>>> N-Queens problem and its solutions

Progetto di Ricerca Operativa†

A.A 2021/2022

Name: Francesco Mazza M63001338

Date: Settembre 2022

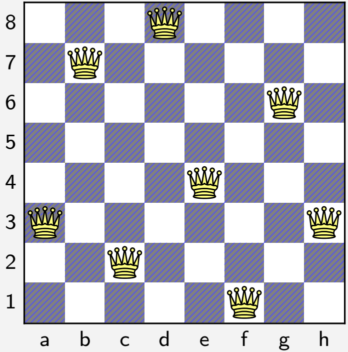
†Prof. Maurizio Boccia

>>> Indice

>>> Formulazione del problema

Il problema delle n regine, ispirato al gioco degli scacchi, fu studiato dal matematico Carl Gauss e formulato nel 1850 come segue:

* Si vuole trovare la collocazione, in una scacchiera di dimensione NxN, di N regine in modo tale che non possano attaccarsi le une con le altre.



Il numero massimo di regine presenti, dovrà ammontare al numero di righe/colonne della scacchiera considerata: N.

Per $N \in \{2, 3\}$ il problema non ammette alcuna soluzione.

>>> Modellazione

Una possibile modellazione del problema introduce come variabili decisionali $x_{i,j}$.

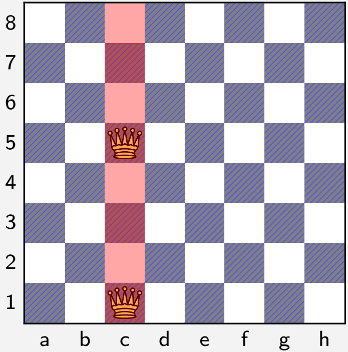
$$\text{Dove } x_{i,j} = \begin{cases} 1, & \texttt{se la posizione (i,j) è occupata} \\ 0, & \texttt{altrimenti} \end{cases}$$

Pertanto la funzione obiettivo risulta:

$$\max \sum_{i=1}^{N} \sum_{j=1}^{N} x_{i,j}$$

\>>> Modellazione 1/5

A partire dalle regole del gioco degli scacchi, i vincoli devono essere i seguenti:



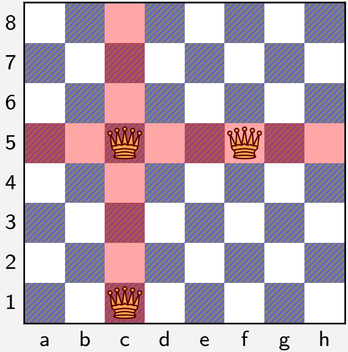
* Due regine sulla stessa colonna si attaccano: per ogni colonna dovrà esserci una sola regina.

$$\sum_{i=1}^{N} x_{i,j} = 1$$

$$\forall j \in \{1, ..., N\}$$

>>> Modellazione 2/5

A partire dalle regole del gioco degli scacchi, i vincoli devono essere i seguenti:

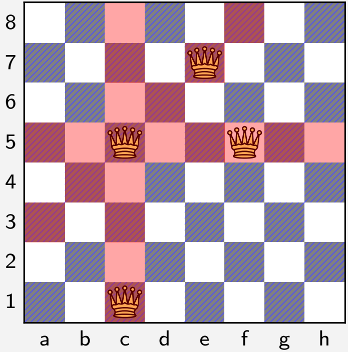


* Due regine sulla stessa riga si attaccano: per ogni riga dovrà esserci una sola regina.

$$\sum_{j=1}^{N} x_{i,j} = 1$$

$$\forall i \in \{1, ..., N\}$$

A partire dalle regole del gioco degli scacchi, i vincoli devono essere i seguenti:



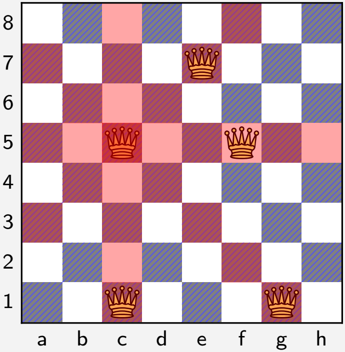
* Per ogni diagonale (2N-1 diagonali) dovrà esserci al massimo una sola regina.

$$\sum_{i,j \in \{1 \ldots N\} : i-j=k} x_{i,j} \leq 1$$

$$\forall k \in \{1-N, \ldots, N-1\}$$

NB: Elementi sulla stessa diagonale hanno valore di i-j = k (costante e pari al numero della diagonale!)

A partire dalle regole del gioco degli scacchi, i vincoli devono essere i seguenti:



* Per ogni anti-diagonale (2N-1 diagonali) dovrà esserci al massimo una sola regina.

$$\sum_{i,j \in \{1 \dots N\} : i+j=k} x_{i,j} \leq 1$$

$$\forall k \in \{2, \dots, 2N\}$$

NB: Elementi sulla stessa anti-diagonale hanno valore di i+j = k (costante e pari al numero della anti-diagonale!)

>>> Modellazione 5/5

Il modello (pli) complessivo risulta:

$$\max \sum_{i=1}^{N} \sum_{j=1}^{N} x_{i,j}$$

$$\sum_{j=1}^{N} x_{i,j} = 1 \qquad \forall i \in \{1, ..., N\}$$

$$\sum_{i=1}^{N} x_{i,j} = 1 \qquad \forall j \in \{1, ..., N\}$$

$$\sum_{i,j \in \{1...N\} : i-j=k} x_{i,j} \leq 1 \qquad \forall k \in \{1-N, ..., N-1\}$$

$$\sum_{i,j \in \{1...N\} : i+j=k} x_{i,j} \leq 1 \qquad \forall k \in \{2, ..., 2N\}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall i, j \in 1, ..., N$$

>>> Soluzione Gurobi

Si definisce la dimensione del problema da risolvere, si dichiara il modello e la funzione obiettivo, dopo aver introdotto le variabili binarie.

```
# Dimensioni scacchiera:
n = 8

# Creazione modello:
m = gp.Model("NQUEENS")

# Introduciamo le variabili decisionali:
x = m.addMVar((n, n), vtype=GRB.BINARY, name="x")

# Funzione obiettivo: massimizzare il numero delle regine presenti (n):
m.setObjective(x.sum(), GRB.MAXIMIZE)
```

>>> Soluzione Gurobi

Introduzione dei vincoli per righe e colonne:

```
# Per ogni riga/colonna valgono i seguenti constraints:
for i in range(n):

    # Al massimo una regina per riga:
    m.addConstr(x[i, :].sum() == 1, name="row"+str(i))

    # Al massimo una regina per colonna:
    m.addConstr(x[:, i].sum() == 1, name="col"+str(i))
```

Per i vincoli sulle diagonali bisogna osservare che:

* Sono in numero $2N - 1$.
* Si può ciclare sul numero di diagonali elencando i valori di i e j appartenenti a ciascuna. Si impone allora che la somma sugli gli indici che soddisfano la condizione sia ≤ 1 .

>>> Soluzione Gurobi

Introduzione dei vincoli sulle diagonali:

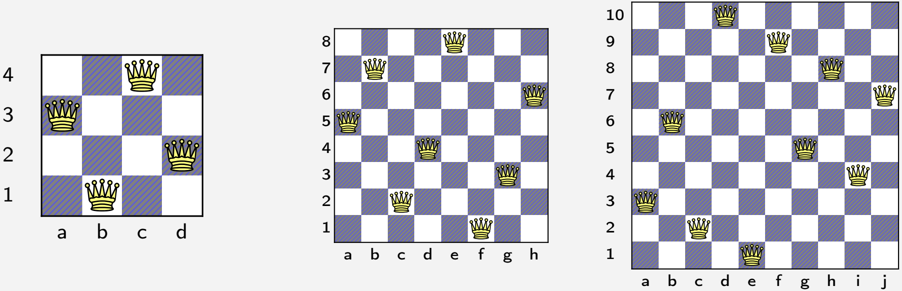
```
# Elenco tutti gli indici da considerare per ogni diag nella somma (vettori I e J)

for k in range(1-n,n-1):
    I = []
    J = []
    for i in range(n):
        for j in range(n):
            if (i-j)==k:
                I.append(i)
                J.append(j)
    m.addConstr(x[I,J].sum() ≤ 1, name="diag"+str(k))
```

Analogo ragionamento per le anti-diagonali.

>>> Soluzione Gurobi

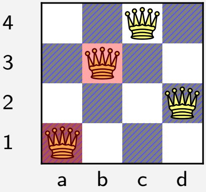
Si riescono ad ottenere soluzioni in tempi ragionevoli per ordini di grandezza dell'ordine delle centinaia di righe/colonne.

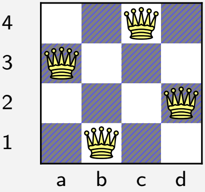


Per la risoluzione di problemi di ottimizzazione combinatoria, spesso si ricorre a metodi euristici i quali permettono di ottenere soluzioni anche per istanze del problema con N elavati.

>>> Ricerca locale

* In euristiche di tipo migliorativo, può ragionare in termini di numero di conflitti sulle regine posizionate. $min\{conflicts\}$.
* Mossa: A partire dalla soluzione corrente S_i, scambia la colonna per due regine poste su righe diverse. Facendolo per ogni colonna si indaga l'introno di S_i





Dove la formattazione è vettoriale, e ogni v[i] rappresenta la pos. di colonna, sulla riga i-esima.

Esempio: $[1, 4, 2, 3] \longrightarrow [2, 4, 1, 3]$

Cioé esprimiamo la soluzione come una permutazione di elementi posti sulle varie colonne: $\Pi = \{\pi(1), \pi(2), ...\pi(N)\}$.

>>> Ricerca locale

1. Genera randomicamente una soluzione con N regine: S_0.
2. Data la soluzione corrente S_i:
   per una riga random, scambia il valore di colonna con quello di ogni altra riga.
3. Seleziona la soluzione migliore S^*.
   3.1 Se $conflicts(S^*) \leq conflicts(S_i) \longrightarrow S_{i+1} = S*$, poi torna allo step 2.
   3.2 Se $conflicts(S^*) = 0$ OR $i \geq MaxIterations \rightarrow$ Stop.

(Esempio S: [4,1,3,2] 1, swap riga 4)

S^1: Swap 4 1 $[2,1,3,4]$ conflitti: 2

S^2: Swap 4 2 $[4,2,3,1]$ conflitti: 2

S^3: Swap 4 3 $[4,1,2,3]$ conflitti: 4

S^4: Swap 4 4 $[4,1,3,2]$ conflitti: 1 $\rightarrow$ nuova S corrente.

Si può evitare di indagare soluzioni già visitate in passato, o ad esempio quelle che hanno fornito un numero di conflitti superiore a quello corrente:

Si può costruire una matrice di dimensioni NxN. Ogni elemento $a_{i,j}$ rappresenta la possibilità di poter effettuare lo scambio tra le regine poste nelle righe i e j.
Se $a_{i,j} > 0$ allora lo swap non viene eseguito.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \multicolumn{5}{c}{\dotfill} \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

>>> Simulated Annealing

Un altro approccio di tipo migliorativo prevede l'utilizzo del Simulated Annealing. Con questo tipo di approccio si prevede:

* La generazione di una soluzione di partenza generata randomicamente (numero qualsiasi di conflitti).
* La generazione di una nuova soluzione, scambiando randomicamente, la colonna in cui si trovano due regine.
* L'accettazione della nuova soluzione, obbedisce ad una legge probabilistica:

  Dove $P(S^*) = \begin{cases} 1, & \text{se } \Delta C \leq 0 \\ e^{\frac{-\Delta C}{\alpha T}}, & \text{se } \Delta C > 0 \end{cases}$ e

  $\Delta C = conflicts(S^*) - conflicts(S)$, αT fattore che modella l'abbassamento della temperatura all'aumentare del numero di iterazioni.

>>> Simulated Annealing

1. Genera randomicamente una soluzione con N regine: S_0.
2. Data la soluzione corrente S_i:
   Inverti la posizione (colonna) di due regine generando la soluzione candidata S^*.
3. Seleziona S^*.
   3.1 Se $DeltaC = conflicts(S^*) - conflicts(S) \leq 0$ allora $S_{i+1} = S^*$.
   3.2 Se $DeltaC = conflicts(S^*) - conflicts(S) > 0$ allora S^* viene accettata con probabilitá $P(S^*)$.
4. Se $conflicts(S_{i+1}) = 0$ OR $i \geq MaxIterations$. Stop,
   else Torna al passo 2.

>>> Simulated annealing: esempio

Esempio S, conf, $\alpha * T$: ($N = 4$ per avere meno iterazioni)

$[1,2,3,4] \to 6, 2$

$[4,2,3,1] \to 2, 1.9$

$[3,2,4,1] \to 1, 1.805$

$[3,2,1,4] \to 4, 1.714749 \to$ accetto.

$[3,4,1,2] \to 4, 1.62901249$

$[2,4,1,3] \to 0 \to$ opt.

Esempio S, conf, $\alpha * T$: ($N = 4$ per avere meno iterazioni)

$[1,2,3,4] \rightarrow 6, 2$

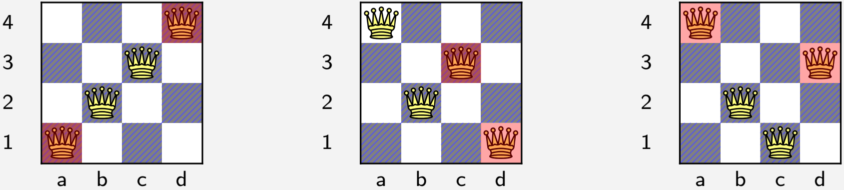
$[4,2,3,1] \rightarrow 2, 1.9$

$[3,2,4,1] \rightarrow 1, 1.805$

$[3,2,1,4] \rightarrow 4, 1.714749 \rightarrow$ accetto.

$[3,4,1,2] \rightarrow 4, 1.62901249$

$[2,4,1,3] \rightarrow 0 \rightarrow$ opt.



>>> Simulated annealing: esempio

Esempio S, conf, $\alpha * T$: ($N = 4$ per avere meno iterazioni)

$[1,2,3,4] \rightarrow 6, 2$

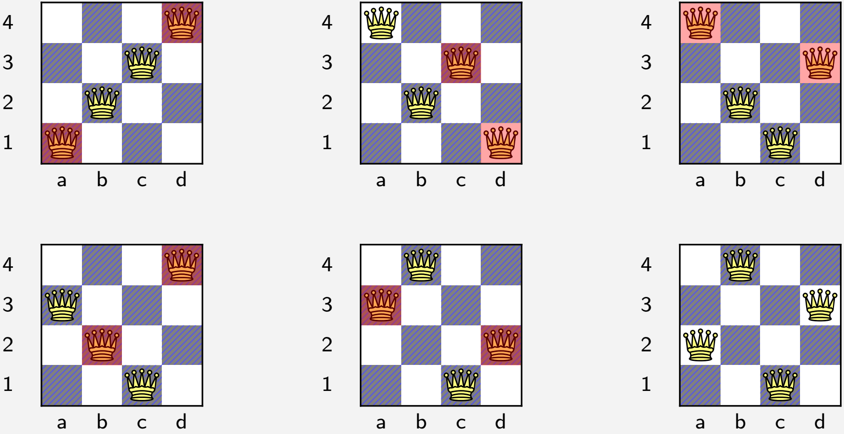
$[4,2,3,1] \rightarrow 2, 1.9$

$[3,2,4,1] \rightarrow 1, 1.805$

$[3,2,1,4] \rightarrow 4, 1.714749 \rightarrow$ accetto.

$[3,4,1,2] \rightarrow 4, 1.62901249$

$[2,4,1,3] \rightarrow 0 \rightarrow$ opt.



>>> Risultati

Gli algoritmi descritti giungono ad una soluzione ammissibile (verificato per $N \leq 1024$):

- Si ottiene una delle soluzioni ammissibili oppure una inammissibile (allo scadere del numero di iterazioni max).
- Alcuni esempi di soluzioni ottenute si trovano del file risultati.txt

[1] Comparison of Heuristic Algorithms for the N-Queen Problem: Ivica Martinjak, M. Golub.

[2] The n-Queens Problem: Craig Letavec, John Ruggiero.