Programación declarativa (2005–06)

Tema 7: Aplicaciones de PD: problemas de grafos y el problema de las reinas

José A. Alonso Jiménez

Andrés Cordón Franco

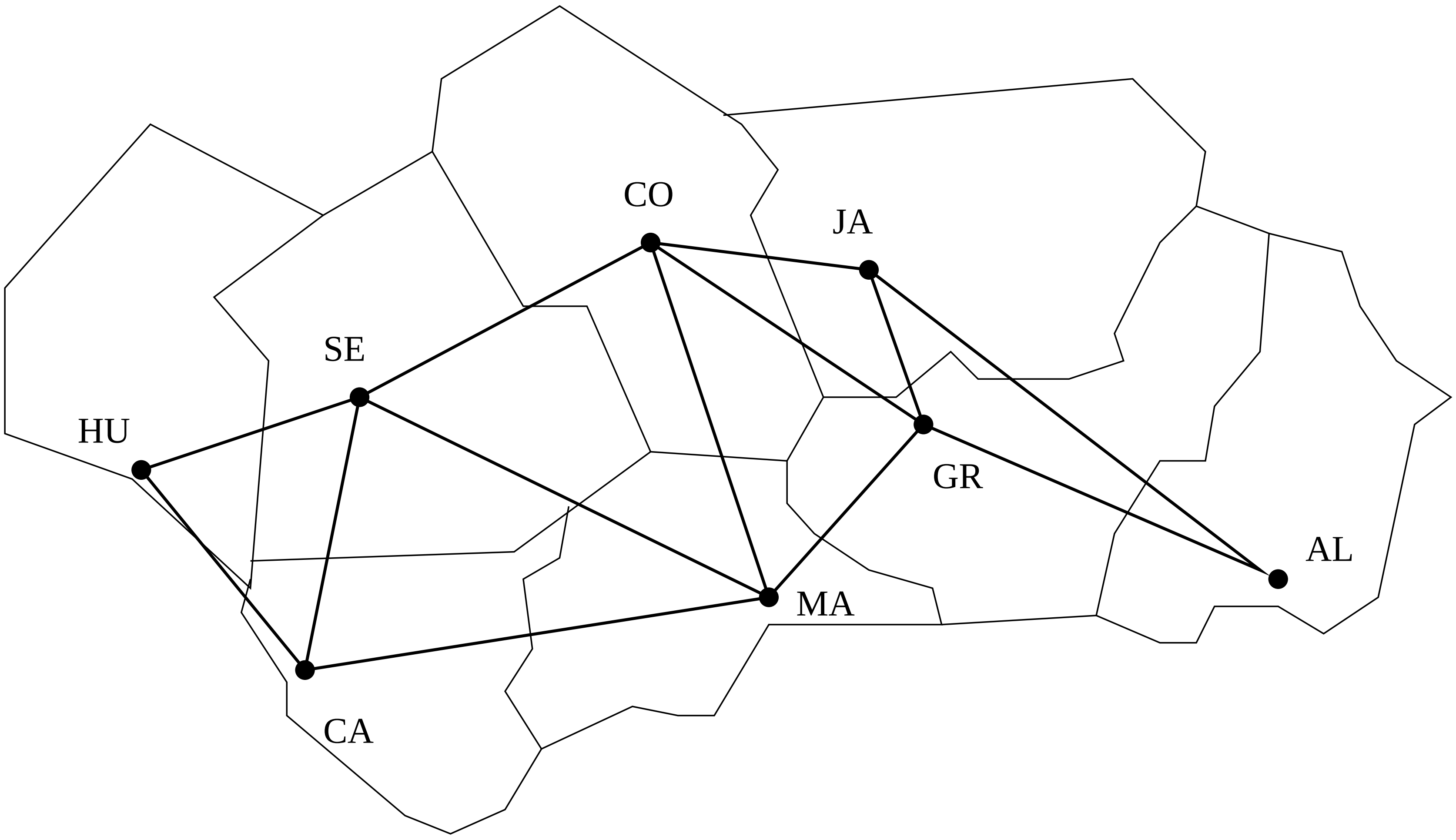
Grupo de Lógica Computacional

Dpto. Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Grafos

- Grafo de Andalucía:



Grafos

- Representación del grafo:
  - `arcos(+L)` se verifica si `L` es la lista de arcos del grafo.

```
arcos([huelva-sevilla,    huelva-cádiz,
       cádiz-sevilla,     sevilla-málaga,
       sevilla-córdoba,   córdoba-málaga,
       córdoba-granada,   córdoba-jaén,
       jaén-granada,      jaén-almería,
       granada-almería]).
```

  - `adyacente(?X,?Y)` se verifica si `X` e `Y` son adyacentes.

```
adyacente(X,Y) :-
   arcos(L),
   (member(X-Y,L) ; member(Y-X,L)).
```

  - `nodos(?L)` se verifica si `L` es la lista de nodos.

```
nodos(L) :-
   setof(X,Y^adyacente(X,Y),L).
```

Grafos: Caminos

- `camino(+A,+Z,-C)` se verifica si `C` es un camino en el grafo desde el nodo `A` al `Z`. Por ejemplo,

```
?- camino(sevilla,granada,C).
C = [sevilla, córdoba, granada] ;
C = [sevilla, málaga, córdoba, granada]
Yes
```

  Definición de `camino`

```
camino(A,Z,C) :-
    camino_aux(A,[Z],C).
```

Grafos: Caminos

- `camino_aux(+A,+CP,-C)` se verifica si `C` es una camino en el grafo compuesto de un camino desde `A` hasta el primer elemento del camino parcial `CP` (con nodos distintos a los de `CP`) junto `CP`.

```
camino_aux(A,[A|C1],[A|C1]).
camino_aux(A,[Y|C1],C) :-
   adyacente(X,Y),
   not(member(X,[Y|C1])),
   camino_aux(A,[X,Y|C1],C).
```

Grafos: Caminos hamiltonianos

- **hamiltoniano(-C)** se verifica si C es un camino hamiltoniano en el grafo (es decir, es un camino en el grafo que pasa por todos sus nodos una vez). Por ejemplo,

```
?- hamiltoniano(C).
C = [almería, jaén, granada, córdoba, málaga, sevilla,
?- findall(_C,hamiltoniano(_C),_L), length(_L,N).
N = 16
```

Definición de **hamiltoniano**

```
hamiltoniano_1(C) :-
    camino(_,_,C),
    nodos(L),
    length(L,N),
    length(C,N).
```

Grafos: Caminos hamiltonianos

- Definición de `hamiltoniano`

```
hamiltoniano_2(C) :-
    nodos(L),
    length(L,N),
    length(C,N),
    camino(_,_,C).
```

- Comparación de eficiencia

```
?- time(findall(_C,hamiltoniano_1(_C),_L)).
37,033 inferences in 0.03 seconds (1234433 Lips)
?- time(findall(_C,hamiltoniano_2(_C),_L)).
13,030 inferences in 0.01 seconds (1303000 Lips)
```

Grafos: Generacion de grafos completos

- `completo(+N,-G)` se verifica si `G` es el grafo completo de orden `N`. Por ejemplo,

```
completo(1,G) => G = []
completo(2,G) => G = [1-2]
completo(3,G) => G = [1-2,1-3,2-3]
completo(4,G) => G = [1-2,1-3,1-4,2-3,2-4,3-4]
```

Definición:

```
completo(N,G) :-
    findall(X-Y, arco_completo(N,X,Y),G).

arco_completo(N,X,Y) :-
    N1 is N-1,
    between(1,N1,X),
    X1 is X+1,
    between(X1,N,Y).
```

Grafos: Generación de grafos aleatorios

- `aleatorio(+P,+N,-G)` se verifica si `G` es un subgrafo de `{1..N}x{1..N}`, donde cada arco se ha elegido con la probabilidad `P` ($0 \leq P \leq 1$). Por ejemplo,

```
?- aleatorio(0.3,5,G).
G = [1-2, 3-4, 4-5]
?- aleatorio(0.3,5,G).
G = [1-2, 2-4, 3-4, 4-5]
```

  Definición:

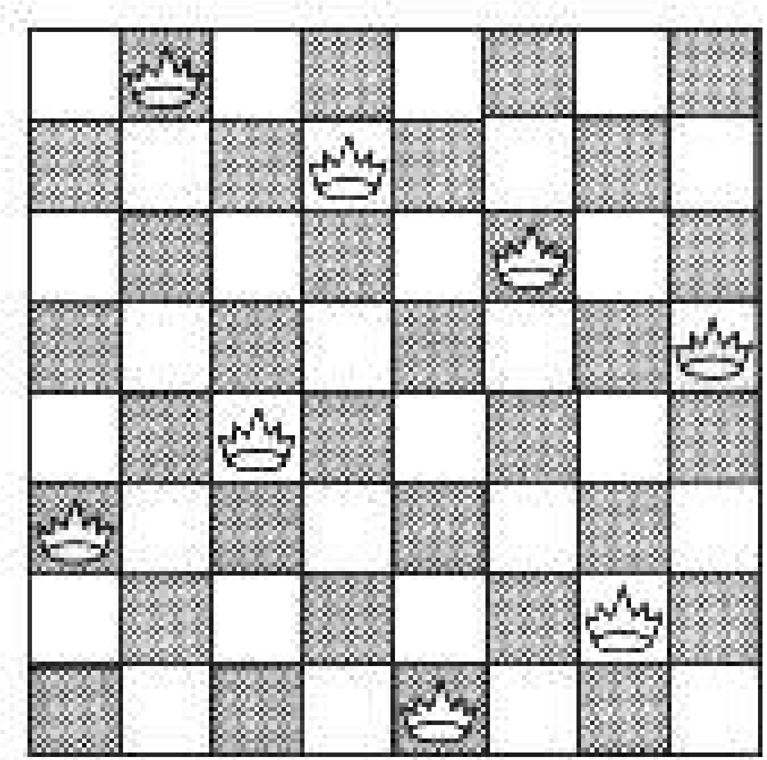
```
aleatorio(P,N,G) :-
    findall(X-Y,
            (arco_completo(N,X,Y),random(Z),Z=<P),
            G).
```

- `random(X)` se verifica si `X` es un número aleatorio en el intervalo `[0,1]`.

```
random(X) :- Y is random(1000), X is Y/1000.
```

El problema de las 8 reinas

- El problema de las ocho reinas consiste en colocar 8 reinas en un tablero rectangular de dimensiones 8 por 8 de forma que no se encuentren más de una en la misma línea: horizontal, vertical o diagonal.



El problema de las 8 reinas: Representación 1

- Sesión:

```
?- tablero(S), solución(S).
S = [1-4, 2-2, 3-7, 4-3, 5-6, 6-8, 7-5, 8-1] ;
S = [1-5, 2-2, 3-4, 4-7, 5-3, 6-8, 7-6, 8-1] ;
S = [1-3, 2-5, 3-2, 4-8, 5-6, 6-4, 7-7, 8-1]
Yes
```

- `tablero(L)` se verifica si `L` es una lista de posiciones que representan las coordenadas de 8 reinas en el tablero.

```
tablero(L) :-
    findall(X-_Y,between(1,8,X),L).
```

El problema de las 8 reinas: Representación 1

- `solución_1(?L)` se verifica si `L` es una lista de pares de números que representan las coordenadas de una solución del problema de las 8 reinas.

```
solución_1([]).
solución_1([X-Y|L]) :-
   solución_1(L),
   member(Y,[1,2,3,4,5,6,7,8]),
   no_ataca(X-Y,L).
```

- `no_ataca([X,Y],L)` se verifica si la reina en la posición `(X,Y)` no ataca a las reinas colocadas en las posiciones correspondientes a los elementos de la lista `L`.

```
no_ataca(_,[]).
no_ataca(X-Y,[X1-Y1|L]) :-
   X =\= X1,          Y =\= Y1,
   X-X1 =\= Y-Y1,     X-X1 =\= Y1-Y,
   no_ataca(X-Y,L).
```

El problema de las 8 reinas: Representación 2

- `solución_2(L)` se verifica si `L` es una lista de 8 números, $[n_1, \ldots, n_8]$, de forma que si las reinas se colocan en las casillas $(1, n_1)$,...,$(8, n_8)$, entonces no se atacan entre sí.

```
solución_2(L) :-
    permutación([1,2,3,4,5,6,7,8],L),
    segura(L).
```

- `segura(L)` se verifica si `L` es una lista de `m` números $[n_1, \ldots, n_m]$ tal que las reinas colocadas en las posiciones $(x, n_1), \ldots,$ $(x+m, n_m)$ no se atacan entre sí.

```
segura([]).
segura([X|L]) :-
    segura(L),
    no_ataca(X,L,1).
```

El problema de las 8 reinas

- `no_ataca(Y,L,D)` se verifica si `Y` es un número, `L` es una lista de números $[n_1, \ldots, n_m]$ y `D` es un número tales que las reinas colocada en la posición `(X,Y)` no ataca a las colocadas en las posiciones $(X + D, n_1), \ldots, (X + D + m, n_m)$.

```
no_ataca(_,[],_).
no_ataca(Y,[Y1|L],D) :-
   Y1-Y =\= D,
   Y-Y1 =\= D,
   D1 is D+1,
   no_ataca(Y,L,D1).
```

El problema de las 8 reinas: Representación 3

- `solución_3(?L)` se verifica si L es una lista de 8 números, $[n_1, \ldots, n_8]$, de forma que si las reinas se colocan en las casillas $(1, n_1), \ldots, (8, n_8)$, entonces no se atacan entre sí.

```
solución_3(L) :-
   solución_3_aux(
      L,
      [1,2,3,4,5,6,7,8],
      [1,2,3,4,5,6,7,8],
      [-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7],
      [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]).
```

El problema de las 8 reinas

- `solución_3_aux(?L,+Dx,+Dy,+Du,+Dv)` se verifica si `L` es una permutación de los elementos de `Dy` de forma que si `L` es $[y_1, \ldots, y_n]$ y `Dx` es $[1, \ldots, n]$, entonces $y_j - j$ $(1 \leq j \leq n)$ son elementos distintos de `Du` e $y_j + j$ $(1 \leq j \leq n)$ son elementos distintos de `Dv`.

```
solucion_aux([],[],_Dy,_Du,_Dv).
solucion_aux([Y|Ys],[X|Dx1],Dy,Du,Dv) :-
   select(Y,Dy,Dy1),
   U is X-Y,
   select(U,Du,Du1),
   V is X+Y,
   select(V,Dv,Dv1),
   solucion_aux(Ys,Dx1,Dy1,Du1,Dv1).
```

El problema de las 8 reinas

- Comparaciones

```
?- time((findall(_S,(tablero_1(_S), solucion_1(_S)),_L)
211,330 inferences in 0.12 seconds (1761083 Lips)
N = 92

?- time((findall(_S,solución_2(_S),_L),length(_L,N))).
1,422,301 inferences in 0.72 seconds (1975418 Lips)
N = 92

?- time((findall(_S,solucion_3(_S),_L),length(_L,N))).
120,542 inferences in 0.07 seconds (1722029 Lips)
N = 92
```

Búsqueda de todas las soluciones para N reinas:

	solución 1		solución 2		solución 3	
N	inferencias	seg.	inferencias	seg.	inferencias	seg.
4	401	0.00	543	0.00	546	0.00
6	8,342	0.00	20,844	0.01	6,660	0.01
8	195,628	0.15	1,422,318	0.91	120,614	0.09
10	5,303,845	4.05	150,300,540	96.82	2,774,095	2.01
12	182,574,715	147.22			83,067,721	64.93

Bibliografía

- I. Bratko *Prolog Programming for Artificial Intelligence (2nd ed.)* (Addison–Wesley, 1990)
  - Cap. 4: “Using Structures: Example Programs”
  - Cap. 9: “Operations on Data Structures”
- L. Sterling y E. Shapiro *The Art of Prolog (2nd editition)* (The MIT Press, 1994)
  - Cap. 2 “Database programming”