Intelligenza Artificiale

Anno Accademico 2022 - 2023

Esercizi in Python:

Tabu Search

(Problema delle n Regine)

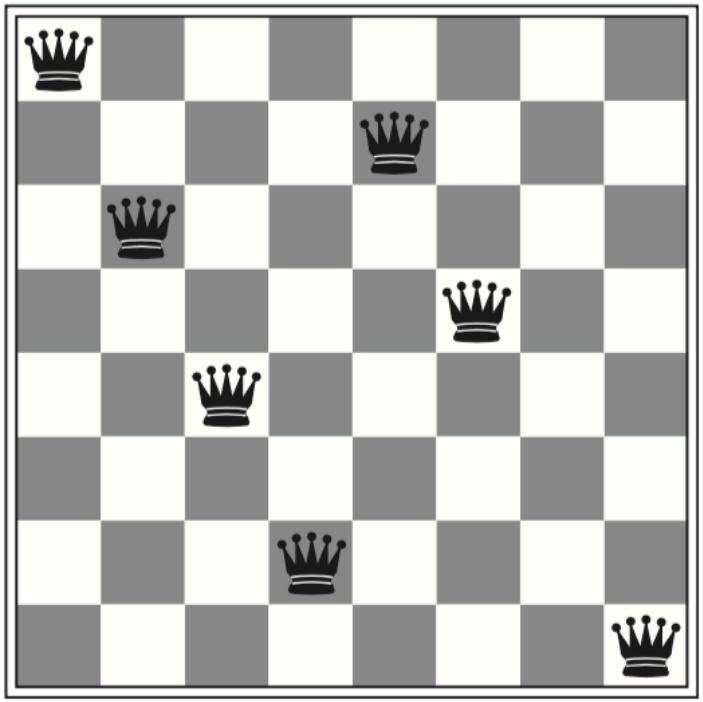
Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

Tabu Search
Problema delle n regine



Tabu Search
Problema delle n regine

Decidiamo di limitare lo spazio degli stati come segue:

- Consideriamo solo stati, ossia disposizioni delle regine nella scacchiera, nei quali ci sia solo una regina in ogni colonna e in ogni riga.
- Consideriamo come possibili mosse per passare da uno stato ad uno stato successore lo scambio di due colonne qualsiasi della scacchiera.
- In tal modo dobbiamo considerare solo gli attacchi sulle diagonali.

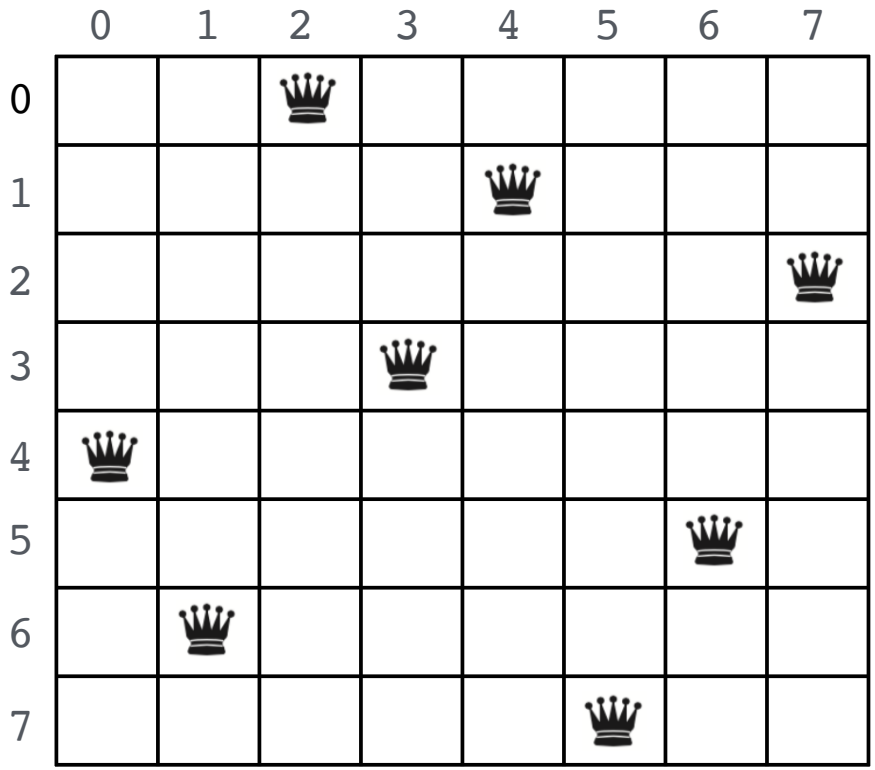
Tabu Search
Problema delle n regine

Pertanto il problema è così definito:

- Stati: una qualsiasi disposizione di **n** regine, in modo tale che ci sia una sola regina per colonna e una sola regina per riga.
- Goal State: una qualsiasi disposizione delle **n** regine sulla scacchiera che non si attaccano a vicenda.
- Funzione successore: un qualsiasi scambio di due colonne della scacchiera.
- Funzione di valutazione: numero di attacchi.
- Test obiettivo: numero di attacchi uguale a zero.

Problema delle N regine

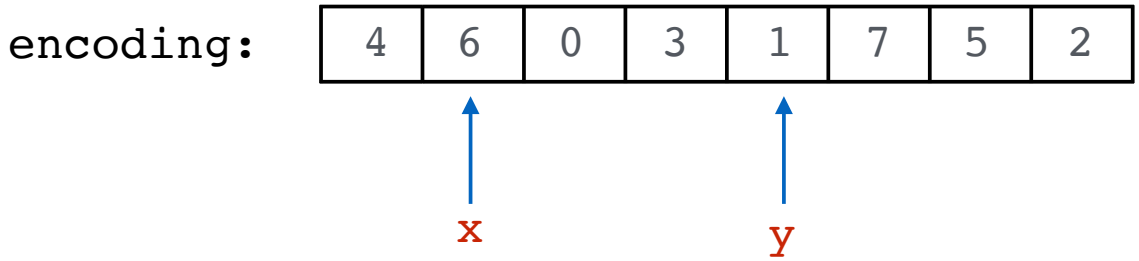
Rappresentazione scacchiera



encoding:	4	6	0	3	1	7	5	2

Descrizione Notebook
Funzione Tweak

- La funzione **`tweak`** sceglie casualmente due indici x e y nell'encoding, scambiando poi i valori degli elementi corrispondenti:



- In tal modo si scambiano le due colonne x e y della scacchiera.

Descrizione Notebook
Inizializzazione Stato

```
def inizializza(sol):

    # shake shake shake
    for c in range(0,DIMENSIONE-1):
        sol = tweak(sol)
    return sol


def tweak(sol):

    sol_copy = np.copy(sol)

    # scegli random due colonne distinte
    x = random.randint(0,DIMENSIONE-1)
    y = random.randint(0,DIMENSIONE-1)
    while x==y:
        y = random.randint(0,DIMENSIONE-1)

    # scambia le due colonne
    temp = sol_copy[y]
    sol_copy[y] = sol_copy[x]
    sol_copy[x] = temp

    return sol_copy
```

Descrizione Notebook
Valutazione Stato (1)

- Tale funzione calcola gli attacchi presenti in un certo stato:

```
def eval_stato(stato):

    # definizione della scacchiera N x N
    board  = [[0] * DIMENSIONE for i in range(DIMENSIONE)]

    # inserimento delle regine ('Q') nelle loro posizioni sulla scacchiera
    for i in range(0,DIMENSIONE):
        board[stato[i]][i]='Q'

    # spostamenti possibili sulla scacchiera
    dx = [-1,1,-1,1]
    dy = [-1,1,1,-1]

    # inizializzazione numero di attacchi (diretti o indiretti)
    conflitti = 0
```

Descrizione Notebook
Valutazione Stato (2)

```
for i in range(0,DIMENSIONE):
    x=stato[i]
    y=i

    # verifica attacchi sulle diagonali
    for j in range(0,4):
        tempx = x
        tempy = y
        while (True):
            tempx = tempx + dx[j]           # spostamento sull'asse x
            tempy = tempy + dy[j]           # spostamento sull'asse y
            if ((tempx < 0) or
                (tempx >= DIMENSIONE) or
                (tempy < 0) or
                (tempy >= DIMENSIONE)):
                break                       # si esce dal ciclo while se lo spostamento va fuori
            if (board[tempx][tempy]=='Q'):
                conflitti = conflitti + 1   # aggiornamento numero di attacchi
return conflitti
```

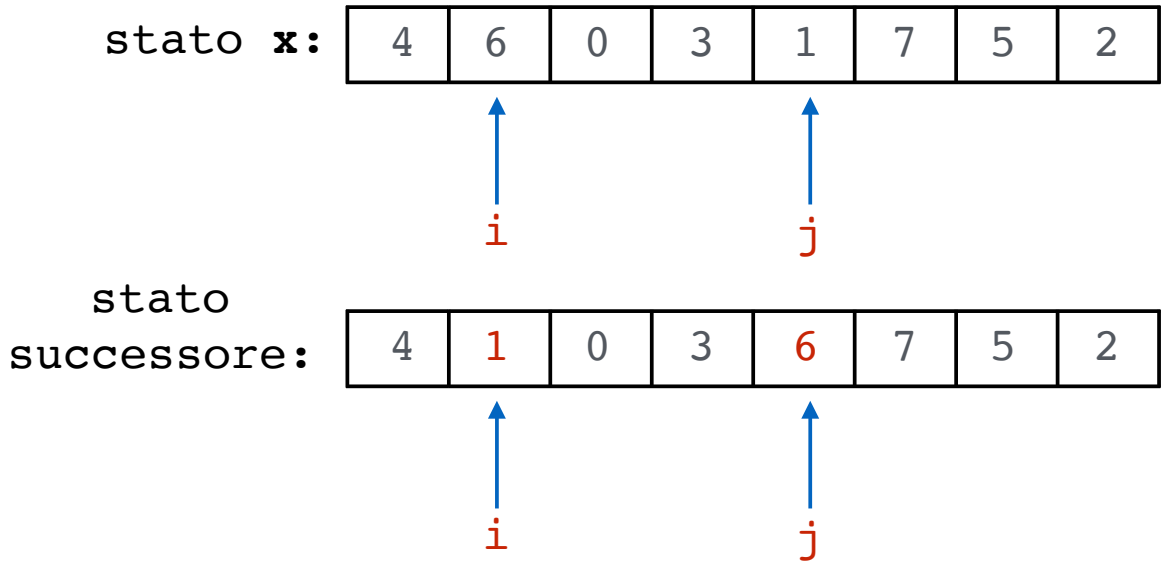
Descrizione Notebook
Generazione del Neighborhood

- La funzione genera tutti i successori di uno stato:

```
def generazione_successori(stato):
    """
    genera la lista ordinata di successori di uno stato
    """

    lista = []
    t = len(stato)

    for i in range(0, t-1):
        for j in range(i+1, t):
            buffer = copy.deepcopy(stato)
            temp = buffer[i]
            buffer[i] = buffer[j]
            buffer[j] = temp
            eval_successore = eval_stato(buffer)
            lista.append((buffer, eval_successore, (stato[i], stato[j])))

    lista.sort(key=lambda x: x[1])  # ordiniamo i successori in base alla loro valutazione
    return(lista)
```

Descrizione Notebook
Generazione Successore

- La funzione precedente, per ogni coppia di indici i e j dello stato, scambia i valori degli elementi corrispondenti. Ad esempio:



Descrizione Notebook
Test Tabu

- Tale funzione verifica se una mossa è **assente** nella **Tabu List**:

```
def tabu_test(sequenza, tabu_list):    # è True se una mossa NON è presente

    a, b = sequenza[2]
    if ((a, b) in tabu_list or (b, a) in tabu_list):
        assente = False
    else:
        assente = True
    return(assente)
```

Descrizione Notebook
Stampa Scacchiera

```
def stampa(sol):

    board = [[0] * DIMENSIONE for i in range(DIMENSIONE)]

    for x in range(0,DIMENSIONE):
        board[sol[x]][x]='Q'
    print("SCACCHIERA")
    for x in range(0,DIMENSIONE):
        for y in range(0,DIMENSIONE):
            if(board[x][y]=='Q'):
                print("Q   ",end=''),
            else:
                print(".   ",end=''),
        print("\n")
    print("\n\n")
```

Descrizione Notebook
Algoritmo (1)

```
def tabu_search(tabu_tenure):

    # impostazione stato iniziale
    stato = list(x for x in range(DIMENSIONE))
    current = inizializza(stato)
    eval_current = eval_stato(current)

    # inizializzazione best
    best = current
    eval_best = eval_current

    tabu_list = {}
    neighbours = []

    cont = 0
```

Descrizione Notebook
Algoritmo (2)

```
# while not criterio_terminazione():
while (cont < 100 and eval_best != 0):

    # generazione successori (stato, eval_stato, mossa) e ordinamento su eval_stato
    lista_successori = generazione_successori(current)
    if cont == 0:
        l = len(lista_successori)
        print('Numero successori: ', l, '\n')

    # selezione successori non tabu
    neighbours = list(filter(lambda n: tabu_test(n, tabu_list), lista_successori))

    next_state = neighbours[0][0]          # selezione del migliore dei successori
    eval_next_state = neighbours[0][1]
    print("Iterazione ", cont, ':')
    print('next_state: ', eval_next_state)
    delta = eval_best - eval_next_state
    if delta > 0:
        best = next_state                  # aggiornamento di best
        eval_best = eval_next_state
    current = next_state
    eval_current = eval_next_state
```

Descrizione Notebook
Algoritmo (3)

```
    # decremento del tabu_tenure
    for mossa in tabu_list:
        tabu_list[mossa] = tabu_list[mossa] - 1

    # eliminazione elementi con tenure uguale a zero
    tabu_list = {k: tabu_list[k] for k in tabu_list if tabu_list[k]!=0}

    # inserimento della mossa di next nella tabu_list
    mossa_next = neighbours[0][2]
    tabu_list[mossa_next] = tabu_tenure

    print("best_eval =", eval_best)
    print('mossa:', mossa_next)
    print('tabu_list:', tabu_list, '\n')

    cont += 1

return(best, eval_best)
```

Descrizione Notebook
Esempio di esecuzione algoritmo

In [25]:
```
DIMENSIONE = 30   # dimensione dei lati della scacchiera N x N (dove N è la DIMENSIONE)
```

In [26]:
```
soluzione, conflitti = tabu_search(5)
```

```
Iterazione  24 :
next_state:  4
best_eval = 4
mossa: (22, 13)
tabu_list: {(6, 13): 1, (13, 1): 2, (1, 6): 3, (22, 17): 4, (22, 13): 5}

Iterazione  25 :
next_state:  2
best_eval = 2
mossa: (0, 21)
tabu_list: {(13, 1): 1, (1, 6): 2, (22, 17): 3, (22, 13): 4, (0, 21): 5}

Iterazione  26 :
next_state:  0
best_eval = 0
mossa: (12, 28)
tabu_list: {(1, 6): 1, (22, 17): 2, (22, 13): 3, (0, 21): 4, (12, 28): 5}
```

Descrizione Notebook
Esempio di esecuzione algoritmo

```
In [27]: soluzione
```

```
Out[27]: array([ 6, 17, 13, 21, 18, 24,  5,  2, 28, 16,  4, 12,  8, 25, 23, 20, 14,
                27, 10,  0, 22, 29, 11,  9,  3,  1, 26,  7, 15, 19])
```

```
In [28]: conflitti
```

```
Out[28]: 0
```

Riferimenti

Glover, F. “Tabu Search - Part I”, *ORSA Journal on Computing*, Vol. 1, No. 3, 1989, pp. 190-206.

Glover, F. “Tabu Search - Part II”, *ORSA Journal on Computing*, Vol. 2, No. 1, 1990, pp. 4-32.

Michalewicz, Z. E Fogel, D.B. *How to Solve It: Modern Heuristics*, Springer, 2010.

Luke, S. *Essentials of Metaheuristics*, Second Edition, 2013.

Lubanovic, B. *Introducing Python*, O’Reilly, 2020.