)	0 %	(1)
mknap2-37	90	5	0.467 %	(4)	0.357 %	(3)	0.038 %	(2)	0 %	(1)
mknap2-38	90	5	1.418 %	(4)	0.509 %	(3)	0.184 %	(2)	0.010 %	(1)
mknap2-40	90	5	0.998 %	(4)	0.292 %	(3)	0.181 %	(2)	0.001 %	(1)
mknap2-42	34	4	0.836 %	(4)	0.231 %	(3)	0.116 %	(2)	0.008 %	(1)
mknap2-43	29	2	2.542 %	(4)	0.392 %	(3)	0.323 %	(2)	0.013 %	(1)
mknap2-44	20	10	2.852 %	(4)	0.614 %	(3)	0.372 %	(2)	0 %	(1)
mknap2-46	37	30	2.666 %	(4)	0.435 %	(3)	0.328 %	(2)	0.002 %	(1)
mknap2-47	28	4	1.217 %	(4)	0.271 %	(3)	0.225 %	(2)	0.007 %	(1)
average			0.841 %	(2.95)	0.639 %	(3.35)	0.521 %	(2.31)	**0.440 %**	(1.39)
# perfect			7	(3)	4	(4)	12	(2)	23	(1)
# failure			10	(4)	4	(2)	4	(2)	4	(2)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.3.3 Von Neumann topology

Table A.9: Details of the small MKP test results for the Von Neumann topology. Bold face indicates statistically significant outperformance of one algorithm for that topology.

			VN BPSO		VN MBPSO		VN PBPSO		VN SBPSO	
problem	n	m	error	(rank)	error	(rank)	error	(rank)	error	(rank)
mknap1-1	6	10	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)
mknap1-2	10	10	0.212 %	(4)	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-3	15	10	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)
mknap1-6	39	5	0.093 %	(1.5)	0.278 %	(4)	0.093 %	(1.5)	0.104 %	(3)
mknap1-7	50	5	0.235 %	(3)	0.280 %	(4)	0.097 %	(2)	0.054 %	(1)
mknap2-1	60	30	0.087 %	(3)	0.285 %	(4)	0.030 %	(2)	0 %	(1)
mknap2-3	28	2	0 %	(2)	0.143 %	(4)	0 %	(2)	0 %	(2)
mknap2-5	28	2	0.308 %	(3)	0.335 %	(4)	0.095 %	(2)	0.054 %	(1)
mknap2-6	28	2	3.943 %	(4)	3.820 %	(3)	3.792 %	(2)	3.698 %	(1)
mknap2-7	28	2	0.647 %	(2)	0.799 %	(4)	0.689 %	(3)	0.278 %	(1)
mknap2-8	28	2	0.359 %	(3)	0.883 %	(4)	0.268 %	(2)	0.110 %	(1)
mknap2-9	105	2	0.206 %	(1)	0.453 %	(4)	0.225 %	(2)	0.247 %	(3)
mknap2-11	30	5	0.399 %	(2)	0.451 %	(4)	0.402 %	(3)	0.348 %	(1)
mknap2-12	30	5	0.157 %	(3)	0.141 %	(2)	0.198 %	(4)	0 %	(1)
mknap2-13	30	5	3.209 %	(2.5)	3.209 %	(2.5)	3.209 %	(2.5)	3.209 %	(2.5)
mknap2-14	30	5	0.038 %	(3)	0.031 %	(2)	0.116 %	(4)	0 %	(1)
mknap2-16	40	5	0.066 %	(2)	0.118 %	(4)	0.072 %	(3)	0 %	(1)
mknap2-18	40	5	9.409 %	(3)	9.465 %	(4)	9.408 %	(2)	9.407 %	(1)
mknap2-19	40	5	0 %	(2)	0.017 %	(4)	0 %	(2)	0 %	(2)
mknap2-21	50	5	0.002 %	(3)	0.025 %	(4)	0 %	(1.5)	0 %	(1.5)
mknap2-22	50	5	0 %	(2)	0.095 %	(4)	0 %	(2)	0 %	(2)
mknap2-23	50	5	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)	0 %	(2.5)
mknap2-24	60	5	0.009 %	(3)	0.157 %	(4)	0.006 %	(2)	0 %	(1)
mknap2-25	60	5	0.005 %	(2)	0.055 %	(4)	0.006 %	(3)	0 %	(1)
mknap2-27	60	5	0 %	(2)	0.035 %	(4)	0 %	(2)	0 %	(2)
mknap2-29	70	5	0.007 %	(2)	0.333 %	(4)	0.012 %	(3)	0 %	(1)
mknap2-30	70	5	0 %	(2)	0.140 %	(4)	0 %	(2)	0 %	(2)
mknap2-31	70	5	0 %	(2)	0.125 %	(4)	0 %	(2)	0 %	(2)
mknap2-32	80	5	0.030 %	(3)	0.133 %	(4)	0 %	(1.5)	0 %	(1.5)
mknap2-34	80	5	0.002 %	(3)	0.043 %	(4)	0 %	(1.5)	0 %	(1.5)
mknap2-35	80	5	0.144 %	(3)	0.263 %	(4)	0.029 %	(2)	0 %	(1)
mknap2-36	90	5	0.069 %	(3)	0.125 %	(4)	0.022 %	(2)	0 %	(1)
mknap2-37	90	5	0.165 %	(3)	0.279 %	(4)	0.048 %	(2)	0 %	(1)
mknap2-38	90	5	0.452 %	(4)	0.420 %	(3)	0.149 %	(2)	0.008 %	(1)
mknap2-40	90	5	0.351 %	(4)	0.208 %	(3)	0.180 %	(2)	0.001 %	(1)
mknap2-42	34	4	0.280 %	(4)	0.201 %	(3)	0.126 %	(2)	0.010 %	(1)
mknap2-43	29	2	0.883 %	(4)	0.340 %	(3)	0.311 %	(2)	0.017 %	(1)
mknap2-44	20	10	1.183 %	(4)	0.359 %	(3)	0.248 %	(2)	0 %	(1)
mknap2-46	37	30	0.985 %	(4)	0.290 %	(2)	0.351 %	(3)	0 %	(1)
mknap2-47	28	4	0.437 %	(4)	0.181 %	(2)	0.205 %	(3)	0.002 %	(1)
average			0.609 %	(2.81)	0.613 %	(3.45)	0.510 %	(2.28)	**0.439 %**	(1.46)
# perfect			9	(3)	4	(4)	12	(2)	25	(1)
# failure			6	(4)	4	(2)	4	(2)	4	(2)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.4 Summarized testing results per algorithm

Table A.10: Summary of the small MKP test results per algorithm. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

	GB BPSO		LB BPSO		VN BPSO	
Measure	error	(rank)	error	(rank)	error	(rank)
avg error	1.117 %	(2.65)	**0.841 %**	(1.80)	**0.609 %**	(1.55)
stdev error	1.913 %		1.716 %		1.635 %	
average SR	42.8 %	(2.45)	50.3 %	(2.03)	56.6 %	(1.53)
stdev SR	41.3 %		43.3 %		41.1 %	
# perfect	5	(3)	7	(2)	9	(1)
# failure	11	(3)	10	(2)	6	(1)
Z-score		4.92		1.12		
p-value		0.0000		0.1314		
Holm α		0.0500		0.0250		

	GB MBPSO		LB MBPSO		VN MBPSO	
Measure	error	(rank)	error	(rank)	error	(rank)
avg error	1.089 %	(2.93)	**0.639 %**	(1.65)	**0.613 %**	(1.43)
stdev error	1.592 %		1.620 %		1.623 %	
average SR	29.9 %	(2.78)	45.7 %	(1.68)	48.6 %	(1.55)
stdev SR	34.7 %		37.4 %		35.3 %	
# perfect	3	(3)	4	(1.5)	4	(1.5)
# failure	4	(2)	4	(2)	4	(2)
p-value		0.0000		0.1635		
Holm α		0.0500		0.0250		

	GB PBPSO		LB PBPSO		VN PBPSO	
Measure	error	(rank)	error	(rank)	error	(rank)
avg error	0.628 %	(2.68)	**0.521 %**	(1.78)	**0.510 %**	(1.55)
stdev error	1.625 %		1.634 %		1.633 %	
average SR	51.4 %	(2.4)	63.4 %	(1.9)	64.8 %	(1.7)
stdev SR	35.1 %		36.8 %		36.8 %	
# perfect	5	(3)	12	(1.5)	12	(1.5)
# failure	4	(2)	4	(2)	4	(2)
rank of rank		3		2		1
p-value		0.0000		0.1515		
Holm α		0.0500		0.0250		

	GB SBPSO		LB SBPSO		VN SBPSO	
Measure	error	(rank)	error	(rank)	error	(rank)
avg error	0.444 %	(2.13)	0.440 %	(2.01)	0.439 %	(1.86)
stdev error	1.640 %		1.641 %		1.641 %	
average SR	82.5 %	(2.09)	81.9 %	(2.04)	82.7 %	(1.88)
stdev SR	31.8 %		33.2 %		32.6 %	
# perfect	21	(3)	23	(2)	25	(1)
# failure	4	(2)	4	(2)	4	(2)
p-value		0.1131		0.2514		
Holm α		0.0500		0.0250		

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.5 Detailed testing results per algorithm

A.5.1 BPSO

Table A.11: Details of the small MKP test results for BPSO. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

			GB BPSO		LB BPSO		VN BPSO	
oblem	n	m	error	(rank)	error	(rank)	error	(rank)
mknap1-1	6	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-2	10	10	0 %	(1)	0.127 %	(2)	0.212 %	(3)
mknap1-3	15	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-6	39	5	0.262 %	(3)	0.121 %	(2)	0.093 %	(1)
mknap1-7	50	5	0.572 %	(3)	0.314 %	(2)	0.235 %	(1)
mknap2-1	60	30	0.228 %	(2)	0.358 %	(3)	0.087 %	(1)
mknap2-3	28	2	0.046 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-5	28	2	1.811 %	(3)	0.336 %	(2)	0.308 %	(1)
mknap2-6	28	2	3.919 %	(2)	3.863 %	(1)	3.943 %	(3)
mknap2-7	28	2	0.768 %	(3)	0.503 %	(1)	0.647 %	(2)
mknap2-8	28	2	0.801 %	(3)	0.235 %	(1)	0.359 %	(2)
mknap2-9	105	2	0.349 %	(3)	0.191 %	(1)	0.206 %	(2)
mknap2-11	30	5	0.416 %	(3)	0.368 %	(1)	0.399 %	(2)
mknap2-12	30	5	0.083 %	(1)	0.162 %	(3)	0.157 %	(2)
mknap2-13	30	5	3.209 %	(2)	3.209 %	(2)	3.209 %	(2)
mknap2-14	30	5	0.030 %	(1)	0.038 %	(2.5)	0.038 %	(2.5)
mknap2-16	40	5	0.018 %	(2)	0.012 %	(1)	0.066 %	(3)
mknap2-18	40	5	10.440 %	(3)	9.449 %	(2)	9.409 %	(1)
mknap2-19	40	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-21	50	5	0.042 %	(3)	0 %	(1)	0.002 %	(2)
mknap2-22	50	5	0.046 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-23	50	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-24	60	5	0.123 %	(3)	0.007 %	(1)	0.009 %	(2)
mknap2-25	60	5	0.016 %	(3)	0.001 %	(1)	0.005 %	(2)
mknap2-27	60	5	0.006 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-29	70	5	0.255 %	(3)	0.012 %	(2)	0.007 %	(1)
mknap2-30	70	5	0.031 %	(3)	0.007 %	(2)	0 %	(1)
mknap2-31	70	5	0.363 %	(3)	0.091 %	(2)	0 %	(1)
mknap2-32	80	5	0.197 %	(3)	0.179 %	(2)	0.030 %	(1)
mknap2-34	80	5	0.254 %	(3)	0.034 %	(2)	0.002 %	(1)
mknap2-35	80	5	0.946 %	(3)	0.666 %	(2)	0.144 %	(1)
mknap2-36	90	5	0.867 %	(3)	0.359 %	(2)	0.069 %	(1)
mknap2-37	90	5	0.627 %	(3)	0.467 %	(2)	0.165 %	(1)
mknap2-38	90	5	1.864 %	(3)	1.418 %	(2)	0.452 %	(1)
mknap2-40	90	5	1.854 %	(3)	0.998 %	(2)	0.351 %	(1)
mknap2-42	34	4	1.668 %	(3)	0.836 %	(2)	0.280 %	(1)
mknap2-43	29	2	3.330 %	(3)	2.542 %	(2)	0.883 %	(1)
mknap2-44	20	10	3.331 %	(3)	2.852 %	(2)	1.183 %	(1)
mknap2-46	37	30	3.665 %	(3)	2.666 %	(2)	0.985 %	(1)
mknap2-47	28	4	2.259 %	(3)	1.217 %	(2)	0.437 %	(1)
average			1.117 %	(2.65)	**0.841 %**	(1.80)	**0.609 %**	(1.55)
# perfect			5	(3)	7	(2)	9	(1)
# failure			11	(3)	10	(2)	6	(1)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.5.2 MBPSO

Table A.12: Details of the small MKP test results for MBPSO. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

			GB MBPSO		LB MBPSO		VN MBPSO	
problem	n	m	error	(rank)	error	(rank)	error	(rank)
mknap1-1	6	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-2	10	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-3	15	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-6	39	5	0.600 %	(3)	0.296 %	(2)	0.278 %	(1)
mknap1-7	50	5	0.725 %	(3)	0.385 %	(2)	0.280 %	(1)
mknap2-1	60	30	1.164 %	(3)	0.249 %	(1)	0.285 %	(2)
mknap2-3	28	2	0.730 %	(3)	0.047 %	(1)	0.143 %	(2)
mknap2-5	28	2	1.463 %	(3)	0.443 %	(2)	0.335 %	(1)
mknap2-6	28	2	3.871 %	(3)	3.840 %	(2)	3.820 %	(1)
mknap2-7	28	2	1.374 %	(3)	0.849 %	(2)	0.799 %	(1)
mknap2-8	28	2	2.088 %	(3)	0.782 %	(1)	0.883 %	(2)
mknap2-9	105	2	0.950 %	(3)	0.494 %	(2)	0.453 %	(1)
mknap2-11	30	5	0.490 %	(3)	0.471 %	(2)	0.451 %	(1)
mknap2-12	30	5	0.201 %	(3)	0.126 %	(1)	0.141 %	(2)
mknap2-13	30	5	3.237 %	(3)	3.209 %	(1.5)	3.209 %	(1.5)
mknap2-14	30	5	0.133 %	(3)	0.110 %	(2)	0.031 %	(1)
mknap2-16	40	5	0.134 %	(3)	0.093 %	(1)	0.118 %	(2)
mknap2-18	40	5	9.471 %	(3)	9.466 %	(2)	9.465 %	(1)
mknap2-19	40	5	0.317 %	(3)	0.088 %	(2)	0.017 %	(1)
mknap2-21	50	5	0.135 %	(3)	0.018 %	(1)	0.025 %	(2)
mknap2-22	50	5	0.452 %	(3)	0.102 %	(2)	0.095 %	(1)
mknap2-23	50	5	0.012 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-24	60	5	0.509 %	(3)	0.118 %	(1)	0.157 %	(2)
mknap2-25	60	5	0.351 %	(3)	0.045 %	(1)	0.055 %	(2)
mknap2-27	60	5	0.285 %	(3)	0.027 %	(1)	0.035 %	(2)
mknap2-29	70	5	1.243 %	(3)	0.223 %	(1)	0.333 %	(2)
mknap2-30	70	5	0.878 %	(3)	0.139 %	(1)	0.140 %	(2)
mknap2-31	70	5	0.623 %	(3)	0.051 %	(1)	0.125 %	(2)
mknap2-32	80	5	0.628 %	(3)	0.208 %	(2)	0.133 %	(1)
mknap2-34	80	5	0.198 %	(3)	0.032 %	(1)	0.043 %	(2)
mknap2-35	80	5	1.385 %	(3)	0.395 %	(2)	0.263 %	(1)
mknap2-36	90	5	0.621 %	(3)	0.142 %	(2)	0.125 %	(1)
mknap2-37	90	5	0.803 %	(3)	0.357 %	(2)	0.279 %	(1)
mknap2-38	90	5	1.255 %	(3)	0.509 %	(2)	0.420 %	(1)
mknap2-40	90	5	0.828 %	(3)	0.292 %	(2)	0.208 %	(1)
mknap2-42	34	4	0.751 %	(3)	0.231 %	(2)	0.201 %	(1)
mknap2-43	29	2	1.442 %	(3)	0.392 %	(2)	0.340 %	(1)
mknap2-44	20	10	1.880 %	(3)	0.614 %	(2)	0.359 %	(1)
mknap2-46	37	30	1.778 %	(3)	0.435 %	(2)	0.290 %	(1)
mknap2-47	28	4	0.571 %	(3)	0.271 %	(2)	0.181 %	(1)
average			1.089 %	(2.93)	**0.639 %**	(1.65)	**0.613 %**	(1.43)
# perfect			3	(3)	4	(1.5)	4	(1.5)
# failure			4	(2)	4	(2)	4	(2)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.5.3 PBPSO

Table A.13: Details of the small MKP test results for PBPSO. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

			GB PBPSO		LB PBPSO		VN PBPSO	
problem	n	m	error	(rank)	error	(rank)	error	(rank)
mknap1-1	6	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-2	10	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-3	15	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-6	39	5	0.261 %	(3)	0.119 %	(2)	0.093 %	(1)
mknap1-7	50	5	0.312 %	(3)	0.114 %	(2)	0.097 %	(1)
mknap2-1	60	30	0.242 %	(3)	0.029 %	(1)	0.030 %	(2)
mknap2-3	28	2	0.050 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-5	28	2	1.000 %	(3)	0.145 %	(2)	0.095 %	(1)
mknap2-6	28	2	3.937 %	(3)	3.846 %	(2)	3.792 %	(1)
mknap2-7	28	2	0.853 %	(3)	0.644 %	(1)	0.689 %	(2)
mknap2-8	28	2	0.851 %	(3)	0.265 %	(1)	0.268 %	(2)
mknap2-9	105	2	0.367 %	(3)	0.207 %	(1)	0.225 %	(2)
mknap2-11	30	5	0.411 %	(3)	0.405 %	(2)	0.402 %	(1)
mknap2-12	30	5	0.168 %	(1)	0.236 %	(3)	0.198 %	(2)
mknap2-13	30	5	3.209 %	(2)	3.209 %	(2)	3.209 %	(2)
mknap2-14	30	5	0.082 %	(1)	0.228 %	(3)	0.116 %	(2)
mknap2-16	40	5	0.045 %	(1)	0.063 %	(2)	0.072 %	(3)
mknap2-18	40	5	9.420 %	(3)	9.408 %	(1.5)	9.408 %	(1.5)
mknap2-19	40	5	0.017 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-21	50	5	0.046 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-22	50	5	0.048 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-23	50	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-24	60	5	0.131 %	(3)	0.015 %	(2)	0.006 %	(1)
mknap2-25	60	5	0.032 %	(3)	0.007 %	(2)	0.006 %	(1)
mknap2-27	60	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-29	70	5	0.464 %	(3)	0.040 %	(2)	0.012 %	(1)
mknap2-30	70	5	0.068 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-31	70	5	0.350 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-32	80	5	0.030 %	(3)	0.004 %	(2)	0 %	(1)
mknap2-34	80	5	0.064 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-35	80	5	0.235 %	(3)	0.079 %	(2)	0.029 %	(1)
mknap2-36	90	5	0.116 %	(3)	0.016 %	(1)	0.022 %	(2)
mknap2-37	90	5	0.055 %	(3)	0.038 %	(1)	0.048 %	(2)
mknap2-38	90	5	0.326 %	(3)	0.184 %	(2)	0.149 %	(1)
mknap2-40	90	5	0.233 %	(3)	0.181 %	(2)	0.180 %	(1)
mknap2-42	34	4	0.235 %	(3)	0.116 %	(1)	0.126 %	(2)
mknap2-43	29	2	0.406 %	(3)	0.323 %	(2)	0.311 %	(1)
mknap2-44	20	10	0.269 %	(2)	0.372 %	(3)	0.248 %	(1)
mknap2-46	37	30	0.476 %	(3)	0.328 %	(1)	0.351 %	(2)
mknap2-47	28	4	0.295 %	(3)	0.225 %	(2)	0.205 %	(1)
average			0.628 %	(2.68)	**0.521 %**	(1.78)	**0.510 %**	(1.55)
# perfect			5	(3)	12	(1.5)	12	(1.5)
# failure			4	(2)	4	(2)	4	(2)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A.5.4 SBPSO

Table A.14: Details of the small MKP test results for SBPSO. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

			GB SBPSO		LB SBPSO		VN SBPSO	
problem	n	m	error	(rank)	error	(rank)	error	(rank)
mknap1-1	6	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-2	10	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-3	15	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap1-6	39	5	0.070 %	(1)	0.116 %	(3)	0.104 %	(2)
mknap1-7	50	5	0.074 %	(2)	0.090 %	(3)	0.054 %	(1)
mknap2-1	60	30	0.021 %	(3)	0.001 %	(2)	0 %	(1)
mknap2-3	28	2	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-5	28	2	0.114 %	(3)	0.043 %	(1)	0.054 %	(2)
mknap2-6	28	2	3.698 %	(2)	3.698 %	(2)	3.698 %	(2)
mknap2-7	28	2	0.347 %	(3)	0.286 %	(2)	0.278 %	(1)
mknap2-8	28	2	0.231 %	(3)	0.121 %	(2)	0.110 %	(1)
mknap2-9	105	2	0.189 %	(1)	0.225 %	(2)	0.247 %	(3)
mknap2-11	30	5	0.348 %	(2)	0.348 %	(2)	0.348 %	(2)
mknap2-12	30	5	0.004 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-13	30	5	3.209 %	(2)	3.209 %	(2)	3.209 %	(2)
mknap2-14	30	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-16	40	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-18	40	5	9.407 %	(2)	9.407 %	(2)	9.407 %	(2)
mknap2-19	40	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-21	50	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-22	50	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-23	50	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-24	60	5	0.002 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-25	60	5	0.001 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-27	60	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-29	70	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-30	70	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-31	70	5	0.004 %	(3)	0 %	(1.5)	0 %	(1.5)
mknap2-32	80	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-34	80	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-35	80	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-36	90	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-37	90	5	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-38	90	5	0.014 %	(3)	0.010 %	(2)	0.008 %	(1)
mknap2-40	90	5	0.001 %	(2)	0.001 %	(2)	0.001 %	(2)
mknap2-42	34	4	0.005 %	(1)	0.008 %	(2)	0.010 %	(3)
mknap2-43	29	2	0 %	(1)	0.013 %	(2)	0.017 %	(3)
mknap2-44	20	10	0 %	(2)	0 %	(2)	0 %	(2)
mknap2-46	37	30	0 %	(1.5)	0.002 %	(3)	0 %	(1.5)
mknap2-47	28	4	0.007 %	(2.5)	0.007 %	(2.5)	0.002 %	(1)
average			0.444 %	(2.13)	0.440 %	(2.01)	0.439 %	(1.86)
# perfect			21	(3)	23	(2)	25	(1)
# failure			4	(2)	4	(2)	4	(2)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Appendix B

Detailed results for large MKPs

This appendix contains the detailed results of the experiments run on the large MKPs. In total 270 such problems were considered, all of which are known from the literature and are described by Chu and Beasley [18]. All problems used are available on-line at the Operations Research library[1]. The 270 problems used can be found in the files “mknapcb1.txt” until “mknapcb9.txt.txt” and are labeled with that filename and the order in which they can be found the files: the problem labeled “mknapcb2-3” is thus the third problem found in the file “mknapcb2.txt”.

Four different algorithms as previously (BPSO, MBPSO, PBPSO, SBPSO) were compared and each algorithm was run using three different topologies for the particle swarm: star (GB), ring (LB), and Von Neumann (VN). Each combination of algorithm and topology was first tuned on the tuning set of 27 problems. The tuned combinations of algorithm and topology were then applied on the test set of 243 problems, where 30 independent runs were simulated for each of these 12 combinations. The exact experimental set-up for both the tuning and the testing process is described in detail in sections 5.2 and 5.3.

This appendix contains detailed results of both the tuning experiments as well as the testing results based on the tuned algorithm topology pairs. An overview of all result tables for the large MKP in this appendix can be found in table B.1.

The detailed results for the tuning experiments are shown organized by algorithm in four tables (B.2, B.3, B.4, B.5). Shown in all result tables for tuning problems is the average error across the independent runs versus the optimum of the LP relaxation problem followed by a rank shown between brackets. In case of the results tables for the tuning group the rank shown is the rank of the chosen best parameter combination in the set of 128 parameter combinations for the problem named on that line. This ranking indicates how well the chosen parameter combination is suited to the problem listed compared to the other parameter combinations investigated.

The result tables for the experiments on the tuned algorithm topology combinations fall into two categories: inter algorithm comparisons and intra algorithm comparisons. The inter algorithm comparisons show the results of the four algorithms for one of the three topologies used, resulting in three tables (B.7, B.8, B.9). These results are summarized in table B.6. The intra algorithm comparisons each show the

[1]`http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table B.1: Guide to tables in Appendix B

Group	Type of comparison	Description	Table	Statistically best
tuning	intra algorithm	BPSO	B.2	*not applicable*
tuning	intra algorithm	MBPSO	B.3	*not applicable*
tuning	intra algorithm	PBPSO	B.4	*not applicable*
tuning	intra algorithm	SBPSO	B.5	*not applicable*
testing	inter algorithm	Summary	B.6	SBPSO
testing	intra algorithm	Summary	B.10	Star & Von Neumann
testing	inter algorithm	Star topology	B.7	GB SBPSO
testing	inter algorithm	Ring topology	B.8	LB SBPSO
testing	inter algorithm	Von Neumann topology	B.9	VN SBPSO
testing	intra algorithm	BPSO	B.11	GB BPSO
testing	intra algorithm	MBPSO	B.12	VN MBPSO
testing	intra algorithm	PBPSO	B.13	VN PBPSO
testing	intra algorithm	SBPSO	B.14	GB SBPSO

results of all three topologies for one of the four algorithms used, resulting in four tables (B.11, B.12, B.13, B.14). These results are summarized in table B.10.

Shown in all result tables for tuning problems is the average error across the independent runs versus the optimum of the LP relaxation problem followed by a rank shown between brackets. In case of the results tables for the testing group the rank shown is the relative performance (based on average error) of that algorithm topology combination versus the other combinations in the same table. This rank is determined for each problem separately and the number on a given line is the average rank over all problems summarized on that line. These ranks thus indicate the relative performance of the algorithm topology combination compared to the other pairs in the same table and are used to determine possible statistically significant outperformance.

The overview table B.1 also lists the best topology or algorithm that resulted from the comparisons made using testing results. The best in this case is meant as statistically significant outperformance ($\alpha = 0.05$) according to a Iman-Davenport test and further Nemenyi-tests with Holm-adjusted α. For a detailed description of this statistical test see appendix E.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.1 Detailed tuning results per algorithm

B.1.1 BPSO

Table B.2: Details of the large MKP tuning results for BPSO.

				GB BPSO		LB BPSO		VN BPSO	
problem	n	m	α	error	(rank)	error	(rank)	error	(rank)
mknapcb1-6	100	5	0.25	3.38 %	(1)	7.05 %	(1)	7.92 %	(3)
mknapcb1-17	100	5	0.50	1.84 %	(1)	3.26 %	(1)	3.03 %	(1)
mknapcb1-27	100	5	0.75	0.83 %	(1)	1.36 %	(1)	1.30 %	(1)
mknapcb2-7	100	10	0.25	10.77 %	(2)	15.63 %	(5)	15.90 %	(5)
mknapcb2-11	100	10	0.50	4.20 %	(1)	6.54 %	(1)	6.70 %	(1)
mknapcb2-22	100	10	0.75	2.30 %	(1)	3.66 %	(1)	3.64 %	(1)
mknapcb3-3	100	30	0.25	18.49 %	(9)	20.09 %	(9)	20.12 %	(10)
mknapcb3-20	100	30	0.50	7.34 %	(1)	9.21 %	(1)	9.24 %	(1)
mknapcb3-24	100	30	0.75	3.41 %	(1)	4.93 %	(1)	4.91 %	(1)
mknapcb4-3	250	5	0.25	5.74 %	(2)	9.12 %	(3)	8.98 %	(3)
mknapcb4-12	250	5	0.50	2.30 %	(1)	3.54 %	(1)	3.72 %	(1)
mknapcb4-27	250	5	0.75	1.47 %	(1)	1.86 %	(1)	1.69 %	(1)
mknapcb5-7	250	10	0.25	13.01 %	(4)	16.21 %	(5)	16.07 %	(5)
mknapcb5-20	250	10	0.50	5.54 %	(1)	8.23 %	(1)	8.04 %	(1)
mknapcb5-21	250	10	0.75	2.43 %	(1)	3.78 %	(1)	3.86 %	(1)
mknapcb6-7	250	30	0.25	20.26 %	(11)	21.33 %	(10)	21.28 %	(8)
mknapcb6-16	250	30	0.50	8.71 %	(1)	10.34 %	(1)	10.58 %	(1)
mknapcb6-23	250	30	0.75	3.63 %	(1)	5.04 %	(1)	5.01 %	(1)
mknapcb7-1	500	5	0.25	6.37 %	(1)	10.24 %	(3)	10.64 %	(3)
mknapcb7-19	500	5	0.50	2.79 %	(1)	4.44 %	(1)	4.50 %	(1)
mknapcb7-30	500	5	0.75	1.75 %	(1)	2.34 %	(1)	2.29 %	(1)
mknapcb8-10	500	10	0.25	15.32 %	(4)	17.54 %	(5)	18.02 %	(5)
mknapcb8-16	500	10	0.50	5.98 %	(1)	8.37 %	(1)	8.20 %	(1)
mknapcb8-26	500	10	0.75	2.89 %	(1)	4.23 %	(1)	4.12 %	(1)
mknapcb9-8	500	30	0.25	20.59 %	(9)	21.60 %	(8)	21.87 %	(10)
mknapcb9-18	500	30	0.50	9.41 %	(1)	10.90 %	(1)	10.94 %	(1)
mknapcb9-26	500	30	0.75	4.15 %	(1)	5.66 %	(1)	5.61 %	(1)
average				6.85 %	(2.3)	8.76 %	(2.5)	8.82 %	(2.6)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.1.2 MBPSO

Table B.3: Details of the large MKP tuning results for MBPSO.

				GB MBPSO		LB MBPSO		VN MBPSO	
problem	n	m	α	error	(rank)	error	(rank)	error	(rank)
mknapcb1-6	100	5	0.25	5.80 %	(11)	4.23 %	(13)	3.27 %	(4)
mknapcb1-17	100	5	0.50	2.83 %	(4)	1.70 %	(8)	1.55 %	(4)
mknapcb1-27	100	5	0.75	1.48 %	(9)	0.92 %	(6)	0.79 %	(4)
mknapcb2-7	100	10	0.25	7.95 %	(4)	4.20 %	(2)	3.95 %	(2)
mknapcb2-11	100	10	0.50	4.75 %	(8)	2.27 %	(4)	2.10 %	(5)
mknapcb2-22	100	10	0.75	2.23 %	(2)	1.12 %	(1)	0.94 %	(1)
mknapcb3-3	100	30	0.25	9.98 %	(6)	6.12 %	(5)	6.03 %	(4)
mknapcb3-20	100	30	0.50	6.25 %	(3)	3.20 %	(1)	3.29 %	(2)
mknapcb3-24	100	30	0.75	3.04 %	(2)	1.64 %	(3)	1.55 %	(1)
mknapcb4-3	250	5	0.25	6.37 %	(4)	4.80 %	(7)	4.43 %	(3)
mknapcb4-12	250	5	0.50	3.40 %	(8)	2.14 %	(2)	1.94 %	(3)
mknapcb4-27	250	5	0.75	2.30 %	(23)	1.44 %	(4)	1.28 %	(2)
mknapcb5-7	250	10	0.25	9.03 %	(12)	6.17 %	(7)	5.10 %	(2)
mknapcb5-20	250	10	0.50	5.19 %	(2)	2.87 %	(3)	2.38 %	(1)
mknapcb5-21	250	10	0.75	2.54 %	(1)	1.50 %	(2)	1.37 %	(6)
mknapcb6-7	250	30	0.25	11.08 %	(6)	7.75 %	(5)	6.67 %	(3)
mknapcb6-16	250	30	0.50	7.64 %	(3)	4.78 %	(2)	4.34 %	(2)
mknapcb6-23	250	30	0.75	3.72 %	(7)	2.09 %	(1)	2.16 %	(4)
mknapcb7-1	500	5	0.25	7.62 %	(14)	6.28 %	(12)	5.59 %	(4)
mknapcb7-19	500	5	0.50	3.67 %	(2)	2.86 %	(4)	2.75 %	(5)
mknapcb7-30	500	5	0.75	2.20 %	(1)	1.95 %	(5)	1.73 %	(4)
mknapcb8-10	500	10	0.25	8.82 %	(3)	6.56 %	(4)	6.18 %	(3)
mknapcb8-16	500	10	0.50	5.98 %	(11)	3.57 %	(3)	3.48 %	(5)
mknapcb8-26	500	10	0.75	3.00 %	(8)	1.88 %	(1)	1.81 %	(3)
mknapcb9-8	500	30	0.25	11.70 %	(13)	7.98 %	(2)	7.53 %	(3)
mknapcb9-18	500	30	0.50	7.70 %	(1)	5.55 %	(1)	4.83 %	(2)
mknapcb9-26	500	30	0.75	4.09 %	(8)	2.87 %	(4)	2.40 %	(3)
average				5.57 %	(6.5)	3.65 %	(4.1)	3.31 %	(3.1)

© University of Pretoria

B.1.3 PBPSO

Table B.4: Details of the large MKP tuning results for PBPSO.

				GB PBPSO		LB PBPSO		VN PBPSO	
problem	n	m	α	error	(rank)	error	(rank)	error	(rank)
mknapcb1-6	100	5	0.25	2.68 %	(1)	2.28 %	(2)	1.57 %	(1)
mknapcb1-17	100	5	0.50	1.76 %	(6)	0.95 %	(2)	0.72 %	(1)
mknapcb1-27	100	5	0.75	0.88 %	(3)	0.63 %	(3)	0.52 %	(2)
mknapcb2-7	100	10	0.25	3.33 %	(1)	4.28 %	(1)	4.00 %	(1)
mknapcb2-11	100	10	0.50	1.40 %	(1)	2.26 %	(1)	1.86 %	(1)
mknapcb2-22	100	10	0.75	0.92 %	(1)	1.21 %	(2)	1.06 %	(2)
mknapcb3-3	100	30	0.25	5.23 %	(2)	7.86 %	(1)	6.99 %	(1)
mknapcb3-20	100	30	0.50	2.65 %	(1)	4.55 %	(1)	4.02 %	(1)
mknapcb3-24	100	30	0.75	1.59 %	(1)	2.83 %	(2)	2.63 %	(1)
mknapcb4-3	250	5	0.25	4.71 %	(7)	3.17 %	(2)	2.82 %	(2)
mknapcb4-12	250	5	0.50	2.37 %	(8)	1.44 %	(1)	1.25 %	(1)
mknapcb4-27	250	5	0.75	1.36 %	(3)	1.10 %	(2)	0.88 %	(1)
mknapcb5-7	250	10	0.25	4.38 %	(2)	4.86 %	(1)	4.82 %	(1)
mknapcb5-20	250	10	0.50	2.03 %	(2)	2.37 %	(1)	2.08 %	(1)
mknapcb5-21	250	10	0.75	1.24 %	(1)	1.44 %	(1)	1.18 %	(1)
mknapcb6-7	250	30	0.25	5.84 %	(2)	8.37 %	(1)	7.67 %	(1)
mknapcb6-16	250	30	0.50	3.47 %	(2)	5.06 %	(1)	4.29 %	(1)
mknapcb6-23	250	30	0.75	1.89 %	(1)	3.02 %	(2)	2.65 %	(1)
mknapcb7-1	500	5	0.25	5.76 %	(7)	4.42 %	(2)	4.27 %	(2)
mknapcb7-19	500	5	0.50	2.65 %	(2)	1.86 %	(1)	1.78 %	(2)
mknapcb7-30	500	5	0.75	1.92 %	(5.5)	1.36 %	(2)	1.24 %	(1)
mknapcb8-10	500	10	0.25	4.93 %	(2)	5.59 %	(1)	5.17 %	(1)
mknapcb8-16	500	10	0.50	2.51 %	(1)	3.17 %	(1)	2.80 %	(1)
mknapcb8-26	500	10	0.75	1.62 %	(1)	1.69 %	(1)	1.47 %	(1)
mknapcb9-8	500	30	0.25	6.41 %	(2)	8.73 %	(1)	8.44 %	(1)
mknapcb9-18	500	30	0.50	4.08 %	(2)	5.54 %	(1)	4.95 %	(1)
mknapcb9-26	500	30	0.75	2.30 %	(1)	3.21 %	(1)	3.07 %	(1)
average				2.96 %	(2.5)	3.45 %	(1.4)	3.12 %	(1.2)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.1.4 SBPSO

Table B.5: Details of the large MKP tuning results for SBPSO.

				GB SBPSO		LB SBPSO		VN SBPSO	
problem	n	m	α	error	(rank)	error	(rank)	error	(rank)
mknapcb1-6	100	5	0.25	2.63 %	(6)	2.80 %	(2)	2.75 %	(3)
mknapcb1-17	100	5	0.50	0.89 %	(1)	1.34 %	(2)	1.04 %	(1)
mknapcb1-27	100	5	0.75	0.54 %	(4)	0.71 %	(3)	0.59 %	(1)
mknapcb2-7	100	10	0.25	2.51 %	(4)	3.63 %	(3)	3.44 %	(2)
mknapcb2-11	100	10	0.50	1.04 %	(1)	1.60 %	(2)	1.59 %	(2)
mknapcb2-22	100	10	0.75	0.42 %	(1)	0.82 %	(2)	0.82 %	(2)
mknapcb3-3	100	30	0.25	2.98 %	(3)	4.12 %	(2)	3.61 %	(2)
mknapcb3-20	100	30	0.50	1.37 %	(1)	2.13 %	(2)	1.80 %	(2)
mknapcb3-24	100	30	0.75	0.58 %	(1)	1.03 %	(1)	0.84 %	(1)
mknapcb4-3	250	5	0.25	3.30 %	(4)	3.82 %	(3)	3.53 %	(2)
mknapcb4-12	250	5	0.50	1.34 %	(1)	1.77 %	(3)	1.47 %	(1)
mknapcb4-27	250	5	0.75	0.86 %	(1)	1.09 %	(1)	1.08 %	(2)
mknapcb5-7	250	10	0.25	3.19 %	(2)	4.12 %	(1)	3.86 %	(2)
mknapcb5-20	250	10	0.50	1.15 %	(1)	2.06 %	(2)	1.83 %	(2)
mknapcb5-21	250	10	0.75	0.60 %	(1)	1.04 %	(6)	0.92 %	(3)
mknapcb6-7	250	30	0.25	3.36 %	(1)	4.69 %	(3)	4.00 %	(2)
mknapcb6-16	250	30	0.50	1.41 %	(1)	2.44 %	(2)	2.26 %	(2)
mknapcb6-23	250	30	0.75	0.73 %	(1)	1.36 %	(5)	1.14 %	(3)
mknapcb7-1	500	5	0.25	4.67 %	(4)	4.62 %	(1)	4.27 %	(1)
mknapcb7-19	500	5	0.50	1.80 %	(1)	2.27 %	(4)	2.16 %	(4)
mknapcb7-30	500	5	0.75	1.21 %	(1)	1.44 %	(4)	1.29 %	(1)
mknapcb8-10	500	10	0.25	3.49 %	(1)	4.33 %	(1)	4.35 %	(2)
mknapcb8-16	500	10	0.50	1.66 %	(1)	2.37 %	(3)	2.21 %	(3)
mknapcb8-26	500	10	0.75	0.79 %	(1)	1.23 %	(6)	1.04 %	(3)
mknapcb9-8	500	30	0.25	3.27 %	(1)	4.73 %	(2)	4.37 %	(2)
mknapcb9-18	500	30	0.50	1.62 %	(1)	2.79 %	(4)	2.54 %	(4)
mknapcb9-26	500	30	0.75	0.87 %	(1)	1.37 %	(5)	1.25 %	(3)
average				1.79 %	(1.7)	2.44 %	(2.8)	2.22 %	(2.1)

© University of Pretoria

B.2 Summarized testing results per topology

Table B.6: Summary of the large MKP test results per topology. Bold face indicates statistically significant outperformance of one algorithm for that topology.

		GB BPSO		GB MBPSO		GB PBPSO		GB SBPSO	
Measure		error	(rank)	error	(rank)	error	(rank)	error	(rank)
average error		4.679 %	(2.909)	5.619 %	(3.885)	3.250 %	(2.206)	**1.740 %**	(1.000)
stdev error		3.468 %		2.723 %		1.718 %		1.170 %	
n	100	3.831 %	(2.877)	5.160 %	(3.889)	2.568 %	(2.235)	**1.260 %**	(1.000)
n	250	4.679 %	(2.877)	5.663 %	(3.889)	3.286 %	(2.235)	**1.758 %**	(1.000)
n	500	5.526 %	(2.975)	6.034 %	(3.877)	3.896 %	(2.148)	**2.201 %**	(1.000)
m	5	3.037 %	(2.383)	4.354 %	(4.000)	3.134 %	(2.617)	**1.875 %**	(1.000)
m	10	3.942 %	(3.012)	5.521 %	(3.988)	2.763 %	(2.000)	**1.553 %**	(1.000)
m	30	7.057 %	(3.333)	6.983 %	(3.667)	3.853 %	(2.000)	**1.791 %**	(1.000)
α	0.25	8.253 %	(3.122)	8.664 %	(3.659)	5.264 %	(2.220)	**3.141 %**	(1.000)
α	0.50	3.751 %	(2.831)	5.344 %	(4.000)	2.799 %	(2.169)	**1.355 %**	(1.000)
α	0.75	1.909 %	(2.769)	2.712 %	(4.000)	1.613 %	(2.231)	**0.676 %**	(1.000)
		LB BPSO		LB MBPSO		LB PBPSO		LB SBPSO	
Measure		error	(rank)	error	(rank)	error	(rank)	error	(rank)
average error		7.006 %	(3.737)	3.922 %	(2.959)	3.650 %	(1.971)	**2.292 %**	(1.333)
stdev error		5.037 %		2.059 %		2.591 %		1.331 %	
n	100	6.348 %	(3.778)	3.044 %	(2.852)	3.101 %	(2.037)	**1.767 %**	(1.333)
n	250	6.951 %	(3.753)	3.917 %	(2.938)	3.626 %	(1.975)	**2.366 %**	(1.333)
n	500	7.719 %	(3.679)	4.805 %	(3.086)	4.221 %	(1.901)	**2.743 %**	(1.333)
m	5	3.091 %	(3.210)	3.289 %	(3.790)	**1.994 %**	(1.000)	2.334 %	(2.000)
m	10	7.520 %	(4.000)	3.654 %	(2.963)	3.112 %	(2.037)	**2.075 %**	(1.000)
m	30	10.407 %	(4.000)	4.824 %	(2.123)	5.842 %	(2.877)	**2.468 %**	(1.000)
α	0.25	11.961 %	(3.829)	6.185 %	(2.817)	6.059 %	(2.024)	**3.893 %**	(1.329)
α	0.50	5.817 %	(3.723)	3.608 %	(3.060)	3.066 %	(1.892)	**1.957 %**	(1.325)
α	0.75	3.063 %	(3.654)	1.878 %	(3.000)	1.738 %	(2.000)	**0.966 %**	(1.346)
		VN BPSO		VN MBPSO		VN PBPSO		VN SBPSO	
Measure		error	(rank)	error	(rank)	error	(rank)	error	(rank)
average error		6.973 %	(3.823)	3.403 %	(2.811)	3.348 %	(2.025)	**2.249 %**	(1.342)
stdev error		5.039 %		1.742 %		2.533 %		1.275 %	
n	100	6.291 %	(3.815)	2.647 %	(2.790)	2.762 %	(2.049)	**1.772 %**	(1.346)
n	250	6.920 %	(3.864)	3.418 %	(2.765)	3.330 %	(2.037)	**2.294 %**	(1.333)
n	500	7.707 %	(3.790)	4.145 %	(2.877)	3.954 %	(1.988)	**2.680 %**	(1.346)
m	5	3.076 %	(3.469)	2.980 %	(3.506)	**1.783 %**	(1.000)	2.433 %	(2.025)
m	10	7.465 %	(4.000)	3.191 %	(2.914)	2.847 %	(2.086)	**2.046 %**	(1.000)
m	30	10.377 %	(4.000)	4.039 %	(2.012)	5.416 %	(2.988)	**2.266 %**	(1.000)
α	0.25	11.943 %	(3.976)	5.382 %	(2.610)	5.739 %	(2.085)	**3.789 %**	(1.329)
α	0.50	5.775 %	(3.855)	3.089 %	(2.819)	2.693 %	(2.000)	**1.917 %**	(1.325)
α	0.75	3.023 %	(3.628)	1.658 %	(3.013)	1.533 %	(1.987)	**0.981 %**	(1.372)

Note that the results shown in table B.6 are repeated results previously shown in tables 5.16, 5.17, and 5.18, but the *Z*-scores, *p*-values, and Holm α's have been left out due to lack of space.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.3 Detailed test results per topology

B.3.1 Star topology

Table B.7: Details of the large MKP test results for the star topology. Bold face indicates statistically significant outperformance of one algorithm for that topology.

				GB BPSO		GB MBPSO		GB PBPSO		GB SBPSO	
problem	n	m	α	error	(rank)	error	(rank)	error	(rank)	error	(rank)
average	100	5	0.25	3.772 %	(2.222)	6.071 %	(4)	3.986 %	(2.778)	1.951 %	(1)
average	100	5	0.50	1.835 %	(2.333)	3.392 %	(4)	1.959 %	(2.667)	0.873 %	(1)
average	100	5	0.75	1.155 %	(2.333)	1.785 %	(4)	1.212 %	(2.667)	0.505 %	(1)
average	100	10	0.25	5.334 %	(3)	8.291 %	(4)	3.631 %	(2)	2.181 %	(1)
average	100	10	0.50	2.384 %	(3)	4.605 %	(4)	1.851 %	(2)	0.853 %	(1)
average	100	10	0.75	1.217 %	(3)	2.371 %	(4)	1.053 %	(2)	0.399 %	(1)
average	100	30	0.25	11.396 %	(4)	10.276 %	(3)	4.928 %	(2)	2.773 %	(1)
average	100	30	0.50	4.786 %	(3)	6.232 %	(4)	2.748 %	(2)	1.195 %	(1)
average	100	30	0.75	2.327 %	(3)	3.069 %	(4)	1.621 %	(2)	0.540 %	(1)
average	250	5	0.25	5.182 %	(2.333)	7.140 %	(4)	5.219 %	(2.667)	3.294 %	(1)
average	250	5	0.50	2.680 %	(2.444)	4.158 %	(4)	2.782 %	(2.556)	1.541 %	(1)
average	250	5	0.75	1.390 %	(2.111)	2.076 %	(4)	1.496 %	(2.889)	0.735 %	(1)
average	250	10	0.25	6.917 %	(3)	8.399 %	(4)	4.502 %	(2)	2.845 %	(1)
average	250	10	0.50	3.161 %	(3)	5.288 %	(4)	2.397 %	(2)	1.186 %	(1)
average	250	10	0.75	1.546 %	(3)	2.603 %	(4)	1.340 %	(2)	0.561 %	(1)
average	250	30	0.25	12.765 %	(4)	10.492 %	(3)	6.282 %	(2)	3.451 %	(1)
average	250	30	0.50	5.556 %	(3)	7.004 %	(4)	3.422 %	(2)	1.471 %	(1)
average	250	30	0.75	2.738 %	(3)	3.513 %	(4)	2.013 %	(2)	0.670 %	(1)
average	500	5	0.25	6.349 %	(2.444)	7.798 %	(4)	6.507 %	(2.556)	4.749 %	(1)
average	500	5	0.50	3.134 %	(2.667)	4.374 %	(4)	3.184 %	(2.333)	2.066 %	(1)
average	500	5	0.75	1.837 %	(2.556)	2.392 %	(4)	1.857 %	(2.444)	1.160 %	(1)
average	500	10	0.25	8.357 %	(3.1)	8.684 %	(3.9)	5.194 %	(2)	3.404 %	(1)
average	500	10	0.50	3.776 %	(3)	5.588 %	(4)	2.819 %	(2)	1.455 %	(1)
average	500	10	0.75	1.889 %	(3)	2.919 %	(4)	1.562 %	(2)	0.726 %	(1)
average	500	30	0.25	14.189 %	(4)	10.823 %	(3)	7.133 %	(2)	3.592 %	(1)
average	500	30	0.50	6.398 %	(3)	7.360 %	(4)	4.075 %	(2)	1.593 %	(1)
average	500	30	0.75	3.083 %	(3)	3.727 %	(4)	2.329 %	(2)	0.764 %	(1)
average				4.635 %	(2.909)	5.571 %	(3.885)	3.226 %	(2.206)	**1.723 %**	(1)

Each line in table B.7 represents the average performance over a set of nine test problems. These sets are too small to indicate statistical outperformance of any algorithm-topology pair on that subset.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.3.2 Ring topology

Table B.8: Details of the large MKP test results for the ring topology. Bold face indicates statistically significant outperformance of one algorithm for that topology.

problem	n	m	α	LB BPSO error	(rank)	LB MBPSO error	(rank)	LB PBPSO error	(rank)	LB SBPSO error	(rank)
average	100	5	0.25	4.209 %	(3.667)	4.173 %	(3.333)	2.107 %	(1)	2.750 %	(2)
average	100	5	0.50	1.928 %	(3.333)	1.982 %	(3.667)	0.976 %	(1)	1.261 %,	(2)
average	100	5	0.75	0.995 %	(3)	1.242 %	(4)	0.625 %	(1)	0.757 %	(2)
average	100	10	0.25	11.659 %	(4)	4.678 %	(2.778)	4.463 %	(2.222)	2.932 %	(1)
average	100	10	0.50	5.413 %	(4)	2.471 %	(3)	2.213 %	(2)	1.358 %	(1)
average	100	10	0.75	2.847 %	(4)	1.362 %	(2.889)	1.220 %	(2.111)	0.630 %	(1)
average	100	30	0.25	17.139 %	(4)	5.908 %	(2)	8.945 %	(3)	3.439 %	(1)
average	100	30	0.50	8.023 %	(4)	3.592 %	(2)	4.487 %	(3)	1.836 %	(1)
average	100	30	0.75	4.533 %	(4)	1.789 %	(2)	2.672 %	(3)	0.826 %	(1)
average	250	5	0.25	5.527 %	(3.667)	5.431 %	(3.333)	3.422 %	(1)	4.074 %	(2)
average	250	5	0.50	2.693 %	(3.111)	2.859 %	(3.889)	1.658 %	(1)	2.069 %	(2)
average	250	5	0.75	1.216 %	(3)	1.535 %	(4)	0.840 %	(1)	0.967 %	(2)
average	250	10	0.25	12.790 %	(4)	5.671 %	(3)	5.095 %	(2)	3.592 %	(1)
average	250	10	0.50	6.138 %	(4)	3.288 %	(3)	2.620 %	(2)	1.794 %	(1)
average	250	10	0.75	3.079 %	(4)	1.724 %	(3)	1.428 %	(2)	0.870 %	(1)
average	250	30	0.25	17.349 %	(4)	7.451 %	(2)	9.625 %	(3)	4.374 %	(1)
average	250	30	0.50	8.586 %	(4)	4.819 %	(2.222)	4.909 %	(2.778)	2.345 %	(1)
average	250	30	0.75	4.840 %	(4)	2.304 %	(2)	2.907 %	(3)	1.107 %	(1)
average	500	5	0.25	6.637 %	(3.111)	6.878 %	(3.889)	4.861 %	(1)	5.282 %	(2)
average	500	5	0.50	3.050 %	(3)	3.514 %	(4)	2.212 %	(1)	2.482 %	(2)
average	500	5	0.75	1.565 %	(3)	1.991 %	(4)	1.246 %	(1)	1.361 %	(2)
average	500	10	0.25	13.926 %	(4)	6.737 %	(3)	5.803 %	(2)	4.071 %	(1)
average	500	10	0.50	6.802 %	(4)	4.143 %	(3)	2.958 %	(2)	1.997 %	(1)
average	500	10	0.75	3.529 %	(4)	2.123 %	(3)	1.612 %	(2)	0.981 %	(1)
average	500	30	0.25	18.192 %	(4)	8.677 %	(2)	10.235 %	(3)	4.507 %	(1)
average	500	30	0.50	9.439 %	(4)	5.846 %	(2.889)	5.453 %	(2.111)	2.498 %	(1)
average	500	30	0.75	5.178 %	(4)	2.828 %	(2)	3.146 %	(3)	1.168 %	(1)
average				6.936 %	(3.737)	3.889 %	(2.959)	3.620 %	(1.971)	**2.271 %**	(1.333)

Each line in table B.8 represents the average performance over a set of nine test problems. These sets are too small to indicate statistical outperformance of any algorithm-topology pair on that subset.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.3.3 Von Neumann topology

Table B.9: Details of the large MKP test results for the Von Neumann topology. Bold face indicates statistically significant outperformance of one algorithm for that topology.

problem	n	m	α	VN BPSO error	(rank)	VN MBPSO error	(rank)	VN PBPSO error	(rank)	VN SBPSO error	(rank)
average	100	5	0.25	4.125 %	(3.778)	3.625 %	(3.222)	1.757 %	(1)	2.975 %	(2)
average	100	5	0.50	1.883 %	(3.667)	1.774 %	(3.333)	0.796 %	(1)	1.386 %	(2)
average	100	5	0.75	0.947 %	(2.889)	1.077 %	(4)	0.504 %	(1)	0.832 %	(2.111)
average	100	10	0.25	11.595 %	(4)	4.064 %	(2.556)	4.052 %	(2.444)	2.862 %	(1)
average	100	10	0.50	5.354 %	(4)	2.160 %	(3)	1.868 %	(2)	1.382 %	(1)
average	100	10	0.75	2.783 %	(4)	1.206 %	(3)	1.031 %	(2)	0.664 %	(1)
average	100	30	0.25	17.095 %	(4)	5.132 %	(2)	8.411 %	(3)	3.235 %	(1)
average	100	30	0.50	7.947 %	(4)	3.057 %	(2)	3.902 %	(3)	1.717 %	(1)
average	100	30	0.75	4.510 %	(4)	1.566 %	(2)	2.364 %	(3)	0.795 %	(1)
average	250	5	0.25	5.610 %	(4)	4.995 %	(3)	3.066 %	(1)	4.078 %	(2)
average	250	5	0.50	2.668 %	(3.778)	2.617 %	(3.222)	1.437 %	(1)	2.149 %	(2)
average	250	5	0.75	1.203 %	(3)	1.362 %	(4)	0.740 %	(1)	1.082 %	(2)
average	250	10	0.25	12.675 %	(4)	4.895 %	(2.667)	4.801 %	(2.333)	3.471 %	(1)
average	250	10	0.50	6.051 %	(4)	2.904 %	(3)	2.312 %	(2)	1.753 %	(1)
average	250	10	0.75	3.070 %	(4)	1.522 %	(3)	1.237 %	(2)	0.860 %	(1)
average	250	30	0.25	17.332 %	(4)	6.333 %	(2)	9.356 %	(3)	3.991 %	(1)
average	250	30	0.50	8.555 %	(4)	3.921 %	(2)	4.308 %	(3)	2.125 %	(1)
average	250	30	0.75	4.787 %	(4)	2.055 %	(2)	2.595 %	(3)	1.036 %	(1)
average	500	5	0.25	6.738 %	(4)	6.299 %	(3)	4.565 %	(1)	5.412 %	(2)
average	500	5	0.50	3.006 %	(3.222)	3.219 %	(3.778)	2.026 %	(1)	2.565 %	(2)
average	500	5	0.75	1.504 %	(2.889)	1.849 %	(4)	1.157 %	(1)	1.422 %	(2.111)
average	500	10	0.25	13.917 %	(4)	5.943 %	(3)	5.652 %	(2)	4.017 %	(1)
average	500	10	0.50	6.788 %	(4)	3.522 %	(3)	2.677 %	(2)	1.973 %	(1)
average	500	10	0.75	3.461 %	(4)	1.905 %	(3)	1.399 %	(2)	0.997 %	(1)
average	500	30	0.25	18.178 %	(4)	7.089 %	(2)	10.003 %	(3)	4.037 %	(1)
average	500	30	0.50	9.449 %	(4)	4.655 %	(2.111)	4.822 %	(2.889)	2.246 %	(1)
average	500	30	0.75	5.160 %	(4)	2.378 %	(2)	2.811 %	(3)	1.112 %	(1)
average				6.903 %	(3.823)	3.375 %	(2.811)	3.32 %	(2.025)	**2.229 %**	(1.342)

Each line in table B.9 represents the average performance over a set of nine test problems. These sets are too small to indicate statistical outperformance of any algorithm-topology pair on that subset.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.4 Summarized testing results per algorithm

Table B.10: Summary of the large MKP test results per algorithm. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

problem		error	(rank)	error	(rank)	error	(rank)
		GB BPSO		LB BPSO		VN BPSO	
average error		**4.679 %**	(1.340)	7.006 %	(2.510)	6.973 %	(2.150)
stdev error		3.468 %		5.000 %		5.000 %	
n	100	**3.831 %**	(1.296)	6.348 %	(2.537)	6.291 %	(2.167)
n	250	**4.679 %**	(1.327)	6.951 %	(2.531)	6.920 %	(2.142)
n	500	**5.526 %**	(1.395)	7.719 %	(2.451)	7.707 %	(2.154)
m	5	**3.037 %**	(2.019)	3.091 %	(2.179)	**3.076 %**	(1.802)
m	10	**3.942 %**	(1.000)	7.520 %	(2.704)	7.465 %	(2.296)
m	30	**7.057 %**	(1.000)	10.407 %	(2.636)	10.377 %	(2.364)
α	0.25	**8.253 %**	(1.037)	11.961 %	(2.555)	11.943 %	(2.409)
α	0.50	**3.751 %**	(1.307)	5.817 %	(2.530)	5.775 %	(2.163)
α	0.75	**1.909 %**	(1.692)	3.063 %	(2.429)	**3.023 %**	(1.878)
		GB MBPSO		LB MBPSO		VN MBPSO	
average error		5.619 %	(3)	3.922 %	(1.99)	**3.403 %**	(1.01)
stdev error		2.723 %		2.100 %		1.700 %	
n	100	5.160 %	(3.000)	3.044 %	(1.988)	**2.647 %**	(1.012)
n	250	5.663 %	(3.000)	3.917 %	(2.000)	**3.418 %**	(1.000)
n	500	6.034 %	(3.000)	4.805 %	(1.975)	**4.145 %**	(1.025)
m	5	4.354 %	(3.000)	3.289 %	(1.963)	**2.980 %**	(1.037)
m	10	5.521 %	(3.000)	3.654 %	(2.000)	**3.191 %**	(1.000)
m	30	6.983 %	(3.000)	4.824 %	(2.000)	**4.039 %**	(1.000)
α	0.25	8.664 %	(3.000)	6.185 %	(2.000)	**5.382 %**	(1.000)
α	0.50	5.344 %	(3.000)	3.608 %	(1.988)	**3.089 %**	(1.012)
α	0.75	2.712 %	(3.000)	1.878 %	(1.974)	**1.658 %**	(1.026)
		GB PBPSO		LB PBPSO		VN PBPSO	
average error		3.250 %	(1.860)	3.650 %	(2.650)	**3.348 %**	(1.500)
stdev error		1.718 %		2.600 %		2.500 %	
n	100	**2.568 %**	(1.790)	3.101 %	(2.667)	**2.762 %**	(1.543)
n	250	3.286 %	(1.864)	3.626 %	(2.654)	**3.330 %**	(1.481)
n	500	3.896 %	(1.914)	4.221 %	(2.617)	**3.954 %**	(1.469)
m	5	3.134 %	(3.000)	1.994 %	(2.000)	**1.783 %**	(1.000)
m	10	**2.763 %**	(1.568)	3.112 %	(2.951)	**2.847 %**	(1.481)
m	30	**3.853 %**	(1.000)	5.842 %	(2.988)	5.416 %	(2.012)
α	0.25	**5.264 %**	(1.695)	6.059 %	(2.646)	**5.739 %**	(1.659)
α	0.50	2.799 %	(1.904)	3.066 %	(2.663)	**2.693 %**	(1.434)
α	0.75	1.613 %	(1.974)	1.738 %	(2.628)	**1.533 %**	(1.397)
		GB SBPSO		LB SBPSO		VN SBPSO	
average error		**1.740 %**	(1.000)	2.292 %	(2.570)	2.249 %	(2.430)
stdev error		1.170 %		1.300 %		1.300 %	
n	100	**1.260 %**	(1.000)	1.767 %	(2.519)	1.772 %	(2.481)
n	250	**1.758 %**	(1.000)	2.366 %	(2.636)	2.294 %	(2.364)
n	500	**2.201 %**	(1.000)	2.743 %	(2.543)	2.680 %	(2.457)
m	5	**1.875 %**	(1.000)	2.334 %	(2.173)	2.433 %	(2.827)
m	10	**1.553 %**	(1.000)	2.075 %	(2.549)	2.046 %	(2.451)
m	30	**1.791 %**	(1.000)	2.468 %	(2.975)	2.266 %	(2.025)
α	0.25	**3.141 %**	(1.000)	3.893 %	(2.659)	3.789 %	(2.341)
α	0.50	**1.355 %**	(1.000)	1.957 %	(2.566)	1.917 %	(2.434)
α	0.75	**0.676 %**	(1.000)	0.966 %	(2.468)	0.981 %	(2.532)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.5 Detailed test results per algorithm

B.5.1 BPSO

Table B.11: Details of the large MKP test results for BPSO. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

				GB BPSO		LB BPSO		VN BPSO	
problem	n	m	α	error	(rank)	error	(rank)	error	(rank)
average	100	5	0.25	3.772 %	(1.111)	4.209 %	(2.667)	4.125 %	(2.222)
average	100	5	0.50	1.835 %	(1.556)	1.928 %	(2.556)	1.883 %	(1.889)
average	100	5	0.75	1.155 %	(3.000)	0.995 %	(1.889)	0.947 %	(1.111)
average	100	10	0.25	5.334 %	(1.000)	11.659 %	(2.778)	11.595 %	(2.222)
average	100	10	0.50	2.384 %	(1.000)	5.413 %	(2.556)	5.354 %	(2.444)
average	100	10	0.75	1.217 %	(1.000)	2.847 %	(2.667)	2.783 %	(2.333)
average	100	30	0.25	11.396 %	(1.000)	17.139 %	(2.389)	17.095 %	(2.611)
average	100	30	0.50	4.786 %	(1.000)	8.023 %	(2.700)	7.947 %	(2.300)
average	100	30	0.75	2.327 %	(1.000)	4.533 %	(2.625)	4.510 %	(2.375)
average	250	5	0.25	5.182 %	(1.111)	5.527 %	(2.333)	5.610 %	(2.556)
average	250	5	0.50	2.680 %	(1.833)	2.693 %	(2.222)	2.668 %	(1.944)
average	250	5	0.75	1.390 %	(3.000)	1.216 %	(1.667)	1.203 %	(1.333)
average	250	10	0.25	6.917 %	(1.000)	12.790 %	(2.778)	12.675 %	(2.222)
average	250	10	0.50	3.161 %	(1.000)	6.138 %	(2.800)	6.051 %	(2.200)
average	250	10	0.75	1.546 %	(1.000)	3.079 %	(2.625)	3.070 %	(2.375)
average	250	30	0.25	12.765 %	(1.000)	17.349 %	(2.556)	17.332 %	(2.444)
average	250	30	0.50	5.556 %	(1.000)	8.586 %	(2.778)	8.555 %	(2.222)
average	250	30	0.75	2.738 %	(1.000)	4.840 %	(3.000)	4.787 %	(2.000)
average	500	5	0.25	6.349 %	(1.111)	6.637 %	(2.333)	6.738 %	(2.556)
average	500	5	0.50	3.134 %	(2.444)	3.050 %	(2.000)	3.006 %	(1.556)
average	500	5	0.75	1.837 %	(3.000)	1.565 %	(1.944)	1.504 %	(1.056)
average	500	10	0.25	8.357 %	(1.000)	13.926 %	(2.700)	13.917 %	(2.300)
average	500	10	0.50	3.776 %	(1.000)	6.802 %	(2.556)	6.788 %	(2.444)
average	500	10	0.75	1.889 %	(1.000)	3.529 %	(2.875)	3.461 %	(2.125)
average	500	30	0.25	14.189 %	(1.000)	18.192 %	(2.444)	18.178 %	(2.556)
average	500	30	0.50	6.398 %	(1.000)	9.439 %	(2.556)	9.449 %	(2.444)
average	500	30	0.75	3.083 %	(1.000)	5.178 %	(2.667)	5.160 %	(2.333)
average				**4.635 %**	(1.339)	6.936 %	(2.506)	6.903 %	(2.155)

Each line in table B.11 represents the average performance over a set of nine test problems. These sets are too small to indicate statistical outperformance of any algorithm-topology pair on that subset.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.5.2 MBPSO

Table B.12: Details of the large MKP test results for MBPSO. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

				GB MBPSO		LB MBPSO		VN MBPSO	
problem	n	m	α	error	(rank)	error	(rank)	error	(rank)
average	100	5	0.25	6.071 %	(3.000)	4.173 %	(2.000)	3.625 %	(1.000)
average	100	5	0.50	3.392 %	(3.000)	1.982 %	(2.000)	1.774 %	(1.000)
average	100	5	0.75	1.785 %	(3.000)	1.242 %	(1.889)	1.077 %	(1.111)
average	100	10	0.25	8.291 %	(3.000)	4.678 %	(2.000)	4.064 %	(1.000)
average	100	10	0.50	4.605 %	(3.000)	2.471 %	(2.000)	2.160 %	(1.000)
average	100	10	0.75	2.371 %	(3.000)	1.362 %	(2.000)	1.206 %	(1.000)
average	100	30	0.25	10.276 %	(3.000)	5.908 %	(2.000)	5.132 %	(1.000)
average	100	30	0.50	6.232 %	(3.000)	3.592 %	(2.000)	3.057 %	(1.000)
average	100	30	0.75	3.069 %	(3.000)	1.789 %	(2.000)	1.566 %	(1.000)
average	250	5	0.25	7.140 %	(3.000)	5.431 %	(2.000)	4.995 %	(1.000)
average	250	5	0.50	4.158 %	(3.000)	2.859 %	(2.000)	2.617 %	(1.000)
average	250	5	0.75	2.076 %	(3.000)	1.535 %	(2.000)	1.362 %	(1.000)
average	250	10	0.25	8.399 %	(3.000)	5.671 %	(2.000)	4.895 %	(1.000)
average	250	10	0.50	5.288 %	(3.000)	3.288 %	(2.000)	2.904 %	(1.000)
average	250	10	0.75	2.603 %	(3.000)	1.724 %	(2.000)	1.522 %	(1.000)
average	250	30	0.25	10.492 %	(3.000)	7.451 %	(2.000)	6.333 %	(1.000)
average	250	30	0.50	7.004 %	(3.000)	4.819 %	(2.000)	3.921 %	(1.000)
average	250	30	0.75	3.513 %	(3.000)	2.304 %	(2.000)	2.055 %	(1.000)
average	500	5	0.25	7.798 %	(3.000)	6.878 %	(2.000)	6.299 %	(1.000)
average	500	5	0.50	4.374 %	(3.000)	3.514 %	(1.889)	3.219 %	(1.111)
average	500	5	0.75	2.392 %	(3.000)	1.991 %	(1.889)	1.849 %	(1.111)
average	500	10	0.25	8.684 %	(3.000)	6.737 %	(2.000)	5.943 %	(1.000)
average	500	10	0.50	5.588 %	(3.000)	4.143 %	(2.000)	3.522 %	(1.000)
average	500	10	0.75	2.919 %	(3.000)	2.123 %	(2.000)	1.905 %	(1.000)
average	500	30	0.25	10.823 %	(3.000)	8.677 %	(2.000)	7.089 %	(1.000)
average	500	30	0.50	7.360 %	(3.000)	5.846 %	(2.000)	4.655 %	(1.000)
average	500	30	0.75	3.727 %	(3.000)	2.828 %	(2.000)	2.378 %	(1.000)
average				5.571 %	(3.000)	3.889 %	(1.988)	**3.375 %**	(1.012)

Each line in table B.12 represents the average performance over a set of nine test problems. These sets are too small to indicate statistical outperformance of any algorithm-topology pair on that subset.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.5.3 PBPSO

Table B.13: Details of the large MKP test results for PBPSO. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

				GB PBPSO		LB PBPSO		VN PBPSO	
problem	n	m	α	error	(rank)	error	(rank)	error	(rank)
average	100	5	0.25	3.986 %	(3.000)	2.107 %	(2.000)	1.757 %	(1.000)
average	100	5	0.50	1.959 %	(3.000)	0.976 %	(2.000)	0.796 %	(1.000)
average	100	5	0.75	1.212 %	(3.000)	0.625 %	(2.000)	0.504 %	(1.000)
average	100	10	0.25	3.631 %	(1.000)	4.463 %	(3.000)	4.052 %	(2.000)
average	100	10	0.50	1.851 %	(1.556)	2.213 %	(3.000)	1.868 %	(1.444)
average	100	10	0.75	1.053 %	(1.556)	1.220 %	(3.000)	1.031 %	(1.444)
average	100	30	0.25	4.928 %	(1.000)	8.945 %	(3.000)	8.411 %	(2.000)
average	100	30	0.50	2.748 %	(1.000)	4.487 %	(3.000)	3.902 %	(2.000)
average	100	30	0.75	1.621 %	(1.000)	2.672 %	(3.000)	2.364 %	(2.000)
average	250	5	0.25	5.219 %	(3.000)	3.422 %	(2.000)	3.066 %	(1.000)
average	250	5	0.50	2.782 %	(3.000)	1.658 %	(2.000)	1.437 %	(1.000)
average	250	5	0.75	1.496 %	(3.000)	0.840 %	(2.000)	0.740 %	(1.000)
average	250	10	0.25	4.502 %	(1.222)	5.095 %	(3.000)	4.801 %	(1.778)
average	250	10	0.50	2.397 %	(1.700)	2.620 %	(3.000)	2.312 %	(1.300)
average	250	10	0.75	1.340 %	(1.875)	1.428 %	(3.000)	1.237 %	(1.125)
average	250	30	0.25	6.282 %	(1.000)	9.625 %	(2.889)	9.356 %	(2.111)
average	250	30	0.50	3.422 %	(1.000)	4.909 %	(3.000)	4.308 %	(2.000)
average	250	30	0.75	2.013 %	(1.000)	2.907 %	(3.000)	2.595 %	(2.000)
average	500	5	0.25	6.507 %	(3.000)	4.861 %	(2.000)	4.565 %	(1.000)
average	500	5	0.50	3.184 %	(3.000)	2.212 %	(2.000)	2.026 %	(1.000)
average	500	5	0.75	1.857 %	(3.000)	1.246 %	(2.000)	1.157 %	(1.000)
average	500	10	0.25	5.194 %	(1.100)	5.803 %	(2.900)	5.652 %	(2.000)
average	500	10	0.50	2.819 %	(2.000)	2.958 %	(2.889)	2.677 %	(1.111)
average	500	10	0.75	1.562 %	(2.250)	1.612 %	(2.750)	1.399 %	(1.000)
average	500	30	0.25	7.133 %	(1.000)	10.235 %	(3.000)	10.003 %	(2.000)
average	500	30	0.50	4.075 %	(1.000)	5.453 %	(3.000)	4.822 %	(2.000)
average	500	30	0.75	2.329 %	(1.000)	3.146 %	(3.000)	2.811 %	(2.000)
average				3.226 %	(1.861)	3.620 %	(2.645)	**3.320 %**	(1.493)

Each line in table B.13 represents the average performance over a set of nine test problems. These sets are too small to indicate statistical outperformance of any algorithm-topology pair on that subset.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

B.5.4 SBPSO

Table B.14: Details of the large MKP test results for SBPSO. Bold face indicates statistically significant outperformance of one or more topologies for that algorithm.

				GB SBPSO		LB SBPSO		VN SBPSO	
problem	n	m	α	error	(rank)	error	(rank)	error	(rank)
average	100	5	0.25	1.951 %	(1.000)	2.750 %	(2.111)	2.975 %	(2.889)
average	100	5	0.50	0.873 %	(1.000)	1.261 %	(2.000)	1.386 %	(3.000)
average	100	5	0.75	0.505 %	(1.000)	0.757 %	(2.111)	0.832 %	(2.889)
average	100	10	0.25	2.181 %	(1.000)	2.932 %	(2.778)	2.862 %	(2.222)
average	100	10	0.50	0.853 %	(1.000)	1.358 %	(2.444)	1.382 %	(2.556)
average	100	10	0.75	0.399 %	(1.000)	0.630 %	(2.222)	0.664 %	(2.778)
average	100	30	0.25	2.773 %	(1.000)	3.439 %	(3.000)	3.235 %	(2.000)
average	100	30	0.50	1.195 %	(1.000)	1.836 %	(3.000)	1.717 %	(2.000)
average	100	30	0.75	0.540 %	(1.000)	0.826 %	(3.000)	0.795 %	(2.000)
average	250	5	0.25	3.294 %	(1.000)	4.074 %	(2.444)	4.078 %	(2.556)
average	250	5	0.50	1.541 %	(1.000)	2.069 %	(2.333)	2.149 %	(2.667)
average	250	5	0.75	0.735 %	(1.000)	0.967 %	(2.000)	1.082 %	(3.000)
average	250	10	0.25	2.845 %	(1.000)	3.592 %	(2.778)	3.471 %	(2.222)
average	250	10	0.50	1.186 %	(1.000)	1.794 %	(2.600)	1.753 %	(2.400)
average	250	10	0.75	0.561 %	(1.000)	0.870 %	(2.563)	0.860 %	(2.438)
average	250	30	0.25	3.451 %	(1.000)	4.374 %	(3.000)	3.991 %	(2.000)
average	250	30	0.50	1.471 %	(1.000)	2.345 %	(3.000)	2.125 %	(2.000)
average	250	30	0.75	0.670 %	(1.000)	1.107 %	(3.000)	1.036 %	(2.000)
average	500	5	0.25	4.749 %	(1.000)	5.282 %	(2.222)	5.412 %	(2.778)
average	500	5	0.50	2.066 %	(1.000)	2.482 %	(2.222)	2.565 %	(2.778)
average	500	5	0.75	1.160 %	(1.000)	1.361 %	(2.111)	1.422 %	(2.889)
average	500	10	0.25	3.404 %	(1.000)	4.071 %	(2.600)	4.017 %	(2.400)
average	500	10	0.50	1.455 %	(1.000)	1.997 %	(2.444)	1.973 %	(2.556)
average	500	10	0.75	0.726 %	(1.000)	0.981 %	(2.500)	0.997 %	(2.500)
average	500	30	0.25	3.592 %	(1.000)	4.507 %	(3.000)	4.037 %	(2.000)
average	500	30	0.50	1.593 %	(1.000)	2.498 %	(3.000)	2.246 %	(2.000)
average	500	30	0.75	0.764 %	(1.000)	1.168 %	(2.778)	1.112 %	(2.222)
average				**1.723 %**	(1.000)	2.271 %	(2.565)	2.229 %	(2.435)

Each line in table B.14 represents the average performance over a set of nine test problems. These sets are too small to indicate statistical outperformance of any algorithm-topology pair on that subset.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Appendix C

Detailed results for the FSP

This appendix contains the detailed results of the experiments run on the FSP. In total, 38 such problems were considered, all of which are known from the literature and are described in appendix D. To solve the underlying classification problem, three different classifiers were used: the GNB, J48, and *k*-NN classifiers.

The J48 and *k*-NN classifiers have parameters that need to be set and these values influence the behavior of the classifier. Therefore, a classifier tuning process was used and this was described in section 6.2.3.3. A separate investigation using an exhaustive search of the classifiers parameters and the feature subset was conducted to test whether this setup of the classifier tuning was adequate. This investigation and an overview of its results was described in section 6.4. In section C.1 of this appendix, detailed results of the investigation on the J48 classifier are given.

Four different algorithms, BPSO, CFBPSO, PBPSO, and SBPSO, were compared in the experiments on the FSP and each algorithm was run in a wrapper setup for each of the three classifiers, resulting in 12 algorithm-classifier pairs. Each such combination of an algorithm and a classifier was tuned on eight tuning problems. This tuning process was described in section 6.5, where an overview of the results was also given. Section C.2 of this appendix contains the detailed results of the PSO parameter tuning experiments.

The tuned algorithm-classifier pair were then applied on the test set of 30 problems, where 30 independent runs were simulated for each of these 12 combinations. The exact experimental set-up for the testing process was described in detail in section 6.2 and an overview of the results was given in section 6.6. Section C.3 of this appendix contains the detailed results of these experiments on the tuned PSO algorithms.

C.1 Detailed FSP results for exhaustive search on J48 parameters

This section contains the detailed results of the experiments in which the parameters for the J48 classifiers were investigated using an exhaustive search. The main results were discussed in section 6.4.2.1 with the summary of the results listed in tables 6.11 and 6.12. The detailed results of the experiments are listed below using a separate table and discussed in a separate subsection for each of the nine datasets studied,

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

namely iris, corral, liver, monk-1, monk-2, pima, breasttissue, glass, and tic-tac-toe.

Each table in this section shows the probability scores for the 49 parameter combinations. The cells in the table are colored in grey-scale according to the value in the cell, with a darker hue indicating a higher probability score. The parameters l and γ which combine to form the combination with the highest probability are indicated in bold and the corresponding cell has the darkest hue. This color scheme makes it possible to quickly identify areas of good combinations of l and γ for use in the J48 classifier on the dataset.

C.1.1 Iris dataset

Table C.1 shows the probability scores for the iris dataset. The best parameter values found are $l = 2$ and $\gamma = 0.050$. A combination of low values for both l and γ performs best. Higher values for l lead to deterioration in the performance of the J48 classifier. A higher value for γ has only leads to a small decrease in performance during exhaustive search.

The parameters actually used in the exhaustive search and found using the method described in section 6.2.3.3 were $l = 18$ and $\gamma = 0.275$. This parameter combination for the J48 classifier is suboptimal, ranking 15th out of 49, but can hardly be distinguished from any combination of l and γ were l ranges between 2 and 34. The chosen parameters values for l and γ thus seem acceptable, as this parameter combination's probability, 2.5%, is 82% of the highest found probability, 3.1%.

Table C.1: Detailed results classifier parameter values for J48 classifier on iris dataset.

$\gamma \setminus l$	**2**	10	18	26	34	42	50
0.050	3.1 %	2.6 %	2.4 %	2.4 %	2.5 %	1.5 %	0.2 %
0.125	2.5 %	2.6 %	2.5 %	2.4 %	2.6 %	1.5 %	0.2 %
0.200	2.7 %	2.6 %	2.5 %	2.5 %	2.6 %	1.5 %	0.2 %
0.275	2.5 %	2.5 %	2.5 %	2.5 %	2.6 %	1.5 %	0.2 %
0.350	2.3 %	2.5 %	2.5 %	2.5 %	2.6 %	1.5 %	0.2 %
0.425	2.3 %	2.4 %	2.5 %	2.4 %	2.6 %	1.5 %	0.2 %
0.500	2.3 %	2.4 %	2.5 %	2.4 %	2.6 %	1.5 %	0.2 %

C.1.2 Corral dataset

Table C.2 shows the probability scores for the corral dataset. The best parameter values found are $l = 2$ and $\gamma = 0.125$. Low values for l perform best while $l \geq 26$ quickly leads to deterioration in the performance of the J48 classifier. Performance is less sensitive to the value of γ. For higher values of γ combinations with $l = 10$ perform less well than those with either $l = 2$ or $l = 18$, leading to a slightly more complicated performance landscape.

The parameters actually used in the exhaustive search and found using the method described in section 6.2.3.3 were $l = 2$ and $\gamma = 0.050$. This parameter combination for the J48 classifier is suboptimal, ranking 7th from 49, as a higher value for γ yields better performance in all cases except where $l =$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

10. Still, the relative performance seems adequate for use in the exhaustive search experiments as the probability score for the chosen parameter combination, 4.1%, is more than 75% of the probability score for the best parameter combination, 4.9%.

Table C.2: Detailed results classifier parameter values for J48 classifier on corral dataset.

$\gamma \setminus l$	**2**	10	18	26	34	42	50
0.050	3.8 %	3.3 %	3.4 %	1.5 %	0.5 %	0.5 %	0.5 %
0.125	4.9 %	3.4 %	3.6 %	1.9 %	0.5 %	0.5 %	0.5 %
0.200	4.9 %	3.0 %	3.6 %	1.9 %	0.5 %	0.5 %	0.5 %
0.275	4.9 %	2.8 %	3.6 %	1.9 %	0.5 %	0.5 %	0.5 %
0.350	4.4 %	2.5 %	3.6 %	1.9 %	0.5 %	0.5 %	0.5 %
0.425	4.2 %	2.3 %	3.6 %	1.8 %	0.5 %	0.5 %	0.5 %
0.500	4.1 %	2.3 %	3.6 %	1.8 %	0.5 %	0.5 %	0.5 %

C.1.3 Liver dataset

Table C.3 shows the probability scores for the liver dataset. The best parameter values found are $l = 42$ and $\gamma = 0.125$. Higher values for l perform better in general on this dataset, with $l = 42$ being the optimum for any of the seven possible values of γ. For lower γ values lead to better performance, and the combination with $l = 42$ is the only one for which the lowest value $\gamma = 0.050$ is not the best. The γ parameter has a slightly larger impact on performance than the l parameter.

The parameters actually used in the exhaustive search and found using the method described in section 6.2.3.3 were $l = 26$ and $\gamma = 0.275$. This parameter combination for the J48 classifier is suboptimal, ranking 17th from 49. Also its probability is only 69% of that for the best found probability, the worst result for the chosen parameter combination for the J48 classifier across the five datasets investigated. This seems not an unacceptable result, but clearly not desirable.

Table C.3: Detailed results classifier parameter values for J48 classifier on liver dataset.

$\gamma \setminus l$	2	10	18	26	34	**42**	50
0.050	2.3 %	1.9 %	2.2 %	2.7 %	2.5 %	3.0 %	2.5 %
0.125	2.0 %	1.5 %	1.9 %	2.6 %	2.4 %	3.3 %	2.3 %
0.200	1.8 %	1.3 %	1.8 %	2.5 %	2.4 %	3.2 %	2.2 %
0.275	1.8 %	1.3 %	1.8 %	2.3 %	2.2 %	2.8 %	2.0 %
0.350	1.6 %	1.2 %	1.7 %	2.1 %	2.1 %	2.7 %	1.9 %
0.425	1.6 %	1.1 %	1.8 %	2.0 %	2.1 %	2.6 %	1.9 %
0.500	1.3 %	0.9 %	1.6 %	1.8 %	1.8 %	2.3 %	1.6 %

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.1.4 Monk-1 dataset

Table C.4 shows the probability scores for the monk-1 dataset. The best parameter values found are $l = 2$ and $\gamma = 0.050$. Low values γ clearly perform best on this dataset, while performance is completely insensitive to the value of l. For γ, lower values lead to better performance, and the combination with $l = 42$ is the only one for which the lowest value $\gamma = 0.050$ is not the best. The γ parameter has a slightly larger impact on performance than the l parameter.

The parameters actually used in the exhaustive search and found using the method described in section 6.2.3.3 were $l = 34$ and $\gamma = 0.275$. Although the different value for l is inconsequential, the value chosen for γ is clearly not ideal. The chosen parameter combination ranks 22nd out of 49. Its probability is still 78% of that for the best found probability, which seems acceptable.

Table C.4: Detailed results classifier parameter values for J48 classifier on monk-1 dataset.

$\gamma \setminus l$	**2**	10	18	26	34	42	50
0.050	2.7 %	2.7 %	2.7 %	2.7 %	2.7 %	2.7 %	2.7 %
0.125	2.5 %	2.5 %	2.5 %	2.5 %	2.5 %	2.5 %	2.5 %
0.200	2.5 %	2.5 %	2.5 %	2.5 %	2.5 %	2.5 %	2.5 %
0.275	2.1 %	2.1 %	2.1 %	2.1 %	2.1 %	2.1 %	2.1 %
0.350	1.7 %	1.7 %	1.7 %	1.7 %	1.7 %	1.7 %	1.7 %
0.425	1.4 %	1.4 %	1.4 %	1.4 %	1.4 %	1.4 %	1.4 %
0.500	1.4 %	1.4 %	1.4 %	1.4 %	1.4 %	1.4 %	1.4 %

C.1.5 Monk-2 dataset

Table C.5 shows the probability scores for the monk-2 dataset. The best parameter values found are $l = 2$ and $\gamma = 0.275$. It can be easily seen that performance is very equal across most combinations investigated, with 40 out of 49 attaining a probability within 75% of that for the best combination and 29 scoring a probability of 2.2%. Only combinations of high γ values and $l \leq 42$ lead to poor results.

Table C.5: Detailed results classifier parameter values for J48 classifier on monk-2 dataset.

$\gamma \setminus l$	**2**	10	18	26	34	42	50
0.050	2.2 %	2.2 %	2.2 %	2.2 %	2.2 %	2.2 %	2.2 %
0.125	2.2 %	2.2 %	2.2 %	2.2 %	2.2 %	2.2 %	2.2 %
0.200	2.2 %	2.2 %	2.2 %	2.2 %	2.2 %	2.2 %	2.2 %
0.275	2.4 %	2.1 %	2.1 %	2.1 %	2.2 %	2.2 %	2.2 %
0.350	2.3 %	1.9 %	2.0 %	2.1 %	2.1 %	2.2 %	2.2 %
0.425	1.9 %	1.6 %	1.6 %	1.8 %	2.0 %	2.1 %	2.2 %
0.500	1.3 %	0.9 %	0.9 %	1.0 %	1.4 %	1.5 %	2.2 %

The J48 parameters actually used during the exhaustive search and found using the method described

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

in section 6.2.3.3 were $l = 2$ and $\gamma = 0.425$. Given that performance is so equal across most of the parameter combinations, it is remarkable that the chosen combination actually is quite poorly ranked: 39th out of 49. Its probability score is still 78% of that for the best found score, however, which seems acceptable.

C.1.6 Pima dataset

Table C.6 shows the probability scores for the pima dataset. The best parameter values found are $l = 34$ and $\gamma = 0.200$. Small values of l combine with larger values for γ are seen to perform worst. This differs from the patterns generally seen on the other datasets, which favor smaller values for both l and γ.

The parameters actually used in the exhaustive search and found using the method described in section 6.2.3.3 were $l = 2$ and $\gamma = 0.425$. This parameter combination for the J48 classifier is very poor, ranking 48th out of 49. Tuning the J48 classifier using all features thus yields a poorly tuned classifier if used on only a subset of features and can be said to fail on this dataset.

Table C.6: Detailed results classifier parameter values for J48 classifier on pima dataset.

$\gamma \setminus l$	2	10	18	26	**34**	42	50
0.050	1.6 %	2.0 %	1.8 %	2.3 %	2.7 %	2.7 %	2.7 %
0.125	1.4 %	1.8 %	1.7 %	2.2 %	2.8 %	2.7 %	2.7 %
0.200	1.4 %	1.9 %	1.7 %	2.3 %	2.9 %	2.7 %	2.7 %
0.275	1.2 %	1.8 %	1.7 %	2.2 %	2.9 %	2.6 %	2.7 %
0.350	1.0 %	1.6 %	1.6 %	2.1 %	2.7 %	2.5 %	2.6 %
0.425	0.8 %	1.3 %	1.5 %	1.9 %	2.4 %	2.3 %	2.4 %
0.500	0.8 %	1.2 %	1.3 %	1.8 %	2.3 %	2.0 %	2.2 %

C.1.7 Breasttissue dataset

Table C.7 shows the probability scores for the breasttissue dataset.

Table C.7: Detailed results classifier parameter values for J48 classifier on breasttissue dataset.

$\gamma \setminus l$	**2**	10	18	26	34	42	50
0.050	9.7 %	5.4 %	0.2 %	0.0 %	0.0 %	0.0 %	0.0 %
0.125	9.5 %	5.1 %	0.2 %	0.0 %	0.0 %	0.0 %	0.0 %
0.200	9.1 %	5.1 %	0.2 %	0.0 %	0.0 %	0.0 %	0.0 %
0.275	8.9 %	5.3 %	0.2 %	0.0 %	0.0 %	0.0 %	0.0 %
0.350	8.9 %	4.8 %	0.2 %	0.0 %	0.0 %	0.0 %	0.0 %
0.425	8.9 %	4.5 %	0.2 %	0.0 %	0.0 %	0.0 %	0.0 %
0.500	8.7 %	4.5 %	0.2 %	0.0 %	0.0 %	0.0 %	0.0 %

The best parameter values found are $l = 2$ and $\gamma = 0.050$, which is in line with the behavior seen

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

across most of the nine datasets investigated. Any parameter combination involving $l = 2$ works well, those with $l = 10$ work adequately, but if l is set higher than 10, performance suffers a great deal, resulting in a combined probability of less than 2% that such a parameter combination works well on any given features subset.

The parameters actually used in the exhaustive search and found using the method described in section 6.2.3.3 were $l = 2$ and $\gamma = 0.125$. This parameter combination ranks second best out of 49 and the chosen classifier tuning method thus is shown to work very well on the breasttissue dataset.

C.1.8 Glass dataset

Table C.8 shows the probability scores for the glass dataset. The best parameter values found are $l = 2$ and $\gamma = 0.050$, which is in line with the behavior seen across most of the nine datasets investigated. Any parameter combination involving $l = 2$ works well, those with $l = 10$ work adequately, but if l is set higher than 10, performance suffers a great deal: the sum of probability scores for all parameter combinations with $l > 10$ was less than 15%. This means that parameter combinations using $l = 2$ or $l = 10$ have a combined probability score of more than 85% and should work well on any given features subset.

Table C.8: Detailed results classifier parameter values for J48 classifier on glass dataset.

$\gamma \setminus l$	**2**	10	18	26	34	42	50
0.050	8.4 %	4.4 %	0.8 %	0.7 %	0.2 %	0.1 %	0.1 %
0.125	8.2 %	4.9 %	0.8 %	0.7 %	0.2 %	0.1 %	0.1 %
0.200	7.9 %	4.7 %	0.7 %	0.8 %	0.2 %	0.1 %	0.1 %
0.275	7.3 %	4.8 %	0.9 %	0.8 %	0.3 %	0.1 %	0.1 %
0.350	7.2 %	4.6 %	0.9 %	0.8 %	0.3 %	0.1 %	0.1 %
0.425	7.2 %	4.7 %	0.9 %	0.8 %	0.3 %	0.1 %	0.1 %
0.500	7.0 %	4.7 %	0.9 %	0.8 %	0.3 %	0.1 %	0.1 %

The parameters actually used in the exhaustive search and found using the method described in section 6.2.3.3 were $l = 2$ and $\gamma = 0.125$. This parameter combination ranks second best out of 49 and the chosen classifier tuning method thus is shown to work very well on the glass dataset.

C.1.9 Tic-tac-toe dataset

Table C.9 shows the probability scores for the tic-tac-toe dataset. The best parameter values found are $l = 18$ and $\gamma = 0.125$ and in general the probability score reduces for higher values for l and lower values for γ, but the shape of the surface is not completely straightforward: the upper and lower edges of the grid show better scores than the minimum at $l = 50$ and $\gamma = 0.200$.

The J48 parameters actually used during the exhaustive search and found using the method described in section 6.2.3.3 were $l = 18$ and $\gamma = 0.125$. This chosen parameter combination ranks 19th out of 49, and its probability score is only 45% of the maximum recorded probability score. Thus, the chosen clas-

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

sifier tuning method has not yielded a good result for the tic-tac-toe dataset. This is somewhat surprising, as the concept underlying the artificial dataset requires all features for successful classification.

Table C.9: Detailed results classifier parameter values for J48 classifier on tic-tac-toe dataset.

$\gamma \setminus l$	**2**	10	18	26	34	42	50
0.050	3.6 %	2.0 %	1.9 %	1.4 %	1.1 %	1.2 %	1.2 %
0.125	3.9 %	2.4 %	2.1 %	1.5 %	1.0 %	1.0 %	0.9 %
0.200	4.1 %	2.8 %	2.0 %	1.8 %	1.0 %	1.0 %	0.9 %
0.275	4.0 %	2.9 %	2.2 %	2.2 %	1.1 %	1.2 %	1.0 %
0.350	4.5 %	3.3 %	2.0 %	2.4 %	1.3 %	1.4 %	1.3 %
0.425	4.4 %	3.1 %	1.7 %	2.3 %	1.2 %	1.5 %	1.4 %
0.500	4.3 %	2.7 %	1.4 %	2.2 %	1.2 %	1.6 %	1.4 %

C.2 Detailed FSP tuning results

This section lists the detailed results of the experiments performed to tune the four PSO algorithms (BPSO, CFBPSO, PBPSO and SBPSO) on the FSP. In this tuning process 128 different parameter combinations were investigated for each PSO algorithm and each classifier. The parameters resulting from the tuning process were discussed in section 6.5. The accuracy achieved and number of features selected using the chosen combination of parameters for each of the four PSO algorithms and each of the three classifiers is shown below.

Note that the process used for tuning was set out in section 6.5.1 and a brief statistical summary (the number of instances, classes and features) of the eight datasets used in the PSO tuning process for FSP can be found in table 6.2. A detailed description of the datasets and any specific actions taken in preprocessing the data is set out in appendix D.

The results in this section are shown separately for each of the three classifiers investigated: the GNB classifier in section C.2.1, the J48 classifier in section C.2.2, and the *k*-NN classifier in section C.2.3. Each of the three sections contains two tables. The first table lists the average accuracy and standard deviation of that accuracy for the best solution found by the PSO, for each of the eight tuning datasets and each of the four PSO algorithms. The average and standard deviation of the accuracy were calculated across 10 independent runs of the PSO algorithm and the accuracy reported for each such run was itself calculated by 10 repeated calculations using 10-fold cross validation with different splits of the dataset. The second table in each section lists the average number of features found and the standard deviation for the same datasets and PSO algorithms.

It is important to keep in mind that the results shown in this section are *not* representative of the performance of the four PSO algorithms on the eight tuning datasets, but instead will probably *overstate* that performance: the accuracies shown are those for that parameter combination out of 128 possible combinations that performed best on average across the same datasets. Hence the results are designed to be the best possible ones to be achieved by the PSO algorithms, rather than what would be expected if the PSO algorithms were tuned in another manner.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.2.1 GNB classifier

Table C.10: Detailed accuracy results for the GNB classifier on the FSP tuning datasets achieved by the chosen PSO parameter combination.

	BPSO			CFBPSO			PBPSO			SBPSO			All features	
Dataset	Avg.	± Stdev	Rank	Avg.	± Stdev	Rank	Avg.	± Stdev	Rank	Avg.	± Stdev	Rank	Avg.	± Stdev
echocardiogram	94.06%	± 0.23%	(1.5)	94.04%	± 0.19%	(3)	94.01%	± 0.14%	(4)	94.06%	± 0.24%	(1.5)	84.52%	± 0.53%
hepatitis	88.13%	± 0.64%	(4)	88.45%	± 0.62%	(1)	88.21%	± 0.50%	(2)	88.19%	± 0.69%	(3)	83.06%	± 0.81%
labor	95.07%	± 0.50%	(2)	94.85%	± 0.24%	(4)	95.01%	± 0.30%	(3)	95.11%	± 0.29%	(1)	42.73%	± 1.59%
lung-cancer	76.10%	± 1.60%	(3)	76.65%	± 1.77%	(2)	73.18%	± 2.80%	(4)	79.45%	± 2.31%	(1)	30.58%	± 3.31%
lymphography	85.42%	± 0.17%	(1.5)	85.22%	± 0.21%	(4)	85.40%	± 0.13%	(3)	85.42%	± 0.17%	(1.5)	71.81%	± 3.16%
promoter	88.92%	± 0.56%	(3)	89.35%	± 0.43%	(2)	88.41%	± 0.37%	(4)	89.88%	± 1.34%	(1)	70.79%	± 0.99%
wine	97.20%	± 0.01%	(1.5)	97.20%	± 0.01%	(3.5)	97.20%	± 0.05%	(1.5)	97.20%	± 0.01%	(3.5)	92.92%	± 0.47%
zoo	96.71%	± 0.17%	(1)	96.54%	± 0.13%	(4)	96.64%	± 0.12%	(3)	96.69%	± 0.18%	(2)	90.08%	± 0.75%
average	90.20%		2.19	90.29%		2.94	89.76%		3.06	90.75%		1.81	70.81%	

Table C.11: Detailed feature selection results for the GNB classifier on the FSP tuning datasets achieved by the chosen PSO parameter combination.

	BPSO		CFBPSO		PBPSO		SBPSO		All features
Dataset	Avg.	± Stdev	Avg.	± Stdev	Avg.	± Stdev	Avg.	± Stdev	Average
echocardiogram	5.8	± 0.9	5.6	± 1.0	5.8	± 0.4	5.5	± 0.5	12
hepatitis	10.7	± 2.5	8.4	± 2.0	10.6	± 2.3	11.7	± 2.9	19
labor	7.2	± 0.8	6.7	± 0.7	7.3	± 0.7	7.3	± 1.2	16
lung-cancer	23.4	± 3.6	17.1	± 3.3	24.1	± 4.5	16.7	± 5.6	56
lymphography	10.1	± 1.0	9.4	± 1.1	10.2	± 0.6	9.5	± 1.8	18
promoter	30.4	± 3.4	25.9	± 1.4	30.8	± 2.5	29.1	± 6.1	57
wine	6.3	± 0.9	6.1	± 0.9	6.5	± 0.8	6.6	± 0.5	13
zoo	10.1	± 0.3	10.0	± 0.5	10.0	± 0.0	10.1	± 0.3	16

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.2.2 J48 classifier

Table C.12: Detailed accuracy results for the J48 classifier on the FSP tuning datasets achieved by the chosen PSO parameter combination.

	BPSO			CFBPSO			PBPSO			SBPSO			All features	
Dataset	Avg.	± Stdev	Rank	Avg.	± Stdev	Rank	Avg.	± Stdev	Rank	Avg.	± Stdev	Rank	Avg.	± Stdev
echocardiogram	92.75%	± 0.98%	(2)	92.80%	± 0.70%	(1)	92.64%	± 0.27%	(4)	92.65%	± 0.44%	(3)	87.43%	± 1.53%
hepatitis	86.99%	± 1.15%	(4)	87.49%	± 0.93%	(2)	87.29%	± 1.14%	(3)	87.68%	± 0.68%	(1)	83.26%	± 1.25%
labor	95.50%	± 0.81%	(2)	95.33%	± 0.70%	(3.5)	95.83%	± 0.88%	(1)	95.33%	± 0.70%	(3.5)	83.20%	± 2.01%
lung-cancer	71.75%	± 9.67%	(3)	73.25%	± 11.14%	(2)	71.50%	± 9.12%	(4)	74.50%	± 12.35%	(1)	41.42%	± 7.05%
lymphography	88.68%	± 1.63%	(2)	88.70%	± 1.61%	(1)	87.86%	± 1.40%	(4)	88.25%	± 1.31%	(3)	77.21%	± 2.34%
promoter	91.54%	± 0.76%	(3)	91.56%	± 0.72%	(2)	91.39%	± 1.17%	(4)	92.54%	± 1.66%	(1)	78.52%	± 2.88%
wine	98.10%	± 0.39%	(4)	98.31%	± 0.26%	(2)	98.32%	± 0.25%	(1)	98.16%	± 0.27%	(3)	93.30%	± 1.20%
zoo	98.09%	± 0.00%	(1.5)	98.09%	± 0.00%	(1.5)	97.82%	± 0.44%	(4)	97.89%	± 0.38%	(3)	92.38%	± 1.24%
average	90.20%		2.69	90.69%		1.88	90.33%		3.13	90.88%		2.31	79.59%	

Table C.13: Detailed feature selection results for the J48 classifier on the FSP tuning datasets achieved by the chosen PSO parameter combination.

	BPSO		CFBPSO		PBPSO		SBPSO		All features
Dataset	Avg.	± Stdev	Avg.	± Stdev	Avg.	± Stdev	Avg.	± Stdev	Average
echocardiogram	6.9	± 1.6	6.3	± 1.6	6.3	± 1.6	5.6	± 2.8	12
hepatitis	6.3	± 3.0	6.6	± 2.6	7.6	± 2.0	7.1	± 3.5	19
labor	6.4	± 1.6	5.8	± 1.3	8	± 0.7	8	± 1.6	16
lung-cancer	23	± 5.0	18.2	± 6.2	20.7	± 6.5	17.1	± 9.8	56
lymphography	8.5	± 1.4	7.8	± 1.2	8.6	± 1.7	7.7	± 1.8	18
promoter	23.8	± 4.7	24.2	± 4.0	28	± 4.4	18.3	± 6.2	57
wine	6.6	± 1.3	7.4	± 1.5	6.7	± 1.5	6.6	± 1.5	13
zoo	8.2	± 0.9	8.1	± 1.6	8.4	± 1.6	8.4	± 1.2	16

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.2.3 *k*-NN classifier

Table C.14: Detailed accuracy results for the *k*-NN classifier on the FSP tuning datasets achieved by the chosen PSO parameter combination.

	BPSO			CFBPSO			PBPSO			SBPSO			All features	
Dataset	Avg.	± Stdev	Rank	Avg.	± Stdev	Rank	Avg.	± Stdev	Rank	Avg.	± Stdev	Rank	Avg.	± Stdev
echocardiogram	95.26%	± 0.37%	(2)	95.07%	± 0.63%	(4)	95.35%	± 0.45%	(1)	95.23%	± 0.38%	(3)	90.95%	± 1.24%
hepatitis	90.51%	± 0.23%	(1)	90.28%	± 0.36%	(3)	90.30%	± 0.34%	(2)	90.20%	± 0.29%	(4)	83.97%	± 0.55%
labor	98.67%	± 0.70%	(3)	98.67%	± 0.70%	(3)	98.83%	± 0.81%	(1)	98.67%	± 0.70%	(3)	90.33%	± 1.13%
lung-cancer	84.25%	± 1.44%	(2)	82.50%	± 2.83%	(4)	83.92%	± 2.64%	(3)	85.92%	± 2.76%	(1)	53.83%	± 3.00%
lymphography	87.50%	± 0.95%	(1)	87.40%	± 0.88%	(2)	87.31%	± 0.66%	(4)	87.35%	± 0.92%	(3)	80.21%	± 1.62%
promoter	89.29%	± 0.63%	(3)	90.37%	± 0.97%	(2)	89.28%	± 1.78%	(4)	91.51%	± 1.92%	(1)	76.35%	± 2.56%
wine	99.33%	± 0.35%	(1)	99.06%	± 0.27%	(2.5)	99.06%	± 0.27%	(2.5)	99.00%	± 0.23%	(4)	96.31%	± 0.59%
zoo	99.07%	± 0.47%	(4)	99.53%	± 0.50%	(1)	99.15%	± 0.30%	(3)	99.26%	± 0.39%	(2)	95.55%	± 1.28%
average	90.20%		2.13	92.86%		2.69	92.90%		2.56	93.39%		2.63	83.44%	

Table C.15: Detailed feature selection results for the *k*-NN classifier on the FSP tuning datasets achieved by the chosen PSO parameter combination.

	BPSO		CFBPSO		PBPSO		SBPSO		All features
Dataset	Avg.	± Stdev	Avg.	± Stdev	Avg.	± Stdev	Avg.	± Stdev	Average
echocardiogram	5.1	± 1.1	5.4	± 1.3	4.7	± 1.2	3.7	± 0.8	12
hepatitis	11	± 2.6	12.3	± 1.7	11.9	± 1.5	10.9	± 2.0	19
labor	9.3	± 1.1	10.5	± 2.3	8.7	± 1.6	9	± 2.1	16
lung-cancer	25.4	± 5.4	29.2	± 4.8	26.4	± 4.2	17.6	± 5.5	56
lymphography	12.6	± 1.8	13	± 1.6	11.6	± 1.0	12.7	± 1.8	18
promoter	36.7	± 5.1	38.6	± 3.9	39.5	± 5.7	38.1	± 5.3	57
wine	7.4	± 1.3	8.7	± 1.5	8.5	± 1.4	8.5	± 0.8	13
zoo	12.2	± 1.6	12.5	± 1.1	11.4	± 1.6	12.4	± 1.7	16

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.3 Detailed FSP testing results

The main numerical experiments performed using a PSO wrapper method to solve the FSP were described in chapter 6 and an overview of the results using the tuned PSO algorithms was presented in section 6.6. This section of the appendix contains the detailed results of the experiments on the FSP testing datasets using the four tuned PSO algorithms: BPSO, CFBPSO, PBPSO, and SBPSO. For each of the three classifiers used in these experiments, a separate subsection is included below: the GNB classifier in subsection C.3.1, the J48 classifier in subsection C.3.2, and the *k*-NN classifier in subsection C.3.3. Each such subsection in turn contains three tables with detailed results on

1. the classification accuracy,
2. the statistical tests performed, and
3. the number of features selected by the PSO algorithms.

The table with the classification accuracy contains the average classification accuracy for all the 30 testing datasets for each of the four PSO algorithms. The classification accuracy is calculated using 10 repetitions of the 10-fold cross validation using only the features selected by each of the four PSO algorithms for that dataset respectively. The standard deviation of the accuracy across those 10 repetitions is listed in the column labeled “stdev”. The columns labeled “rank” denote the rank of the achieved accuracy compared to that of the other three PSO algorithms. The bottom line of the table shows the average classification accuracy, which is less informative, and the more important average rank achieved by each of the four PSO algorithms on the 30 testing datasets.

The section labeled “statistical test” contains a table with the results of two different sets of statistical tests. The first of these is the F-test that was discussed in section 6.6.1. The three right most columns of the table contain the F-statistic and p-value of the F-test, and whether the outcome showed a significant difference at a confidence level of $\alpha = 5\%$ labeled “Signif.”. The second set of tests uses the ranks of the average classification accuracy for the four PSO algorithms. The leftmost four columns show the ranks for all datasets, while columns five to eight show the same ranks, but only for those datasets for which the F-test showed a significant difference. The bottom lines of the table contain the Z-score, p-value, and Holm-α of the Iman-Davenport test. This is performed once using all 30 datasets, and once using only those datasets that are selected by the F-test.

The table with the number of features selected contains just that: the average number of features selected and the standard deviation across the 30 independent runs of each PSO algorithm.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.3.1 GNB classifier

C.3.1.1 Classification accuracy

Table C.16: Final accuracy results on the FSP using the GNB classifier

GNB	BPSO			CFBPSO			PBPSO			SBPSO		
Dataset	Average	Stdev	Rank	Average	Stdev	Rank	Average	Stdev	Rank	Average	Stdev	Rank
arrhythmia	65.60%	0.81%	(3)	65.97%	0.65%	(1)	64.11%	0.64%	(4)	65.71%	1.63%	(2)
audiology	76.55%	0.27%	(2)	76.57%	0.51%	(1)	75.66%	0.56%	(4)	76.23%	0.47%	(3)
australian	87.03%	0.12%	(4)	87.04%	0.15%	(2)	87.03%	0.19%	(3)	87.07%	0.12%	(1)
bands	74.36%	0.68%	(1)	74.35%	0.53%	(2)	73.85%	0.66%	(4)	74.08%	0.97%	(3)
breasttissue	67.47%	0.53%	(3)	67.40%	0.58%	(4)	67.50%	0.38%	(2)	67.57%	0.54%	(1)
corral	85.67%	0.68%	(3)	85.71%	0.82%	(2)	85.76%	0.85%	(1)	85.43%	0.91%	(4)
crx	73.54%	0.05%	(2)	73.56%	0.07%	(1)	73.54%	0.06%	(3)	73.54%	0.12%	(4)
dermatology	95.50%	0.43%	(1)	95.42%	0.43%	(3)	94.91%	0.59%	(4)	95.49%	0.60%	(2)
german	76.59%	0.15%	(2)	76.64%	0.13%	(1)	76.55%	0.17%	(3)	76.55%	0.25%	(4)
glass	47.60%	0.25%	(3)	47.66%	0.29%	(1)	47.54%	0.27%	(4)	47.63%	0.26%	(2)
hill-valley	50.51%	0.37%	(2)	50.47%	0.40%	(3)	50.46%	0.37%	(4)	50.66%	0.37%	(1)
horse-colic	89.48%	0.14%	(3)	89.52%	0.15%	(2)	89.23%	0.25%	(4)	89.54%	0.17%	(1)
ionosphere	68.98%	11.24%	(3)	72.67%	0.51%	(1)	65.43%	13.47%	(4)	70.25%	9.35%	(2)
iris	96.41%	0.11%	(4)	96.44%	0.11%	(3)	96.47%	0.14%	(1)	96.45%	0.11%	(2)
liver	59.94%	0.36%	(3)	59.93%	0.43%	(4)	60.03%	0.48%	(1)	59.98%	0.33%	(2)
monk-1	64.80%	0.32%	(4)	64.89%	0.34%	(1)	64.88%	0.35%	(2)	64.88%	0.31%	(3)
monk-2	67.13%	0.01%	(1)	67.13%	0.01%	(2)	67.13%	0.01%	(3)	67.13%	0.01%	(4)
movement-libras	39.49%	0.76%	(3)	40.43%	0.73%	(2)	37.50%	0.87%	(4)	41.67%	1.15%	(1)
musk-1	72.48%	0.81%	(3)	74.37%	0.72%	(2)	69.86%	2.74%	(4)	80.41%	1.20%	(1)
parity5-5	46.31%	0.35%	(4)	46.35%	0.28%	(1)	46.33%	0.35%	(2)	46.31%	0.37%	(3)
parkinsons	86.80%	0.37%	(3)	87.35%	0.39%	(1)	86.74%	0.27%	(4)	87.00%	0.58%	(2)
pima	76.50%	0.09%	(2)	76.49%	0.10%	(4)	76.50%	0.10%	(3)	76.52%	0.09%	(1)
sonar	82.62%	0.70%	(2)	82.24%	0.84%	(3)	81.70%	0.74%	(4)	82.98%	0.80%	(1)
soybean	78.00%	1.25%	(3)	79.55%	2.72%	(1)	78.16%	1.49%	(2)	77.85%	1.38%	(4)
spectf	79.42%	0.00%	(3)	79.42%	0.00%	(1)	79.42%	0.00%	(4)	79.42%	0.01%	(2)
tic-tac-toe	74.48%	0.07%	(3)	74.52%	0.11%	(2)	74.47%	0.10%	(4)	74.52%	0.11%	(1)
vehicle	59.54%	0.35%	(2)	59.55%	0.44%	(1)	59.43%	0.31%	(3)	59.33%	0.71%	(4)
vote	95.75%	0.09%	(4)	95.78%	0.07%	(1)	95.77%	0.08%	(3)	95.77%	0.09%	(2)
vowel	45.13%	0.18%	(2)	45.07%	0.15%	(3)	45.14%	0.18%	(1)	45.07%	0.17%	(4)
wdbc	96.45%	0.10%	(2)	96.44%	0.14%	(3)	96.39%	0.18%	(4)	96.49%	0.15%	(1)
average	72.67%		(2.67)	72.96%		(1.97)	72.25%		(3.10)	73.05%		(2.27)

© University of Pretoria

C.3.1.2 Statistical test

Table C.17: Statistical tests on final accuracy results on the FSP using the GNB classifier

GNB	All datasets				Selected datasets				F-test		
	BPSO	CFBPSO	PBPSO	SBPSO	BPSO	CFBPSO	PBPSO	SBPSO			$\alpha = 0.05$
Dataset	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	F-stat.	p-value	Signif.
arrhythmia	(3)	(1)	(4)	(2)	(3)	(1)	(4)	(2)	15.22	0.0000	**TRUE**
audiology	(2)	(1)	(4)	(3)	(2)	(1)	(4)	(3)	18.91	0.0000	**TRUE**
australian	(4)	(2)	(3)	(1)					0.40	0.7533	FALSE
bands	(1)	(2)	(4)	(3)					2.59	0.0580	FALSE
breasttissue	(3)	(4)	(2)	(1)					0.44	0.7264	FALSE
corral	(3)	(2)	(1)	(4)					0.70	0.5547	FALSE
crx	(2)	(1)	(3)	(4)					0.19	0.9005	FALSE
dermatology	(1)	(3)	(4)	(2)	(1)	(3)	(4)	(2)	6.78	0.0004	**TRUE**
german	(2)	(1)	(3)	(4)					1.24	0.3010	FALSE
glass	(3)	(1)	(4)	(2)					0.81	0.4915	FALSE
hill-valley	(2)	(3)	(4)	(1)					1.26	0.2931	FALSE
horse-colic	(3)	(2)	(4)	(1)	(3)	(2)	(4)	(1)	13.73	0.0000	**TRUE**
ionosphere	(3)	(1)	(4)	(2)					2.07	0.1095	FALSE
iris	(4)	(3)	(1)	(2)					1.18	0.3208	FALSE
liver	(3)	(4)	(1)	(2)					0.29	0.8304	FALSE
monk-1	(4)	(1)	(2)	(3)					0.37	0.7700	FALSE
monk-2	(1)	(2)	(3)	(4)					0.06	0.9809	FALSE
movement-libras	(3)	(2)	(4)	(1)	(3)	(2)	(4)	(1)	87.88	0.0000	**TRUE**
musk-1	(3)	(2)	(4)	(1)	(3)	(2)	(4)	(1)	179.37	0.0000	**TRUE**
parity5-5	(4)	(1)	(2)	(3)					0.06	0.9802	FALSE
parkinsons	(3)	(1)	(4)	(2)	(3)	(1)	(4)	(2)	9.73	0.0000	**TRUE**
pima	(2)	(4)	(3)	(1)					0.35	0.7873	FALSE
sonar	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	11.24	0.0000	**TRUE**
soybean	(3)	(1)	(2)	(4)	(3)	(1)	(2)	(4)	4.21	0.0079	**TRUE**
spectf	(3)	(1)	(4)	(2)					0.39	0.7585	FALSE
tic-tac-toe	(3)	(2)	(4)	(1)					1.52	0.2141	FALSE
vehicle	(2)	(1)	(3)	(4)					1.04	0.3805	FALSE
vote	(4)	(1)	(3)	(2)					0.69	0.5578	FALSE
vowel	(2)	(3)	(1)	(4)					1.11	0.3505	FALSE
wdbc	(2)	(3)	(4)	(1)					1.81	0.1514	FALSE
average rank	(2.67)	**(1.97)**	(3.10)	**(2.27)**	**(2.56)**	**(1.78)**	(3.78)	**(1.89)**	# datasets		9
rank of rank	(3)	(1)	(4)	(2)	(3)	(1)	(4)	(2)			
Z-score	2.1000		3.4000	0.9000	0.8520		2.1909	0.1217			
p-value	0.0179		0.0003	0.1841	0.1971		0.0142	0.4516			
Holm α	0.0250		0.0500	0.0167	0.0250		0.0500	0.0167			

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.3.1.3 Number of features selected

Table C.18: Number of features selected using the GNB classifier

GNB		BPSO		CFBPSO		PBPSO		SBPSO	
Dataset	# Ftrs	Average	Stdev	Average	Stdev	Average	Stdev	Average	Stdev
arrhythmia	279	137.1	12.4	98.2	9.5	159.4	9.9	137.7	36.0
audiology	69	34.3	2.7	29.5	3.1	39.8	3.8	41.6	3.7
australian	14	9.0	1.1	8.5	1.2	9.0	1.0	9.4	0.7
bands	39	19.8	2.1	16.5	2.4	22.0	2.0	21.0	3.6
breasttissue	9	5.0	0.0	5.0	0.0	5.0	0.0	5.0	0.0
corral	6	4.0	0.0	4.0	0.2	4.1	0.3	4.2	0.4
crx	15	4.5	0.5	4.2	0.4	4.9	0.6	4.6	0.8
dermatology	34	16.7	1.6	14.7	1.5	17.8	1.7	18.0	1.5
german	24	10.4	1.2	10.0	1.2	10.8	1.2	11.3	1.6
glass	9	3.5	0.9	3.1	1.0	3.3	1.0	3.1	1.0
hill-valley	100	53.8	10.1	47.9	14.0	53.5	7.2	56.0	16.8
horse-colic	36	11.2	1.2	9.6	1.2	13.2	1.9	11.0	1.5
ionosphere	34	4.9	3.5	2.9	1.6	5.6	6.2	4.7	4.9
iris	4	2.0	0.0	2.0	0.0	2.0	0.0	2.0	0.0
liver	6	2.1	0.4	2.0	0.0	2.2	0.5	2.1	0.4
monk-1	6	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
monk-2	6	2.9	1.3	2.4	1.4	3.0	1.2	3.4	1.4
movement-libras	90	31.3	3.6	23.2	3.5	37.6	6.0	23.5	6.4
musk-1	166	52.3	5.8	35.9	5.5	49.3	21.7	24.2	7.8
parity5-5	10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
parkinsons	22	9.6	2.4	5.9	1.5	10.3	2.3	9.5	3.2
pima	8	2.0	0.0	2.0	0.0	2.0	0.0	2.0	0.0
sonar	60	24.5	2.7	20.6	3.3	28.6	2.6	21.7	5.3
soybean	35	21.0	2.2	19.0	2.6	21.9	2.4	21.9	2.1
spectf	44	22.2	7.5	19.9	7.5	23.2	5.0	21.0	7.8
tic-tac-toe	9	9.0	0.0	9.0	0.0	9.0	0.0	9.0	0.0
vehicle	18	9.2	0.5	9.1	0.4	9.3	0.5	9.3	0.8
vote	16	4.6	0.6	4.4	0.7	4.7	0.8	4.5	1.0
vowel	10	4.0	0.0	4.0	0.0	4.0	0.0	4.0	0.0
wdbc	30	9.0	1.3	8.3	1.4	9.6	1.2	8.3	1.6

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.3.2 J48 classifier

C.3.2.1 Classification accuracy

Table C.19: Final accuracy results on the FSP using the J48 classifier

J48	BPSO			CFBPSO			PBPSO			SBPSO		
Dataset	Average	Stdev	Rank	Average	Stdev	Rank	Average	Stdev	Rank	Average	Stdev	Rank
arrhythmia	73.49%	0.82%	(2)	73.66%	0.62%	(1)	73.32%	0.79%	(3)	73.17%	0.90%	(4)
audiology	79.82%	0.80%	(1)	79.81%	0.72%	(2)	79.48%	0.90%	(4)	79.60%	0.68%	(3)
australian	87.19%	0.52%	(1)	87.12%	0.68%	(3)	87.15%	0.45%	(2)	87.09%	0.52%	(4)
bands	79.33%	1.12%	(2)	79.43%	1.23%	(1)	79.07%	1.11%	(4)	79.15%	1.46%	(3)
breasttissue	69.79%	1.50%	(3)	70.27%	1.31%	(1)	70.24%	1.30%	(2)	69.38%	1.78%	(4)
corral	98.90%	0.73%	(3)	98.91%	0.66%	(2)	98.88%	0.70%	(4)	98.97%	0.62%	(1)
crx	74.10%	0.54%	(4)	74.14%	0.53%	(3)	74.15%	0.51%	(2)	74.32%	0.65%	(1)
dermatology	97.38%	0.13%	(1)	97.30%	0.20%	(3)	97.25%	0.21%	(4)	97.30%	0.22%	(2)
german	74.20%	0.76%	(1)	74.04%	0.66%	(4)	74.19%	0.66%	(2)	74.09%	0.72%	(3)
glass	73.51%	1.82%	(2)	73.21%	2.13%	(4)	73.39%	1.82%	(3)	73.57%	1.35%	(1)
hill-valley	46.62%	0.78%	(3)	50.13%	0.42%	(1)	46.60%	0.74%	(4)	47.13%	1.36%	(2)
horse-colic	88.59%	0.76%	(4)	88.63%	0.66%	(2)	88.62%	0.56%	(3)	88.69%	0.41%	(1)
ionosphere	92.45%	0.98%	(3)	92.63%	0.65%	(1)	92.43%	0.74%	(4)	92.53%	1.10%	(2)
iris	94.64%	0.41%	(3)	94.58%	0.36%	(4)	94.76%	0.25%	(1)	94.71%	0.31%	(2)
liver	68.84%	1.63%	(3)	68.70%	1.43%	(4)	68.90%	1.40%	(2)	68.95%	1.29%	(1)
monk-1	75.01%	0.00%	(2)	75.01%	0.00%	(4)	75.01%	0.00%	(3)	75.01%	0.03%	(1)
monk-2	89.81%	0.54%	(1)	89.79%	0.88%	(2)	89.72%	0.69%	(4)	89.74%	0.79%	(3)
movement-libras	73.27%	1.02%	(2)	73.68%	1.09%	(1)	72.84%	1.27%	(3)	72.10%	1.28%	(4)
musk-1	86.15%	1.21%	(2)	86.28%	0.83%	(1)	85.63%	0.94%	(3)	85.55%	1.00%	(4)
parity5-5	96.64%	0.52%	(4)	97.10%	0.45%	(1)	97.05%	0.61%	(2)	96.86%	0.45%	(3)
parkinsons	90.21%	1.68%	(2)	89.96%	1.71%	(3)	89.73%	1.64%	(4)	90.21%	1.56%	(1)
pima	75.11%	0.41%	(3)	75.16%	0.41%	(1)	75.09%	0.50%	(4)	75.12%	0.40%	(2)
sonar	82.36%	1.67%	(1)	81.91%	1.33%	(3)	80.81%	1.66%	(4)	81.95%	2.33%	(2)
soybean	95.25%	0.29%	(2)	95.20%	0.32%	(3)	95.11%	0.27%	(4)	95.33%	0.27%	(1)
spectf	80.60%	1.93%	(2)	80.42%	1.85%	(3)	80.24%	1.71%	(4)	81.21%	1.92%	(1)
tic-tac-toe	93.25%	0.56%	(4)	93.45%	0.50%	(2)	93.44%	0.58%	(3)	93.48%	0.46%	(1)
vehicle	72.89%	0.51%	(2)	72.94%	0.49%	(1)	72.76%	0.65%	(4)	72.87%	0.43%	(3)
vote	96.36%	0.15%	(1)	96.34%	0.22%	(2)	96.33%	0.23%	(3)	96.31%	0.21%	(4)
vowel	79.13%	0.52%	(3)	79.20%	0.42%	(2)	79.21%	0.53%	(1)	79.06%	0.41%	(4)
wdbc	95.65%	0.55%	(4)	95.81%	0.62%	(2)	95.66%	0.59%	(3)	95.90%	0.63%	(1)
average	82.68%		(2.37)	82.83%		(2.23)	82.57%		(3.10)	82.65%		(2.30)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.3.2.2 Statistical test

Table C.20: Statistical tests on final accuracy results on the FSP using the J48 classifier

J48	All datasets				Selected datasets				*F*-test		
	BPSO	CFBPSO	PBPSO	SBPSO	BPSO	CFBPSO	PBPSO	SBPSO			$\alpha = 0.05$
Dataset	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	*F*-stat.	*p*-value	Signif.
arrhythmia	(2)	(1)	(3)	(4)					1.66	0.1812	FALSE
audiology	(1)	(2)	(4)	(3)					1.03	0.3852	FALSE
australian	(1)	(3)	(2)	(4)					0.12	0.9461	FALSE
bands	(2)	(1)	(4)	(3)					0.40	0.7520	FALSE
breasttissue	(3)	(1)	(2)	(4)					1.79	0.1546	FALSE
corral	(3)	(2)	(4)	(1)					0.08	0.9699	FALSE
crx	(4)	(3)	(2)	(1)					0.69	0.5628	FALSE
dermatology	(1)	(3)	(4)	(2)					1.61	0.1921	FALSE
german	(1)	(4)	(2)	(3)					0.26	0.8530	FALSE
glass	(2)	(4)	(3)	(1)					0.18	0.9103	FALSE
hill-valley	(3)	(1)	(4)	(2)	(3)	(1)	(4)	(2)	80.65	0.0000	**TRUE**
horse-colic	(4)	(2)	(3)	(1)					0.12	0.9454	FALSE
ionosphere	(3)	(1)	(4)	(2)					0.23	0.8752	FALSE
iris	(3)	(4)	(1)	(2)					1.31	0.2755	FALSE
liver	(3)	(4)	(2)	(1)					0.13	0.9402	FALSE
monk-1	(2)	(4)	(3)	(1)					0.80	0.5000	FALSE
monk-2	(1)	(2)	(4)	(3)					0.08	0.9695	FALSE
movement-libras	(2)	(1)	(3)	(4)	(2)	(1)	(3)	(4)	7.52	0.0002	**TRUE**
musk-1	(2)	(1)	(3)	(4)	(2)	(1)	(3)	(4)	2.96	0.0368	**TRUE**
parity5-5	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	3.73	0.0143	**TRUE**
parkinsons	(2)	(3)	(4)	(1)					0.43	0.7294	FALSE
pima	(3)	(1)	(4)	(2)					0.09	0.9628	FALSE
sonar	(1)	(3)	(4)	(2)	(1)	(3)	(4)	(2)	3.10	0.0308	**TRUE**
soybean	(2)	(3)	(4)	(1)					2.49	0.0657	FALSE
spectf	(2)	(3)	(4)	(1)					1.17	0.3242	FALSE
tic-tac-toe	(4)	(2)	(3)	(1)					0.87	0.4580	FALSE
vehicle	(2)	(1)	(4)	(3)					0.49	0.6932	FALSE
vote	(1)	(2)	(3)	(4)					0.26	0.8530	FALSE
vowel	(3)	(2)	(1)	(4)					0.47	0.7019	FALSE
wdbc	(4)	(2)	(3)	(1)					0.88	0.4522	FALSE
average rank	**(2.37)**	**(2.23)**	(3.10)	**(2.30)**	**(2.40)**	**(1.40)**	(3.20)	**(3.00)**	# datasets selected		5
rank of rank	(3)	(1)	(4)	(2)	(2)	(1)	(4)	(3)			
Z-score	0.4000		2.6000	0.2000	1.0954		1.9718	1.7527			
p-value	0.3446		0.0047	0.4207	0.1367		0.0243	0.0398			
Holm α	0.0250		0.0500	0.0167	0.0167		0.0500	0.0250			

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.3.2.3 Number of features selected

Table C.21: Number of features selected using the J48 classifier

J48		BPSO		CFBPSO		PBPSO		SBPSO	
Dataset	# Ftrs	Average	Stdev	Average	Stdev	Average	Stdev	Average	Stdev
arrhythmia	279	125.2	15.9	123.2	14.2	126.8	14.6	160.5	27.0
audiology	69	34.2	4.1	37.3	5.7	35.3	6.6	43.6	6.3
australian	14	7.0	1.4	7.2	1.6	7.6	2.1	8.0	1.8
bands	39	18.3	3.1	17.9	3.5	17.4	3.0	20.0	5.2
breasttissue	9	4.5	1.1	4.4	0.8	4.4	0.8	4.6	1.1
corral	6	4.6	0.5	4.5	0.5	4.6	0.5	4.8	0.4
crx	15	7.8	2.3	8.3	2.2	8.1	1.7	8.4	1.8
dermatology	34	17.3	2.4	17.8	2.0	17.3	2.4	19.5	2.9
german	24	11.8	3.1	12.4	3.1	11.5	2.8	13.8	4.0
glass	9	5.2	0.8	5.2	0.8	5.1	0.8	5.4	0.9
hill-valley	100	46.2	12.1	0.0	0.0	46.6	11.0	47.7	28.5
horse-colic	36	18.1	3.1	19.5	3.1	18.6	3.2	24.8	3.2
ionosphere	34	14.8	3.5	14.3	3.4	14.1	3.3	14.6	4.1
iris	4	2.9	0.6	3.0	0.6	2.9	0.6	2.8	0.7
liver	6	4.2	1.1	3.9	1.3	4.2	1.0	4.1	1.0
monk-1	6	3.9	1.1	3.5	1.1	3.5	1.1	3.3	1.2
monk-2	6	6.0	0.0	6.0	0.0	6.0	0.0	6.0	0.0
movement-libras	90	40.1	7.9	38.5	4.9	41.6	6.4	52.7	12.5
musk-1	166	82.6	14.7	83.7	11.5	79.7	12.9	104.0	19.1
parity5-5	10	5.4	1.3	5.6	0.6	5.6	0.7	5.8	0.7
parkinsons	22	7.4	2.6	7.3	2.6	8.1	2.1	6.6	2.7
pima	8	5.7	1.1	5.7	1.1	5.5	1.3	6.3	1.1
sonar	60	22.0	3.5	22.5	4.8	24.7	4.1	24.4	6.9
soybean	35	21.3	2.2	22.5	1.3	21.7	2.4	23.1	2.8
spectf	44	15.4	3.5	14.3	4.1	17.2	4.2	10.9	4.6
tic-tac-toe	9	9.0	0.0	9.0	0.0	9.0	0.0	9.0	0.0
vehicle	18	13.2	1.5	13.6	1.6	12.7	1.8	14.4	2.2
vote	16	8.5	1.6	9.6	1.9	9.2	1.7	10.8	2.0
vowel	10	8.2	1.1	8.7	1.1	8.4	1.2	8.6	1.0
wdbc	30	11.8	3.2	11.2	2.3	12.7	2.4	11.6	3.6

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.3.3 *k*-NN classifier

C.3.3.1 Classification accuracy

Table C.22: Final accuracy results on the FSP using the *k*-NN classifier

k-NN	BPSO			CFBPSO			PBPSO			SBPSO		
Dataset	Average	Stdev	Rank	Average	Stdev	Rank	Average	Stdev	Rank	Average	Stdev	Rank
arrhythmia	65.24%	1.09%	(4)	66.24%	1.13%	(2)	65.49%	1.14%	(3)	70.06%	3.12%	(1)
audiology	83.00%	0.70%	(4)	83.33%	0.58%	(2)	83.15%	0.64%	(3)	83.68%	0.56%	(1)
australian	87.41%	0.39%	(3)	87.44%	0.34%	(2)	87.23%	0.38%	(4)	87.47%	0.50%	(1)
bands	81.54%	3.07%	(2)	81.05%	3.50%	(3)	80.77%	4.02%	(4)	82.03%	4.24%	(1)
breasttissue	70.86%	1.09%	(2)	70.67%	1.51%	(4)	70.75%	1.18%	(3)	71.02%	1.03%	(1)
corral	99.24%	1.87%	(3)	99.63%	0.87%	(1)	99.43%	1.35%	(2)	99.23%	1.34%	(4)
crx	71.01%	1.16%	(3)	71.10%	1.10%	(2)	70.96%	1.11%	(4)	71.24%	1.16%	(1)
dermatology	98.13%	0.34%	(2)	98.02%	0.38%	(3)	97.97%	0.41%	(4)	98.18%	0.25%	(1)
german	76.38%	0.74%	(4)	76.55%	0.80%	(2)	76.44%	0.64%	(3)	77.00%	0.73%	(1)
glass	77.40%	2.18%	(2)	77.59%	1.94%	(1)	77.19%	2.38%	(4)	77.27%	2.18%	(3)
hill-valley	62.55%	0.65%	(3)	62.46%	0.68%	(4)	62.63%	0.67%	(2)	63.51%	1.16%	(1)
horse-colic	88.12%	0.51%	(2)	87.65%	0.56%	(4)	87.79%	0.56%	(3)	88.16%	0.66%	(1)
ionosphere	93.75%	0.56%	(3)	93.71%	0.62%	(4)	93.76%	0.74%	(2)	94.43%	0.46%	(1)
iris	96.16%	0.42%	(4)	96.21%	0.44%	(1)	96.17%	0.37%	(2)	96.17%	0.39%	(3)
liver	67.87%	1.02%	(2)	67.42%	1.27%	(4)	67.73%	1.22%	(3)	67.96%	1.16%	(1)
monk-1	100.00%	0.00%	(2.5)	100.00%	0.00%	(2.5)	100.00%	0.00%	(2.5)	100.00%	0.00%	(2.5)
monk-2	82.70%	0.53%	(4)	82.86%	0.46%	(1)	82.79%	0.45%	(3)	82.83%	0.50%	(2)
movement-libras	88.91%	0.50%	(4)	88.98%	0.39%	(3)	88.98%	0.55%	(2)	89.51%	0.65%	(1)
musk-1	93.99%	0.86%	(2)	93.68%	1.20%	(3)	93.28%	1.32%	(4)	95.16%	1.09%	(1)
parity5-5	98.33%	9.09%	(2)	99.99%	0.03%	(1)	96.59%	12.93%	(4)	98.32%	9.13%	(3)
parkinsons	99.29%	0.18%	(2)	99.30%	0.16%	(1)	99.28%	0.20%	(3)	99.19%	0.27%	(4)
pima	74.93%	0.65%	(4)	74.99%	0.50%	(3)	75.07%	0.50%	(2)	75.19%	0.38%	(1)
sonar	92.25%	1.81%	(2)	91.70%	2.03%	(4)	91.74%	1.71%	(3)	92.62%	1.97%	(1)
soybean	94.48%	0.26%	(2)	94.47%	0.30%	(3)	94.46%	0.43%	(4)	94.50%	0.33%	(1)
spectf	83.00%	1.40%	(4)	83.17%	0.84%	(2)	83.08%	1.53%	(3)	84.00%	1.16%	(1)
tic-tac-toe	90.47%	0.26%	(3)	90.51%	0.24%	(2)	90.46%	0.27%	(4)	90.56%	0.22%	(1)
vehicle	73.72%	0.61%	(2)	73.49%	0.67%	(4)	73.59%	0.50%	(3)	73.77%	0.58%	(1)
vote	96.25%	0.31%	(1)	96.13%	0.38%	(3)	96.08%	0.40%	(4)	96.17%	0.48%	(2)
vowel	99.11%	0.07%	(1)	99.09%	0.08%	(2)	99.08%	0.09%	(3)	99.08%	0.09%	(4)
wdbc	97.60%	0.19%	(4)	97.67%	0.18%	(1)	97.65%	0.24%	(2)	97.63%	0.20%	(3)
average	86.12%		(2.75)	86.17%		(2.48)	85.99%		(3.08)	86.53%		(1.68)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.3.3.2 Statistical test

Table C.23: Statistical tests on final accuracy results on the FSP using the k-NN classifier

k-NN	All datasets				Selected datasets				F-test		
	BPSO	CFBPSO	PBPSO	SBPSO	BPSO	CFBPSO	PBPSO	SBPSO			$\alpha = 0.05$
Dataset	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	(Rank)	F-stat.	p-value	Signif.
arrhythmia	(4)	(2)	(3)	(1)	(4)	(2)	(3)	(1)	33.65	0.0000	**TRUE**
audiology	(4)	(2)	(3)	(1)	(4)	(2)	(3)	(1)	4.88	0.0035	**TRUE**
australian	(3)	(2)	(4)	(1)					1.56	0.2045	FALSE
bands	(2)	(3)	(4)	(1)					0.50	0.6841	FALSE
breasttissue	(2)	(4)	(3)	(1)					0.34	0.7979	FALSE
corral	(3)	(1)	(2)	(4)					0.42	0.7424	FALSE
crx	(3)	(2)	(4)	(1)					0.26	0.8541	FALSE
dermatology	(2)	(3)	(4)	(1)					1.74	0.1657	FALSE
german	(4)	(2)	(3)	(1)	(4)	(2)	(3)	(1)	3.32	0.0236	**TRUE**
glass	(2)	(1)	(4)	(3)					0.15	0.9313	FALSE
hill-valley	(3)	(4)	(2)	(1)	(3)	(4)	(2)	(1)	7.96	0.0001	**TRUE**
horse-colic	(2)	(4)	(3)	(1)	(2)	(4)	(3)	(1)	4.24	0.0076	**TRUE**
ionosphere	(3)	(4)	(2)	(1)	(3)	(4)	(2)	(1)	7.41	0.0002	**TRUE**
iris	(4)	(1)	(2)	(3)					0.07	0.9736	FALSE
liver	(2)	(4)	(3)	(1)					0.92	0.4361	FALSE
monk-1	(2.5)	(2.5)	(2.5)	(2.5)							FALSE
monk-2	(4)	(1)	(3)	(2)					0.45	0.7193	FALSE
movement-libras	(4)	(3)	(2)	(1)	(4)	(3)	(2)	(1)	6.34	0.0006	**TRUE**
musk-1	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	11.64	0.0000	**TRUE**
parity5-5	(2)	(1)	(4)	(3)					0.52	0.6705	FALSE
parkinsons	(2)	(1)	(3)	(4)					1.46	0.2312	FALSE
pima	(4)	(3)	(2)	(1)					1.03	0.3837	FALSE
sonar	(2)	(4)	(3)	(1)					1.23	0.3043	FALSE
soybean	(2)	(3)	(4)	(1)					0.05	0.9829	FALSE
spectf	(4)	(2)	(3)	(1)	(4)	(2)	(3)	(1)	3.07	0.0318	**TRUE**
tic-tac-toe	(3)	(2)	(4)	(1)					0.64	0.5889	FALSE
vehicle	(2)	(4)	(3)	(1)					1.03	0.3822	FALSE
vote	(1)	(3)	(4)	(2)					0.73	0.5381	FALSE
vowel	(1)	(2)	(3)	(4)					0.78	0.5095	FALSE
wdbc	(4)	(1)	(2)	(3)					0.43	0.7350	FALSE
average rank	(2.75)	(2.48)	(3.08)	**(1.68)**	(3.33)	(2.89)	**(2.78)**	**(1.00)**	# datasets selected		9
rank of rank	(3)	(2)	(4)	(1)	(4)	(3)	(2)	(1)			
Z-score	3.2000	2.4000	4.2000		2.5560	2.0692	1.9475				
p-value	0.0007	0.0082	0.0000		0.0053	0.0193	0.0257				
Holm α	0.0250	0.0167	0.0500		0.0500	0.0250	0.0167				

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C.3.3.3 Number of features selected

Table C.24: Number of features selected using the *k*-NN classifier

k-NN		BPSO		CFBPSO		PBPSO		SBPSO	
Dataset	# Ftrs	Average	Stdev	Average	Stdev	Average	Stdev	Average	Stdev
arrhythmia	279	119.7	17.5	100.9	36.7	119.3	19.4	65.6	43.4
audiology	69	41.7	3.2	45.6	3.3	45.3	3.3	45.4	4.7
australian	14	6.5	1.0	6.8	1.1	7.2	1.1	6.3	1.3
bands	39	20.0	3.0	23.5	5.2	21.7	4.9	17.4	7.7
breasttissue	9	6.6	1.3	6.7	1.4	5.9	1.3	5.8	1.5
corral	6	4.2	0.4	4.1	0.3	4.2	0.4	4.3	0.4
crx	15	8.8	2.2	8.9	1.8	9.4	2.1	7.7	1.9
dermatology	34	23.1	2.9	25.7	3.2	24.5	2.4	24.6	2.9
german	24	11.4	2.0	12.1	1.9	11.8	1.9	9.0	2.0
glass	9	6.1	0.7	6.4	0.6	6.4	0.7	6.0	0.8
hill-valley	100	47.9	5.6	51.1	7.1	47.0	5.2	40.4	11.6
horse-colic	36	22.0	3.9	24.4	3.4	22.7	4.1	22.2	5.2
ionosphere	34	11.0	2.0	11.0	2.2	10.4	2.4	7.6	1.6
iris	4	3.3	0.9	3.3	0.9	3.2	1.0	3.3	0.9
liver	6	3.5	0.9	3.5	0.9	3.6	0.9	3.5	0.8
monk-1	6	3.8	0.4	3.8	0.4	3.8	0.4	3.9	0.3
monk-2	6	6.0	0.0	6.0	0.0	6.0	0.0	6.0	0.0
movement-libras	90	44.5	6.3	46.2	6.4	44.6	5.4	38.6	8.9
musk-1	166	84.1	7.3	94.2	5.9	85.4	9.9	69.9	14.3
parity5-5	10	6.6	1.3	6.8	0.8	6.2	1.6	6.6	1.3
parkinsons	22	12.9	1.4	13.4	1.7	13.4	1.3	12.9	1.2
pima	8	5.6	1.2	5.6	1.2	5.7	1.2	5.1	1.4
sonar	60	29.4	3.2	33.2	3.5	29.4	4.1	25.9	4.6
soybean	35	22.3	1.9	24.0	1.7	22.4	2.0	22.1	2.0
spectf	44	18.2	4.2	19.3	3.6	18.2	4.0	12.8	3.0
tic-tac-toe	9	9.0	0.0	9.0	0.0	9.0	0.0	9.0	0.0
vehicle	18	12.2	1.5	12.6	1.2	12.1	1.6	11.8	1.4
vote	16	5.9	1.1	6.7	1.6	6.5	1.6	5.8	1.7
vowel	10	9.0	0.0	9.0	0.0	9.0	0.0	9.0	0.0
wdbc	30	20.2	2.1	21.2	2.6	20.6	2.1	20.7	2.3

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Appendix D

Description of datasets used for the FSP

This appendix contains a description of all datasets used in the experiments in this thesis. Table D.1 lists the datasets alphabetically, noting the number of instances classes, and features. The number of features is further split into nominal and numerical features. Also listed is whether the dataset was used during the tuning of the PSO algorithms, or during testing instead.

The separate sections below contain a description of the various datasets used. Each section is labeled with the full dataset name followed by the shorthand name used in the rest of the text between parentheses. More detailed information on the features and classes present in each dataset can be found at the UCI machine learning repository [5] on-line. For this purpose a link to the page dedicated to the dataset is listed at the end of each section. Where possible these descriptions were copied directly from the description provided at the UCI machine learning repository. Note that two of the datasets used can no longer be found on the UCI machine learning repository. For these datasets, a different URL is provided. The datasets no longer available in the UCI machine learning repository are corral and parity5-5.

Also listed are any actions taken or alterations made to the dataset for the experiments in this thesis that are not covered by the standard data preprocessing described in section 6.2.2. Most often this means that a separate training and testing set provided at the UCI Machine Learning repository was combined to form a single set of instances. This was done because the *k*-fold cross validation process used repeatedly splits the dataset into training and testing sets, removing the need to have these sets separated beforehand.

Cardiac Arrhythmia Database (arrhythmia)

Concerning the original study:

> The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups. Class 01 refers to "normal" ECG classes, 02 to 15 refers to different classes of arrhythmia, and class 16 refers to the rest of unclassified ones. For the time being, there exists a computer program that makes such a classification. However, there are differences between the cardiologist's and the programs classification. Taking the cardiologist's as a gold standard we aim to minimize this difference by means of machine learning tools.

`http://archive.ics.uci.edu/ml/datasets/arrhythmia`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table D.1: Note that the information in this table is an amalgamation of that in tables 6.1 and 6.2.

dataset	# instances	# classes	# features	# numerical	# nominal	Use
arrhythmia	452	13	279	267	12	testing
audiology	226	24	69	0	69	testing
australian	690	2	14	8	6	testing
bands	540	2	39	20	19	testing
breasttissue	106	6	9	9	0	testing
corral	64	2	6	0	6	testing
crx	690	2	15	6	9	testing
dermatology	366	6	34	1	33	testing
echocardiogram	132	2	12	12	0	tuning
german	1000	2	24	7	13	testing
glass	214	7	9	9	0	testing
hepatitis	155	2	19	19	0	tuning
hill-valley	1212	2	100	100	0	testing
horse-colic	368	2	36	7	29	testing
ionosphere	351	2	34	34	0	testing
iris	150	3	4	4	0	testing
labor	57	2	16	8	8	tuning
liver	345	2	6	6	0	testing
lung-cancer	32	3	56	0	56	tuning
lymphography	148	4	18	3	15	tuning
monk-1	432	2	6	6	0	testing
monk-2	432	2	6	6	0	testing
movement-libras	360	15	90	90	0	testing
musk-1	476	2	166	166	0	testing
parity5-5	1024	2	10	0	10	testing
parkinsons	195	2	22	22	0	testing
pima	768	2	8	8	0	testing
promoter	106	2	57	0	57	tuning
sonar	208	2	60	60	0	testing
soybean	683	19	35	0	35	testing
spectf	267	2	44	44	0	testing
tic-tac-toe	958	2	9	9	2	testing
vehicle	847	5	18	18	0	testing
vote	435	2	16	0	16	testing
vowel	990	11	10	10	0	testing
wdbc	569	2	30	30	0	testing
wine	178	3	13	13	0	tuning
zoo	101	7	16	0	16	tuning

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Audiology Database - Standardized (audiology)

The dataset stems from the domain of clinical audiology which involves the evaluation and diagnosis of hearing disorders. The dataset contains sequential cases from Baylor College of Medicine. Each instance contains a hearing disorder diagnosis, plus information on patient-reported symptoms, patient history information, and the results of routine tests. The latter is the data that a clinician considers when diagnosing a patient.

The database used is audiology.standardized, a standardized version of the original dataset. Note that in the experiments in this thesis the original training set of 200 instances and test set of 26 instances were combined into a single dataset of 226 instances.

`http://archive.ics.uci.edu/ml/datasets/Audiology+%28Standardized%29`

Australian Credit Approval (australian)

This dataset concerns credit card applications. All feature names and values have been changed to meaningless symbols to protect confidentiality of the data. This dataset is interesting because there is a good mix of features – continuous, nominal with small numbers of values, and nominal with larger numbers of values. There are also a few missing values.

`http://archive.ics.uci.edu/ml/datasets/australian`

Cylinder Bands (bands)

The abstract from the original study reads as:

> Machine learning tools show significant promise for knowledge acquisition, particularly when human expertise is inadequate. Recently, process delays known as cylinder banding in rotogravure printing were substantially mitigated using control rules discovered by decision tree induction. Our work exemplifies a more general methodology which transforms the knowledge acquisition task from one in which rules are directly elicited from an expert, to one in which a learning system is responsible for rule generation. The primary responsibilities of the human expert are to evaluate the merits of generated rules, and to guide the acquisition and classification of data necessary for machine induction. These responsibilities require the expert to do what an expert does best: to exercise his or her expertise. This seems a more natural fit to an expert's capabilities than the requirements of traditional methodologies that experts explicitly enumerate the rules that they employ.

`http://archive.ics.uci.edu/ml/datasets/bands`

Breast Tissue Data Set (breasttissue)

This dataset contains transformed data from clinical images made from breast tissue in the form of an impedance spectrum combined with a classification by experts. Impedance measurements of freshly ex-

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

cised breast tissue were made at the following frequencies: 15.625, 31.25, 62.5, 125, 250, 500, 1000 KHz. These measurements plotted in the (real, -imaginary) plane constitute the impedance spectrum from where the breast tissue features are computed. The dataset can be used for predicting the classification of either the original six classes or of four classes by merging together the fibro-adenoma, mastopathy and glandular classes whose discrimination is not important (they cannot be accurately discriminated anyway).

`http://archive.ics.uci.edu/ml/datasets/breast+tissue`

Corral Data Set (corral)

The basic idea behind this dataset is an artificial domain where the target concept is $(A_0 \wedge A_1) \vee (B_0 \wedge B_1)$. Additionally, one irrelevant and one correlated feature are added. The correlated feature is a feature highly correlated with the label, but with a 25% error rate.

Note that in the experiments in this thesis the original training set of 32 instances and test set of 32 instances were combined into a single dataset of 64 instances.

This dataset is no longer available on the UCI Machine Learning repository.

`http://www.sgi.com/tech/mlc/db/corral.names`

Credit Approval Data Set (crx)

This file concerns credit card applications. All feature names and values have been changed to meaningless symbols to protect confidentiality of the data. This dataset is interesting because there is a good mix of features – continuous, nominal with small numbers of values, and nominal with larger numbers of values. There are also a few missing values.

`http://archive.ics.uci.edu/ml/datasets/credit+approval`

Dermatology Data Set (dermatology)

The differential diagnosis of erythemato-squamous diseases is a real problem in dermatology. They all share the clinical features of erythema and scaling, with very little differences. The diseases in this group are psoriasis, seboreic dermatitis, lichen planus, pityriasis rosea, cronic dermatitis, and pityriasis rubra pilaris. Usually a biopsy is necessary for the diagnosis but unfortunately these diseases share many histopathological features as well. Another difficulty for the differential diagnosis is that a disease may show the features of another disease at the beginning stage and may have the characteristic features at the following stages. Patients were first evaluated clinically with 12 features. Afterwards, skin samples were taken for the evaluation of 22 histopathological features. The values of the histopathological features are determined by an analysis of the samples under a microscope.

In the dataset constructed for this domain, the family history feature has the value 1 if any of these diseases has been observed in the family, and 0 otherwise. The age feature simply represents the age of the patient. Every other feature (clinical and histopathological) was given a degree in the range of 0 to

© University of Pretoria

3. Here, 0 indicates that the feature was not present, 3 indicates the largest amount possible, and 1, 2 indicate the relative intermediate values.

`http://archive.ics.uci.edu/ml/datasets/Dermatology`

Echocardiogram Data Set (echocardiogram)

All the patients suffered heart attacks at some point in the past. Some are still alive and some are not. The survival and still-alive variables, when taken together, indicate whether a patient survived for at least one year following the heart attack.

The problem addressed by past researchers was to predict from the other variables whether or not the patient will survive at least one year. The most difficult part of this problem is correctly predicting that the patient will NOT survive. (Part of the difficulty seems to be the size of the data set.)

`http://archive.ics.uci.edu/ml/datasets/echocardiogram`

Statlog (German Credit Data) Data Set (german)

This dataset concerns credit card applications in Germany. The original dataset, in the form provided by Prof. Hofmann, was used which contains categorical/symbolic features.

`http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29`

Glass Identification Data Set (glass)

The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence if it is correctly identified. The data set contains measurements of the refractive index and different mineral and metal contents.

Note that the original data file " glass.data" contains an ID number for each instance. Since the instances are also ordered by class, this ID number can be used to classify the dataset with 100% accuracy. In the experiments in this thesis the ID number was excluded from the dataset.

`http://archive.ics.uci.edu/ml/datasets/Glass+Identification`

Hepatitis Data Set (hepatitis)

The dataset contains information on chronic hepatitis patients and the classes indicate if the patient lives or dies. For each patient, the 19 features are potentially useful predictors like age, sex and standard chemical measurements.

`http://archive.ics.uci.edu/ml/datasets/Hepatitis`

© University of Pretoria

Hill-Valley Data Set (hill-valley)

This is an artificially generated dataset which contains examples of stylized hills and valley's. Each record represents 100 points on a two-dimensional graph. When plotted in order (from 1 through 100) as the Y co-ordinate, the points will create either a Hill (a bump in the terrain) or a Valley (a dip in the terrain).

The dataset consists of different parts:

- two datasets (without noise) are a training/testing set pair where the hills or valleys have a smooth transition.
- two datasets (with noise) are a training/testing set pair where the terrain is uneven, and the hill or valley is not as obvious when viewed closely.

Note that in the experiments in this thesis all four original training and tests sets were combined into a single dataset of 1212 instances.

`http://archive.ics.uci.edu/ml/datasets/hill-valley`

Horse Colic Data Set (horse-colic)

Note that features 25, 26, and 27 in the original dataset each are codes that combine four indicators with regards to lesions. These three features were each split into four separate features. Attribute 24, which indicates whether the problem (lesion) was surgical. was chosen as the class feature.

`http://archive.ics.uci.edu/ml/datasets/Horse+Colic`

Ionosphere (ionosphere)

This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power in the order of 6.4 kilowatts. See the work by Sigillito *et al.* [128] for more details. The targets were free electrons in the ionosphere. “Good” radar returns are those showing evidence of some type of structure in the ionosphere. “Bad” returns are those that do not; their signals pass through the ionosphere.

Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 features per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

`http://archive.ics.uci.edu/ml/datasets/ionosphere`

Iris Data Set (iris)

The data set contains three classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other two; the latter are not linearly separable from each other. The predicted feature is the class of iris plant.

`http://archive.ics.uci.edu/ml/datasets/Iris`

© University of Pretoria

Labor Relations Data Set (labor)

The data includes all collective agreements reached in the business and personal services sector for locals with at least 500 members (teachers, nurses, university staff, police, etc) in Canada in 1987 and the first quarter of 1988.

`http://archive.ics.uci.edu/ml/datasets/Labor+Relations`

BUPA Liver Disorders Data Set (liver)

The dataset consists of simplified medical records of individual male individuals who were treated for a liver disorder, classified into two different classes of disorders. Included are five variables from blood tests which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption. The other variable notes the number of half-pint equivalents of alcoholic beverages drunk per day by the individual.

It is known that in the original dataset four duplicates are present (row 84 and 86, row 141 and 318, row 143 and 150, row 170 and 176). These duplicated rows were not removed, but the full set of 345 instances was used.

`http://archive.ics.uci.edu/ml/datasets/Liver+Disorders`

Lung Cancer Data Set (lung-cancer)

The data describes three types of pathological lung cancers. No information is known on what the 56 individual variables represent nor on where the data was originally used. All variables are nominal, taking on integer values 0-3. In the original data four values for the fifth feature were -1. These values have been changed to unknown. In the original data one value for the 39th feature was 4. This value has been changed to unknown.

`http://archive.ics.uci.edu/ml/datasets/Lung+Cancer`

Lymphography Data Set (lymphography)

This is one of three domains provided by the Oncology Institute that has repeatedly appeared in the machine learning literature. The other two are the Breast Cancer Data Set and the Primary Tumor Data Set. The data consists of a classification of the diagnosis of a lymph in one of four classes, plus 18 features each described by nominal values in the original dataset.

`http://archive.ics.uci.edu/ml/datasets/lymphography`

MONK-1 Data Set (monk-1)

The MONK's problems were the basis of a first international comparison of learning algorithms. One significant characteristic of this comparison is that it was performed by a collection of researchers, each

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

of whom was an advocate of the technique they tested (often they were the creators of the various methods). In this sense, the results are less biased than in comparisons performed by a single person advocating a specific learning method, and more accurately reflect the generalization behavior of the learning techniques as applied by knowledgeable users.

There are three MONK's problems. Each problem relies on the same artificial robot domain, in which robots are described by six different features:

1. head shape $\in$ round, square, octagon
2. body shape $\in$ round, square, octagon
3. is smiling $\in$ yes, no
4. holding $\in$ sword, balloon, flag
5. jacket color $\in$ red, yellow, green, blue
6. has tie $\in$ yes, no

For MONK-1, the class is a binary concept described by

$$(\text{head shape} = \text{body shape}) \vee (\text{jacket color} = \text{red})$$

and there is no noise in the dataset. Hence, features 1, 2, and 5 should be selected.

`http://archive.ics.uci.edu/ml/datasets/monks-problems`

MONK-2 Data Set (monk-2)

The domain for the MONK-2 problem is exactly the same as that described above under the MONK-1 Data Set. For MONK-2, the class concept described by the data set is that exactly two of the six features have their first value. There is no noise in the dataset. All features should be selected to be able to correctly determine the class concept.

`http://archive.ics.uci.edu/ml/datasets/monks-problems`

Libras Movement Data Set (movement-libras)

The dataset contains 15 classes of 24 instances each, where each class references a hand movement type in LIBRAS. Videos of the movements have been converted into 90 numeric features.

In the video pre-processing, a time normalization is carried out selecting 45 frames from each video, according to an uniform distribution. In each frame, the centroid pixels of the segmented objects (the hand) are found, which compose the discrete version of the curve F with 45 points. All curves are normalized in the unitary space.

In order to prepare these movements to be analyzed by algorithms, a mapping operation was conducted, that is, each curve F was mapped in a representation with 90 features, representing the coordinates of movement.

`http://archive.ics.uci.edu/ml/datasets/Libras+Movement`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Musk (Version 1) Data Set (musk-1)

The dataset describes a set of 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 molecules are judged to be non-musks. The goal is to learn to predict whether new molecules will be musks or non-musks. However, the 166 features that describe these molecules depend upon the exact shape, or conformation, of the molecule. Because bonds can rotate, a single molecule can adopt many different shapes. To generate this data set, the low-energy conformations of the molecules were generated and then filtered to remove highly similar conformations. This left 476 conformations. Then, a feature vector was extracted that describes each conformation.

This many-to-one relationship between feature vectors and molecules is called the multiple instance problem. When learning a classifier for this data, the classifier should classify a molecule as “musk” if *any* of its conformations is classified as a musk. A molecule should be classified as “non-musk” if *none* of its conformations is classified as a musk.

As prescribed by the data set’s creators, the features “molecule_name” and “conformation_name” were excluded from the dataset used in the experiments.

`http://archive.ics.uci.edu/ml/datasets/Musk+%28Version+1%29`

Parity 5+5 Data Set (parity5-5)

The dataset consists of instances of 10 features that are bits, with the class being the parity of the bits 2, 3, 4, 6, and 8. The other five features are irrelevant for the classification.

Note that in the experiments in this thesis the full set of all 1024 possible combinations of bits was used.

This dataset is no longer available on the UCI machine learning repository.

`http://www.sgi.com/tech/mlc/db/parity5+5.all`

Parkinsons Data Set (parkinsons)

This dataset is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson’s disease (PD). Each column in the table is a particular voice measure, and each row corresponds to one of 195 voice recordings from these individuals (“name” column). The main aim of the data is to discriminate healthy people from those with PD, according to “status” column which is set to 0 for healthy and 1 for PD. There are around six recordings per patient, the name of the patient is identified in the first column.

Note that the names of the patients were discarded for the experiments in this thesis.

`http://archive.ics.uci.edu/ml/datasets/parkinsons`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Pima Indians Diabetes Data Set (pima)

The dataset comes from the National Institute of Diabetes and Digestive and Kidney Diseases. The aim is to forecast the onset of diabetes mellitus. The diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to World Health Organization criteria (i.e., if the two hour post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). The population lives near Phoenix, Arizona, USA. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

`http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes`

Molecular Biology (Promoter Gene Sequences) Data Set (promoter)

The dataset contains E. Coli gene sequences (DNA) which represent either promoter sequences or non-promoter sequences. The domain theory for recognizing promoters is that promoters have a region where a protein (RNA polymerase) must make contact and the helical DNA sequence must have a valid conformation so that the two pieces of the contact region spatially align.

The original dataset consists of instances with the features coded as text strings of length 57 listing the DNA nucleotides, each a letter from “agtc”. These were converted to 57 integers 1-4 based on the rank from the list “agtc”. The name feature was discarded for the experiments in this thesis.

`http://archive.ics.uci.edu/ml/datasets/promoter`

Connectionist Bench (Sonar, Mines vs. Rocks) Data Set (sonar)

The dataset consists of patterns obtained by bouncing sonar signals off rocks and a metal cylinder at various angles and under various conditions. The transmitted sonar signal is a frequency-modulated chirp, rising in frequency. The data set contains signals obtained from a variety of different aspect angles, spanning 90 degrees for the cylinder and 180 degrees for the rock.

Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time. The integration aperture for higher frequencies occur later in time, since these frequencies are transmitted later during the chirp.

The label associated with each record contains the letter “R” if the object is a rock and “M” if it is a mine (metal cylinder). The numbers in the labels are in increasing order of aspect angle, but they do not encode the angle directly.

`http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar`
`%2C+Mines+vs.+Rocks%29`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Soybean (Large) Data Set (soybean)

This dataset consists of instances of soybean diseases, classified by experts into 19 classes. There are 35 categorical features, some nominal and some ordered. The values for features are encoded numerically, with the first value encoded as 0, the second as 1, and so forth. This mean the data was already partially preprocessed as the textual values were converted into numbers.

Note that in the experiments in this thesis the original training set of 307 instances and test set of 376 instances were combined into a single dataset of 683 instances.

`http://archive.ics.uci.edu/ml/datasets/Soybean+%28Large%29`

SPECTF Heart Data Set (spectf)

The dataset describes diagnoses of cardiac Single Proton Emission Computed Tomography (SPECT) images. Each of the patients is classified based on the cardiologists' diagnoses into two categories: normal or abnormal. The database of 267 SPECT image sets (patients) was processed to extract features that summarize the original SPECT images. As a result, 44 continuous feature patterns were created for each patient.

Note that in the experiments in this thesis the original training set of 80 instances and test set of 187 instances were combined into a single dataset of 267 instances.

`http://archive.ics.uci.edu/ml/datasets/SPECTF+Heart`

Tic-Tac-Toe Endgame Data Set (tic-tac-toe)

This database encodes the complete set of possible board configurations at the end of tic-tac-toe games, where "x" is assumed to have played first. This means all 958 board configurations were either "x" or "o" has won, or the board has been completely filled resulting in a tie. The target concept encoded by the class feature is "win for x" (i.e., true when "x" has one of eight possible ways to create a "three-in-a-row").

`http://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame`

Statlog (Vehicle Silhouettes) Data Set (vehicle)

The purpose is to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette. The vehicle may be viewed from one of many different angles. Four "Corgie" model vehicles were used for the experiment: a double decker bus, Chevrolet van, Saab 9000 and an Opel Manta 400. This particular combination of vehicles was chosen with the expectation that the bus, van and either one of the cars would be readily distinguishable, but it would be more difficult to distinguish between the cars.

The features were extracted from the silhouettes by the HIPS (Hierarchical Image Processing System) extension BINATTS, which extracts a combination of scale independent features utilizing both classical

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

moments based measures such as scaled variance, skewness and kurtosis about the major/minor axes and heuristic measures such as hollows, circularity, rectangularity and compactness.

The images were acquired by a camera looking downwards at the model vehicle from a fixed angle of elevation (34.2 degrees to the horizontal). The vehicles were placed on a diffuse back-lit surface (light box). The vehicles were painted matte black to minimize highlights. The images were captured using a CRS4000 framestore connected to a vax 750. All images were captured with a spatial resolution of 128x128 pixels quantized to 64 greylevels. These images were thresholded to produce binary vehicle silhouettes, negated (to comply with the processing requirements of BINATTS) and thereafter subjected to shrink-expand-expand-shrink HIPS modules to remove "salt and pepper" image noise.

The vehicles were rotated and their angle of orientation was measured using a radial graticule beneath the vehicle. 0 and 180 degrees corresponded to "head on" and "rear" views respectively while 90 and 270 corresponded to profiles in opposite directions. Two sets of 60 images, each set covering a full 360 degree rotation, were captured for each vehicle. The vehicle was rotated by a fixed angle between images. These datasets are known as e2 and e3 respectively.

A further two sets of images, e4 and e5, were captured with the camera at elevations of 37.5 degrees and 30.8 degrees respectively. These sets also contain 60 images per vehicle apart from the vans in image set e4, which contains only 46 owing to the difficulty of containing the van in the image at some orientations.

The dataset on the UCI Machine Learning repository consists of nine separate files which form a roughly equal split of the instances. In the experiments in this thesis these files are combined into a single dataset.

`http://archive.ics.uci.edu/ml/datasets/Statlog+%28Vehicle+Silhouettes%29`

Congressional Voting Records Data Set (vote)

This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac (CQA), 98th Congress, 2nd session 1984. The CQA lists nine different types of votes: voted for, paired for, and announced for (these three simplified to yea), voted against, paired against, and announced against (these three simplified to nay), voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known (these three simplified to an unknown disposition). Hence, the dataset contains instances (one for each representative) which each consists of a class label identifying party affiliation and 16 features stating the simplified vote made on key issues.

Note that for this dataset it was ensured that the label "unknown" was converted to a numerical value of 0.5, with yea valued as 1 and nay as 0.

`http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records`

© University of Pretoria

Connectionist Bench (Vowel Recognition) Data Set (vowel)

The dataset consists of a numerical representation of vowel sounds from multiple speakers. The vowel data was recorded for examples of the eleven steady state vowels of English spoken by fifteen speakers for a speaker normalization study. The steady state vowel were spoken using the following 11 words: heed, hid, head, had,hard, hud, hod, hoard, hood, who'd, heard. The word was uttered once by each of the fifteen speakers.

The speech signals were low pass filtered at 4.7kHz and then digitized to 12 bits with a 10kHz sampling rate. Twelfth order linear predictive analysis was carried out on six 512 sample Hamming windowed segments from the steady part of the vowel. The reflection coefficients were used to calculate 10 log area parameters, yielding 10 numerical features.

Each speaker thus yielded six frames of speech from eleven vowels. This combines to 15 (speakers) times 6 (frames) times 11 (vowels) equals 990 instances. The classification problem is to determine the vowel spoken from the 10 features.

Note that in the experiments in this thesis the original training set from "speakers" 0-47 and test set of speakers 48-99 were combined into a single dataset.

`http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Vowel+Recognition+-+Deterding+Data%29`

Breast Cancer Wisconsin (Diagnostic) Data Set (wdbc)

This database consists or diagnoses of breast cancer from images. The data comes from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. Samples arrive periodically as Dr. Wolberg reports his clinical cases. The database therefore reflects this chronological grouping of the data.

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Thirty real-valued features are computed for each cell nucleus. The class is set by the diagnosis whether the cell is benign or malignant.

Data was taken from the file labeled "wdbc.data", containing 569 instances. The case number present in the dataset was not used in the experiments in this thesis.

`http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29`

Wine Data Set (wine)

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

`http://archive.ics.uci.edu/ml/datasets/wine`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Zoo Data Set (zoo)

This dataset consists of classifications of animals into seven different types using 16 features. Of these features, 15 are boolean and the other - denoting the number of legs - is integer valued. A breakdown of which animals are in which type is enumerated below.

1. aardvark, antelope, bear, boar, buffalo, calf, cavy, cheetah, deer, dolphin, elephant, fruit bat, giraffe, girl, goat, gorilla, hamster, hare, leopard, lion, lynx, mink, mole, mongoose, opossum, oryx, platypus, polecat, pony, porpoise, puma, pussycat, raccoon, reindeer, seal, sea lion, squirrel, vampire, vole, wallaby,wolf
2. chicken, crow, dove, duck, flamingo, gull, hawk, kiwi, lark, ostrich, parakeet, penguin, pheasant, rhea, skimmer, skua, sparrow, swan, vulture, wren
3. pit viper, sea snake, slow worm, tortoise, tuatara
4. bass, carp, catfish, chub, dogfish, haddock, herring, pike, piranha, seahorse, sole, stingray, tuna
5. frog, frog, newt, toad
6. flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp
7. clam, crab, crayfish, lobster, octopus, scorpion, sea wasp, slug, starfish, worm

The dataset contains one instance for each animal, except for "frog". The two instances for frog a different, as one has the feature for venomous set to true, the other has it set to false.

Note that the full dataset of 101 instances (with two instances for frog) and 16 features was used the experiments in this thesis, while the name feature was discarded.

`http://archive.ics.uci.edu/ml/datasets/zoo`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Appendix E

Statistical methodology

The algorithm-topology pairs were compared for significant differences in performance using the Iman-Davenport test (ID-test) [55], which is a refinement of the better known Friedman test [37]. The ID-test was used to analyze the performance, measured for the MKP as the average error[1] on each of the problems and for FSP as the average classification error on each of the datasets. The null hypothesis of the ID-test was that all algorithm-topology pairs had the same median performance. The significance level was chosen as 0.05.

In case the ID-test rejected the null-hypotheses and showed a significant difference in the performance of the algorithm-topology pairs, further post-hoc tests were performed in order to determine which of the algorithm-topology pairs outperformed the other pairs. The post-hoc test used was that proposed by Nemenyi [103], which considers the differences in the average rank of the performance over all problems.

For the Nemenyi test, the Z-score (the normalized distance in average rank of the average error) was used as input:

$$Z = \frac{|\overline{R}_1 - \overline{R}_2|}{\sqrt{\frac{k\,(k+1)}{6\,N}}} \tag{E.1}$$

where $\overline{R}_i$ is the average rank of the average error for algorithm-topology pair i, k is the total number of algorithm-topology pairs being compared, and N is the number of problems or datasets on which the pairs were compared. This standard normally distributed Z-score was then translated into a p-value.

Because the post-hoc tests involved multiple pair-wise comparisons, the significance level α needed to be adjusted in order to maintain equal family-wise error rates. For this purpose the Holm-Bonferroni method [53] was used: the largest difference in average rank found in the Nemenyi test was compared at significance level α, the second largest difference was compared at significance level $\alpha/2$, and the k-th largest difference was compared at significance level α/k.

[1]The error is defined as the deviation from the known optimum for the small MKPs, and as the deviation from the LP relaxation bound for the large MKPs.

© University of Pretoria