Università di Perugia
Dipartimento di Matematica e Informatica

Esame di Artificial Intelligent Systems

Giro del cavallo ♘

Professore
Prof. Stefano Marcugini

Studente
Nicolò Posta

Anno Accademico 2022-2023

Indice

1 Obiettivo

L'obiettivo di questo progetto è fornire una soluzione accademica al problema del giro del cavallo implementando l'algoritmo DFS base e con l'euristica di Warnsdorff nel linguaggio funzionale OCaml.

2 Natura del problema e vincoli

Il problema del giro del cavallo può essere di due differenti nature:

- **Rilassato:** in questo caso è sufficiente trovare una successione di mosse che permette al cavallo di visitare tutte le case della scacchiera una singola volta senza ripetizioni.
- **Non rilassato:** in questo caso, invece, bisognerà trovare una successione di mosse che permette al cavallo di visitare tutte le case della scacchiera una sola volta ma ora con il vincolo che questo sia un percorso chiuso, ovvero che l'ultima mossa porti ad una mossa di distanza dalla casa di inizio permettendo di ricominciare il percorso da capo.

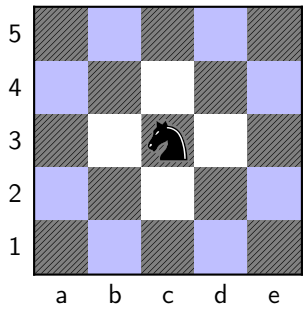
Come si può intuire la risoluzione del problema rilassato è molto più semplice che quella per risolvere il problema che richiede un percorso chiuso in quanto ci sono più soluzioni al primo che al secondo.

n	Chiusi	Aperti	Totali
8	26,534,728,821,064	19,565,293,442,158,840	19,591,828,170,979,904

Tabella 1: Numero di percorsi in una matrice 8×8

I vincoli del problema sono i seguenti:

- Il cavallo ha l'iconico movimento ad "L" degli scacchi.



- Il cavallo non può passare due volte sulla stessa casa della scacchiera.

- Il cavallo può partire da qualsiasi casa della scacchiera.
- Il cavallo deve tornare alla casa di partenza nel caso del problema non rilassato, invece non è richiesto nel caso del problema rilassato.
- Il cavallo non può uscire dalla scacchiera

Vincoli sulla grandezza della scacchiera: La scacchiera ha dei vincoli diversi a seconda se il problema da risolvere è quello rilassato o meno.

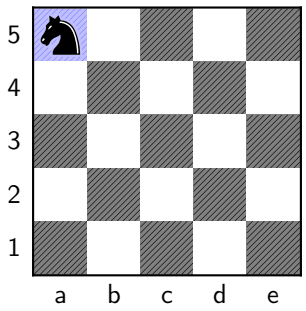
Per quello **rilassato** è sufficiente che per scacchiere $n \times n$ si abbia $n \geq 5$ oppure $n = 1$.

Invece per il problema **non rilassato** è necessario un ulteriore vincolo, ovvero che n sia **pari**.

Questo può essere *intuitivamente* dedotto dal fatto che ogni mossa del cavallo va a cambiare il colore della casa nella quale va a posizionarsi e perciò se si ha una scacchiera con un numero dispari di case e si visiteranno tutte quante solo una volta si avrà sempre che la casa finale sarà dello stesso colore della casa di partenza, richiedendo obbligatoriamente una mossa extra per poterla raggiungere, rendendo così il problema irrisolvibile.

3 Strutture dati

Per risolvere il problema ho trattato ogni casa della scacchiera come una tupla indicante la posizione della casa stessa, ad esempio la casa nell'angolo in alto a sinistra è stata rappresentata come la tupla **(0,0)** il che mi ha permesso di non dover utilizzare una matrice per definire la scacchiera.



Per eseguire le mosse ho definito una lista di tuple da sommare a quella rappresentante la casa della scacchiera effettivamente andando a ritornare il valore della casa che si viene a raggiungere eseguendo una specifica mossa.
In fine ho dovuto controllare se la mossa appena effettuata fosse valida, ovvero se il risultato della somma risultasse in una casa effettivamente presente nella scacchiera, in quanto somme con valori risultanti di segno negativo sono ovviamente mosse che porterebbero il cavallo a saltare fuori dalla scacchiera.
Di seguito ho riportato le funzioni implementate in **ocaml**:

```
(*Lista delle mosse che il cavallo può compiere *)
let mosse_cavallo = [(-1,-2); (-2,-1); (1,-2); (-2,1);
                     (-1,2); (2,-1); (1,2); (2,1)]

(*Questa funzione calcola la casa nella quele arriva il
cavallo data una certa casa iniziale ed una mossa da eseguire *)
let esegui_mossa (a,b) (c,d) = (a+c,b+d)

(*Lunghezza massima del cammino in una matrice n x n,
ovvero la grandezza della scacchiera *)
let max_cammino = (n * n)

(*Controlla se la mossa eseguita dal cavallo è una
mossa valida all'interno della scacchiera,
ovvero controlla se il cavallo non esce fuori dalla scacchiera *)
let mossa_in_scacchiera (a,b) =
    (a >= 0) && (a < n) &&
    (b >= 0) && (b < n)

(*Ritorna la lista delle mosse valide partendo
da una casa (x,y) posizione_cavallo *)
let mosse_valide posizione_cavallo =
    List.filter (mossa_in_scacchiera)
            (List.map (esegui_mossa posizione_cavallo) mosse_cavallo)
```

4 Algoritmi per la risoluzione del problema non rilassato

4.1 DFS senza euristica

L'utilizzo dell'algoritmo DFS per la scansione dello spazio degli stati alla ricerca di una soluzione è utilizzabile solo con scacchiere di piccolissime dimensioni (soltanto quelle di grandezza 5×5 e 6×6) in quanto la ricerca di un percorso chiuso (ovvero un cammino *Hamiltoniano*) è un problema **NP-complesso** e perciò la grandezza dello spazio delle soluzioni tende ad ingrandirsi in maniera esponenziale impedendo così la risoluzione del problema in un tempo polinomiale rendendo la risoluzione di scacchiere più grandi praticamente *impossibile*.

```
let dfs inizio successivi =
    let estendi cammino =
            List.map (function (x,y) -> (x,y)::cammino)
                (List.filter (fun (x,y) -> not (List.mem (x,y) cammino))
                             (successivi (List.hd cammino)))
    in let rec ricerca_ausiliaria = function
            [] -> raise NonTrovato
            | cammino::resto -> if List.mem (List.hd cammino) (successivi inizio)
            && max_cammino = List.length cammino
                then stampa_soluzione(cammino)
                else ricerca_ausiliaria ((estendi cammino) @ resto)
    in ricerca_ausiliaria [[inizio]]
```

Dopo aver trovato la soluzione al problema ho aggiunto la stampa a schermo di una matrice per controllare l'effettiva soluzione dal problema.

4.2 DFS con l'euristica di Warnsdorff

Come detto in precedenza l'algoritmo DFS da solo non è in grado di risolvere il problema del salto del cavallo, ma questo non perchè il problema è intrinsecamente troppo complesso da risolvere, ma soltanto perchè lo spazio degli stati è troppo grande da cercare nella sua interezza come fa la DFS.
Per ovviare al problema della DFS base possiamo introdurre un ordinamento delle mosse da prendere prima in considerazione così da evitare di rendere delle case della scacchiera irraggiungibili anche solo dopo poche mosse.
Questo è necessario perchè la DFS non inizia a fare *backtrack* fino a che non si è raggiunti una casa nella quale il cavallo non ha più mosse disponibili, ma questa non è detto che sia una soluzione al problema in quanto è possibile raggiungere degli stati nei quali non si hanno più mosse disponibili ma ancora ci sono delle case che non sono mai state visitate dal cavallo. L'euristica di **Warnsdorff** tenta di ovviare a questo problema ordinando le successive mosse del cavallo, cosa che non fa la DFS normale, in base al minor numero delle **successive mosse** che il cavallo sarà in grado di fare dopo di essa. Ovvero fa prediligere

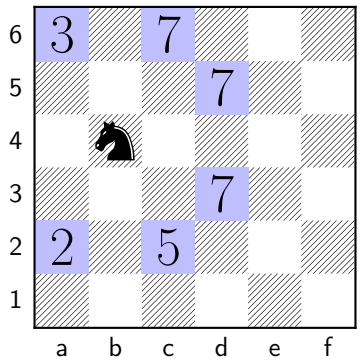
mosse *ai lati* della scacchiera invece che *verso l'interno* rendendo il cammino una specie di "*spirale*" attorno ai bordi della scacchiera. Questa soluzione funziona perchè dato che le case ai lati della scacchiera non permettono tutte le mosse possibili del cavallo ma solo quelle che non gli permettono di uscire dalla scacchiera hanno ovviamente anche meno case dalle quali possono essere raggiunte. Se queste case non sono tra le prime ad essere visitate è molto probabile che tutte le case che portano ad esse saranno già state visitate in precedenza rendendole così *irraggiungibili* e ovviamente impediranno di trovare una soluzione valida al problema del giro del cavallo. Queste case così dette *irraggiungibili* possono verificarsi dopo pochi passi dell'algoritmo ma non essere rilevabili subito in quanto il cavallo trova sempre altre case dove saltare fino a che non si blocca. Questo va ad obbligare la DFS a fare **backtrack** fino a che non è in grado di evitare che questa casa diventi *irraggiungibile* portando via moltissimo tempo e rendendo la ricerca in profondità di una soluzione molto ardua. L'utilizzo dell'euristica di **Warnsdorff** permette di visitare per prime le case con meno mosse possibili partendo da esse così che si elimini questo problema della formazione di case *irraggiungibili*.

```
(*DFS con euristica di Warnsdorff*)
let dfs_Warnsdorff inizio successivi =
    let estendi cammino =
            List.map (function (x,y) -> (x,y)::cammino)
                (List.filter (fun (x,y) -> not (List.mem (x,y) cammino))
                             (successivi (List.hd cammino)))
    in let rec ricerca_ausiliaria = function
            [] -> raise NonTrovato
            | cammino::resto -> if List.mem (List.hd cammino) (successivi inizio)
            && max_cammino = List.length cammino
                then stampa_soluzione cammino
                else ricerca_ausiliaria ((List.sort confronto (estendi cammino))
                @ resto)
    in ricerca_ausiliaria [[inizio]]
```

Di seguito ho riportato il codice dell'implementazione dell'ordinamento delle mosse nell'euristica di Warnsdorff.

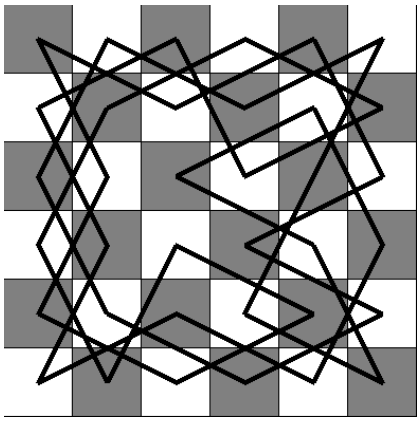
```
(*Ordinamento delle mosse secondo l'euristica di Warnsdorff*)
let confronto elem1 elem2 =
    let successivi_1 = List.filter (fun (x,y) -> not(List.mem (x,y) elem1))
                                                 (mosse_valide (List.hd elem1)) in

    let successivi_2 = List.filter (fun (x,y) -> not(List.mem (x,y) elem2))
                                                 (mosse_valide (List.hd elem2)) in

    let conto_1 = List.length(successivi_1) in

    let conto_2 = List.length(successivi_2) in

    (*Serve come funzione per l'ordinamento di List.sort*)
    if conto_1 > conto_2 then 1
    else if conto_1 = conto_2 then 0
    else -1
```

Di seguito una rappresentazione grafica dell'euristica di Warnsdorff. Ogni casa contiene un numero intero che indica il numero di mosse che il cavallo potrebbe fare da essa. In questo caso, la regola ci dice di spostarci sul quadrato con il numero intero più piccolo, ovvero la casa **a2**.

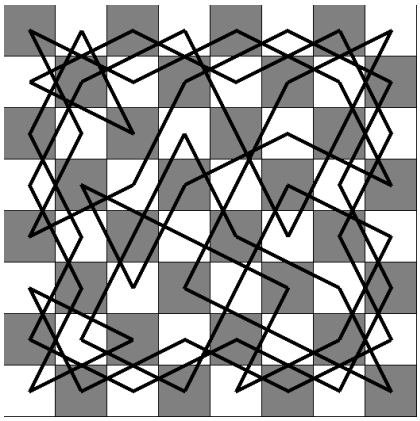


5 Soluzioni trovate

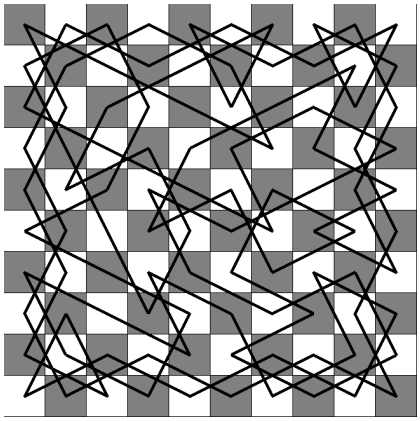
Di seguito ho stampato le soluzioni trovate su diverse scacchiere utilizzando l'euristica di Warnsdorff.



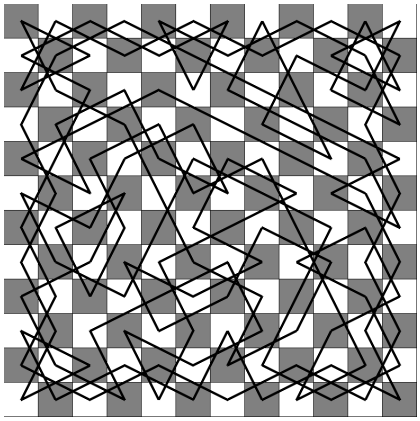
(a) Scacchiera 6 x 6



(b) Scacchiera 8 x 8



(c) Scacchiera 10 x 10



(d) Scacchiera 12 x 12