UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Miglioramenti

Vincoli ridondanti

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

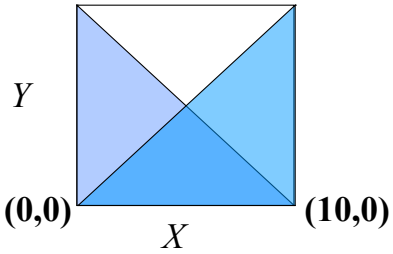
Prof. MARCO GAVANELLI

VINCOLI RIDONDANTI

- La propagazione in generale non è completa: alcuni valori che non portano a soluzioni possono rimanere nei domini
- Può accadere che aggiungendo vincoli che sono logicamente ridondanti si ottenga ulteriore propagazione
- Intuitivamente, un vincolo è logicamente ridondante se è implicato dagli altri vincoli del problema
- **Definizione**: un vincolo *c* è logicamente ridondante nel *CSP* = < *V, D, C* > se
  tutte le soluzioni del *CSP*
  sono anche soluzioni del *CSP'* = <*V, D, C* **U** {*c*} >
- I vincoli ridondanti possono essere utili per ridurre lo spazio di ricerca anche se introducono un overhead (trade-off).

Vincoli logicamente ridondanti

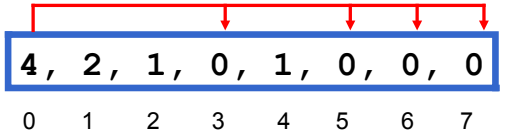
```
[X,Y]::0..10,     X#>=Y,
    Y #=< 10-X
```



- *Nessuna propagazione (entrambi i vincoli già AC).*
- *Se aggiungo il vincolo (logicamente ridondante)* `Y#=<5` *ho propagazione*

VINCOLI RIDONDANTI

- Esempio: Sequenza magica
- Dato un insieme di *n*+1 variabili $X_0,\ldots,X_n$. Ogni X_i deve rispettare i seguenti vincoli :
  - 0 compare X_0 volte in soluzione
  - 1 compare X_1 volte
  - …
  - *n* appare X_n volte.



- Vincolo `occurrences(V,L,N)` (`fd_global`) impone che `V` compaia `N` volte nella lista `L`
- `magic_sequence([`$X_0,\ldots,X_n$`]):-`

```
X0,…,Xn ::[0..n],
occurrences(0,[X0,…,Xn], X0),
occurrences(1,[X0,…,Xn], X1),
...,
occurrences(n,[X0,…,Xn], Xn),
```

VINCOLI RIDONDANTI

- Vincolo ridondante: si noti che la somma di tutte le variabili moltiplicate per il loro valore è uguale al numero di celle nella sequenza.

4	2	1	0	1	0	0	0
0	1	2	3	4	5	6	7

- Quindi, le variabili soddisfano il vincolo:

$$X_1 + 2*X_2 + \ldots + N*X_n = N + 1$$

- `magic_sequence([X0,…,Xn]):-`

```
X0,…,Xn ::[0..n],
occurrences(0,[X0,…,Xn], X0),
occurrences(1,[X0,…,Xn], X1),
...,
occurrences(n,[X0,…,Xn], Xn),
X1+ 2*X2 +…+ N*Xn = N + 1,
...
```

Con P4 2Ghz, N=23

•senza vincolo ridondante

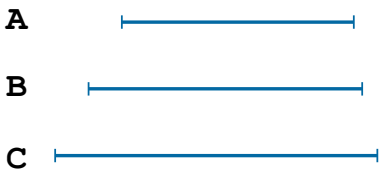
5.88s

•con vincolo ridondante

0.55s

Vincoli ridondanti

- Ovviamente, ci sono moltissimi casi in cui aggiungere un vincolo logicamente ridondante non aggiunge alcuna propagazione, perché anche la propagazione è ridondante
- Es: `A #=< B, B #=< C, A #=< C`
- `A #=< B`
  - Rimuove da dom_A i valori superiori a max_B
  - Rimuove da dom_B i valori inferiori a min_A (quindi $min_B \geq min_A$)
- `B #=< C`
  - Rimuove da dom_B i valori superiori a max_C
  - Rimuove da dom_C i valori inferiori a min_B
- `A #=< C`
  - Rimuove da dom_A i valori superiori a max_C
  - Rimuove da dom_C i valori inferiori a min_A



UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Strategie di ricerca

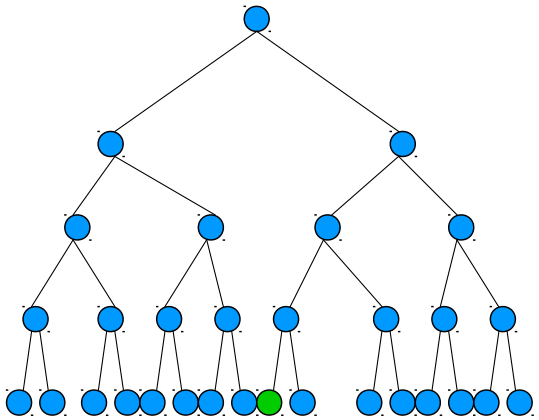
Limited Discrepancy Search
libreria `fd_search`

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Altri algoritmi di Search

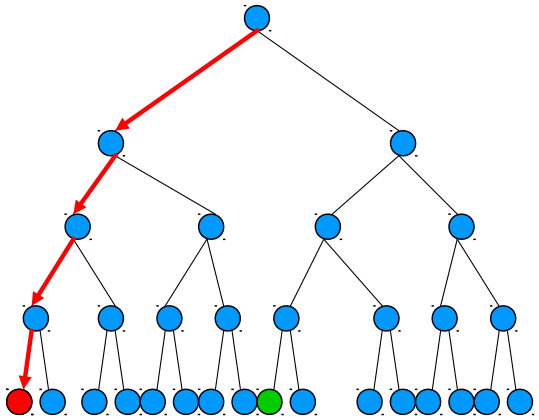
- ***Esplorazione depth-1st**: in profondità lungo il ramo di sinistra, poi torna indietro 1 passo, ecc.*
- *esplora le foglie di sinistra prima di quelle di destra.*
- *Supponiamo che l'euristica sia molto buona: in tutto il percorso compie solo 1 errore.*
  - *errore all'ultimo nodo => ☺*
  - *errore al primo => ☹*



Soluzione

Limited Discrepancy Search

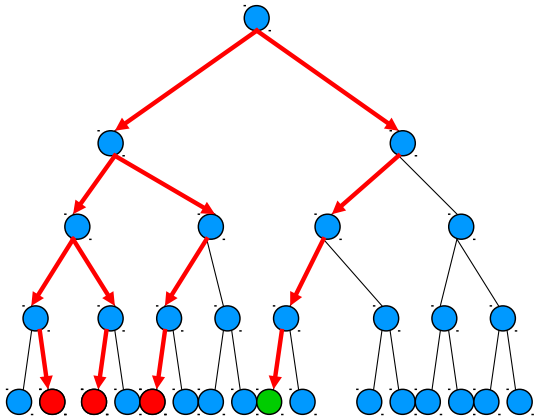
- *Discrepanza=0*
  seguo l'euristica



Soluzione

Limited Discrepancy Search

- *Discrepanza=1*: *Permetto che ci sia un errore nell'euristica.*
- *Prendo sempre il ramo di sinistra e prendo solo una volta il ramo di destra*



Soluzione

fd_search

- *Libreria* `fd_search`:

  `search`(L, Arg, Select, Choice, Method, Opt)
- *L: Lista di variabili.*
- *Arg = 0, Opt=*`[]` *(per i nostri usi, V manuale per approfondimenti)*
- *Select: euristica di selezione della variabile.*
  - `input_order, first_fail, smallest, largest, occurrence, most_constrained, max_regret, anti_first_fail`
- *Choice: euristica di selezione del valore*
  - `indomain, indomain_min, indomain_max, indomain_middle, indomain_median, indomain_split, indomain_random, indomain_interval`
- *Method: Algoritmo di search*
  - `complete, bbs(Steps), lds(Disc), credit(Credit, bbs(Steps) or lds(Disc)), dbs(Level, bbs(Steps) or lds(Disc)), sbds`
- *Opt: Lista di opzioni. Normalmente Opt=*`[]` *; se si desidera conoscere il numero di backtracks Opt=*`[backtrack(NumBacktr)]`

LDS in ECLiPSe

```
risolvi(L):-
   crea_domini(L),
   imponi_vincoli(L),
   search(L,0,most_constrained, indomain, lds(5),[]).
ottimizza(L):-
   crea_domini(L),
   imponi_vincoli(L),
   crea_obiettivo(L,F),
   minimize(search(L,0,most_constrained, indomain,
   lds(5),[]), F).
```

Domanda: è completo così? Come posso renderlo completo?

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Miglioramenti

Simmetrie

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

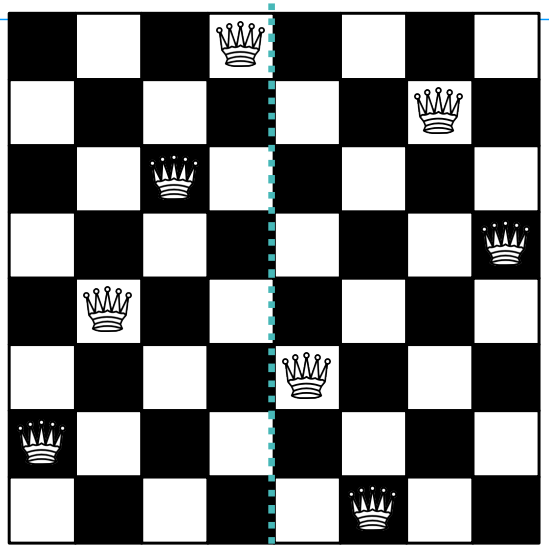
Simmetrie

- Es: Map coloring:
  - I colori sono intercambiabili.
  - Nel modello in cui gli stati sono le variabili e i valori sono i colori, abbiamo che i valori sono intercambiabili → value symmetry
- Data una soluzione (parziale), qualunque permutazione di colori è una soluzione (parziale)

n colori → $n!$ simmetrie

N-regine

- Modello con una variabile per colonna, le righe sono i valori
- Simmetria orizzontale: ci sono valori simmetrici
- Simmetria verticale: ci sono variabili simmetriche
- Rotazioni, simmetrie diagonali: simmetrie che coinvolgono variabili e valori



Definizione

- *Una simmetria è una permutazione di valori o variabili decisionali che conserva il concetto di soluzione:*
  - *Trasforma soluzioni (parziali) in soluzioni (parziali)*
  - *Trasforma non-soluzioni (parziali) in non-soluzioni (parziali)*
- *Le simmetrie sono studiate formalmente nella teoria dei gruppi*
  - *L'inverso di una simmetria è una simmetria*
  - *L'identità è una simmetria*
  - *La composizione di simmetrie è una simmetria*

Osservazione

Quando ci sono simmetrie:

- *ogni soluzione può essere trasformata in un'altra soluzione tramite una simmetria → tempo polinomiale per ottenere un'altra soluzione*
- *Ogni non-soluzione è associata ad un'altra non-soluzione per ogni simmetria*
  - *Se ho molte simmetrie, rischio di perdere molto tempo ad esplorare soluzioni simmetriche*

Modello CP: SIMMETRIE

- Problema: si perde tempo per esplorare stati equivalenti che non aggiungono alcuna informazione.
- Metodi per rimuovere le simmetrie:
  - cercare un modello che non contenga simmetrie
  - aggiungere vincoli al modello (esempio ordinamenti tra variabili)
  - aggiungere vincoli dinamicamente nel corso della ricerca
  - modificare la strategia di ricerca per rimuovere le simmetrie.
- Argomento su cui vi è una ricerca molto attiva

Symmetry Breaking

Aggiungere vincoli

ESEMPIO SEMPLICE

- Pigeonhole problem: Abbiamo *n*-1 gabbie in cui devono essere collocati *n* piccioni, uno per gabbia.
- Abbiamo *n* variabili P_1, … P_n che rappresentano i piccioni e un dominio contenente le gabbie da 1 a *n*-1
- Usiamo tutti vincoli binari di diverso (per vedere l'impatto delle simmetrie)
- Simmetrie di permutazione *n* variabili *n*! stati simmetrici
- Problema chiaramente insolubile ma abbiamo simmetrie di permutazione

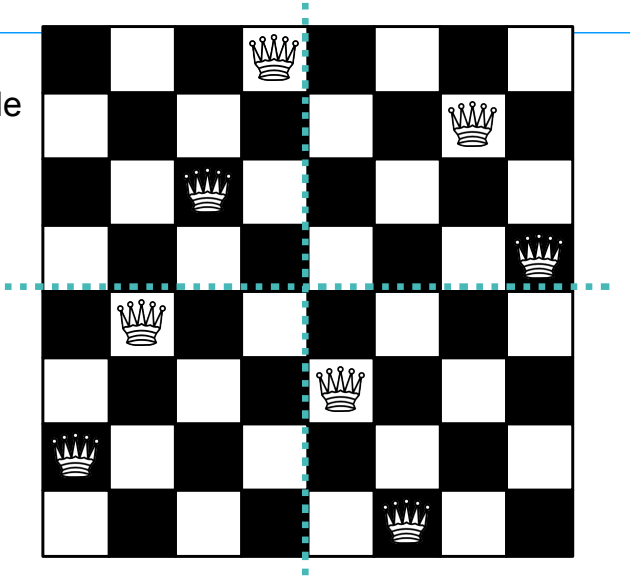
COME RIMUOVERE LE SIMMETRIE

- Si può imporre $P_1 < P_2$, $P_2 < P_3$ ….$P_{n-1} < P_n$ e si rimuovono tutte le simmetrie

	Con simmetrie		*NO simmetrie*	
N piccioni	*Size Tree*	*Tempo (s)*	*Size Tree*	*Tempo (s)*
6	*119*	*0.213*	*15*	*0.042*
7	*719*	*1.031*	*31*	*0.064*
8	*5039*	*7.619*	*63*	*0.102*
9	*40319*	*61.902*	*127*	*0.228*

N-regine

- Modello con una variabile per colonna, le righe sono i valori
- Simmetria orizzontale: ci sono valori simmetrici
- Simmetria verticale: ci sono variabili simmetriche
- Rotazioni, simmetrie diagonali: simmetrie che coinvolgono variabili e valori



SPORT SCHEDULING

- Dobbiamo allocare delle partite in diverse settimane
- Ci sono *n* squadre (nell'esempio da 0 a 7). Tutte le squadre devono giocare contro tutte le altre, quindi in totale ho *n**(*n*-1)/2 partite. Devo allocare una partita per ogni cella in modo che in ogni settimana una squadra giochi una sola volta.

	Week 1	*Week 2*	*Week 3*	*Week 4*	*Week 5*	*Week 6*	*Week 7*
Game 1	*0 vs 1*	*0 vs 2*	*4 vs 7*	*3 vs 6*	*3 vs 7*	*1 vs 5*	*2 vs 4*
Game 2	*2 vs 3*	*1 vs 7*	*0 vs 3*	*5 vs 7*	*1 vs 4*	*0 vs 6*	*5 vs 6*
Game 3	*4 vs 5*	*3 vs 5*	*1 vs 6*	*0 vs 4*	*2 vs 6*	*2 vs 7*	*0 vs 7*
Game 4	*6 vs 7*	*4 vs 6*	*2 vs 5*	*1 vs 2*	*0 vs 5*	*3 vs 4*	*1 vs 3*

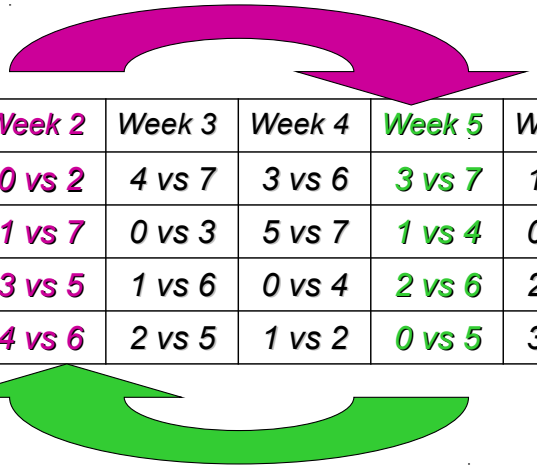
SPORT SCHEDULING: simmetrie

- *Le settimane sono simmetriche*
- *I Game sono simmetrici*

	Week 1	*Week 2*	*Week 3*	*Week 4*	*Week 5*	*Week 6*	*Week 7*
Game 1	*0 vs 1*	*0 vs 2*	*4 vs 7*	*3 vs 6*	*3 vs 7*	*1 vs 5*	*2 vs 4*
Game 2	*2 vs 3*	*1 vs 7*	*0 vs 3*	*5 vs 7*	*1 vs 4*	*0 vs 6*	*5 vs 6*
Game 3	*4 vs 5*	*3 vs 5*	*1 vs 6*	*0 vs 4*	*2 vs 6*	*2 vs 7*	*0 vs 7*
Game 4	*6 vs 7*	*4 vs 6*	*2 vs 5*	*1 vs 2*	*0 vs 5*	*3 vs 4*	*1 vs 3*

SPORT SCHEDULING: simmetrie

- *Cambiando l'ordine di due settimane trovo una soluzione equivalente*



	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Game 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Game 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Game 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Game 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

SPORT SCHEDULING: simmetrie

- *Cambiando l'ordine di due game trovo una soluzione equivalente*

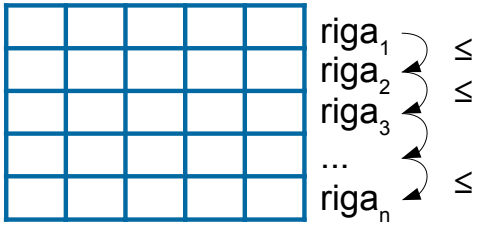
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Game 1	0 vs 1	3 vs 7	4 vs 7	3 vs 6	0 vs 2	1 vs 5	2 vs 4
Game 2	2 vs	1 vs 4	0 vs 3	5 vs 7	1 vs 7	0 vs 6	5 vs 6
Game 3	4 vs 5	2 vs 6	1 vs 6	0 vs 4	3 vs 5	2 vs 7	0 vs 7
Game 4	6 vs	0 vs 5	2 vs 5	1 vs 2	4 vs 6	3 vs 4	1 vs 3

PERCHE' PREOCCUPARSENE

- Per una matrice ***nxm*** che ha simmetrie di riga e di colonna ci sono ***n!*m!*** simmetrie (super-esponenziale)
- Per ogni soluzione (totale o parziale) ce ne sono ***n!*m!*** simmetriche, ma anche per ogni fallimento
- Eliminare tutte le simmetrie non è facile
- Ci si accontenta spesso di eliminarne alcune, in modo da ottenere un albero ridotto.

Simmetrie di riga

- *Supponiamo che in un problema basato su una matrice ci sia una simmetria di righe (scambiando due righe si ottiene una soluzione equivalente)*
- *Così come nel pigeonhole problem avevamo aggiunto un ordinamento, anche nelle matrici ha senso aggiungere un ordinamento fra le righe*



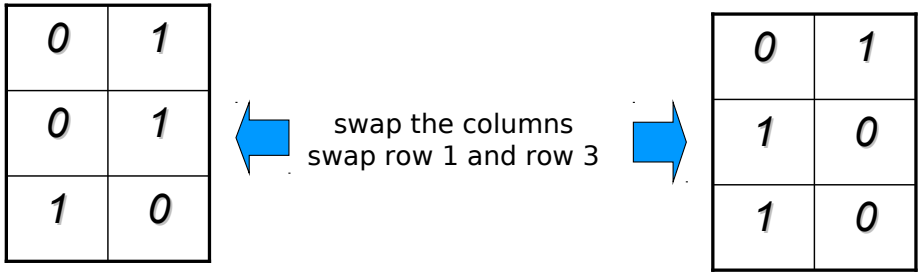
- *Però un ordinamento fra righe significa che dobbiamo imporre un ordinamento fra due array (o fra due liste)*

Lexico

- Un possibile ordinamento fra liste di valori è l'ordinamento lessicografico
- $La <_{lex} Lb$ se (detti $La = [A|Ta]$ e $Lb = [B|Tb]$)
  - $A < B$ oppure
  - $A = B$ e $Ta <_{lex} Tb$
- Analogamente per il $\leq$
- In ECLiPSe sono definiti due vincoli
  - **`lex_lt`** con ordinamento $<$
  - **`lex_le`** con ordinamento $\leq$
- Entrambi hanno come parametri due liste di variabili con dominio

Simmetrie di riga e colonna

- Qualora in un problema ci siano contemporaneamente simmetrie di riga e di colonna, si possono imporre ordinamenti lessicografici sia sulle righe sia sulle colonne
- Questo non elimina tutte le simmetrie



Strategie di ricerca

Strategie di ricerca per evitare simmetrie

Value symmetry

- In alcuni problemi, i valori sono intercambiabili
- Es: map coloring:
  - *Portogallo = verde, Spagna=blu, Francia=verde*
  - *Portogallo = blu, Spagna=rosso, Francia=blu*
- Data una soluzione (parziale), qualunque permutazione di colori è una soluzione (parziale)

$$n \text{ colori} \rightarrow n! \text{ simmetrie}$$

- L'importante non è il valore assegnato ad una variabile, ma qual è il gruppo di variabili che hanno quel valore

Strategia di ricerca per value symmetry

- Alla prima variabile si assegna il primo valore
- Alle variabili successive si possono assegnare
  - I valori usati nelle precedenti variabili
  - Più un nuovo valore
- Con questa strategia si eliminano tutte le *n*! simmetrie
- Overhead costante in tempo e spazio

Possibili colori:{r,g,b,y}

{r} {r,g} {r,g,b}{r,g,b}

r	g	r		
It	Fr	Es	Ch	De

Quadrato magico

- Un quadrato magico di ordine *N* è una matrice *N*x*N* che contiene i numeri interi da 1 a N^2, tale che
  - la somma dei numeri su ogni riga
  - la somma dei numeri su ogni colonna
  - la somma dei numeri sulle due diagonali sia sempre costante

2	7	6	15
9	5	1	15
4	3	8	15
15	15	15	15 15

- Si scriva un predicato che prende in ingresso un numero *N* e fornisce un quadrato magico di ordine *N*.
- *Predicati utili*:
  - `matrix(N,M,Righe,Colonne)` genera una matrice *N*x*M* di variabili, data sia per Righe che per Colonne. Usare lib(matrix_util)
    ES: matrix(2,3,R,Col) fornisce
    R= [[A,B,C], [D,E,F]]
    Col=[[A,D], [B,E], [C,F]]
  - `flatten(ListaDiListe,Lista)` appiattisce una lista di liste in una lista semplice
    Es: flatten([A,[1,X],[[3]]],L) fornisce L=[A,1,X,3].

Esercizio: Codici a correzione di errore

- Si vuole generare una matrice binaria (cioè i cui elementi sono 0 o 1) *N*×*M* tale da massimizzare la minima distanza di Hamming fra ogni coppia di righe.
- La distanza di Hamming fra due sequenze di bit è il numero di bit diversi nella stessa posizione. Ad esempio, le due sequenze

```
1 0 0 1 0
1 1 0 0 0
```

  sono a distanza 2 (hanno 2 bit diversi).
- Ad esempio, dati *N*=4 e *M*=3 una soluzione ottima è data dalla matrice:

```
0 0 0
0 1 1
1 0 0
1 1 1
```

- Si scriva un programma ECLiPSe che genera tale matrice.
- Si cerchi poi di migliorare l'efficienza del programma considerando le simmetrie