Sliding Block Puzzle - Sviluppo in PROLOG

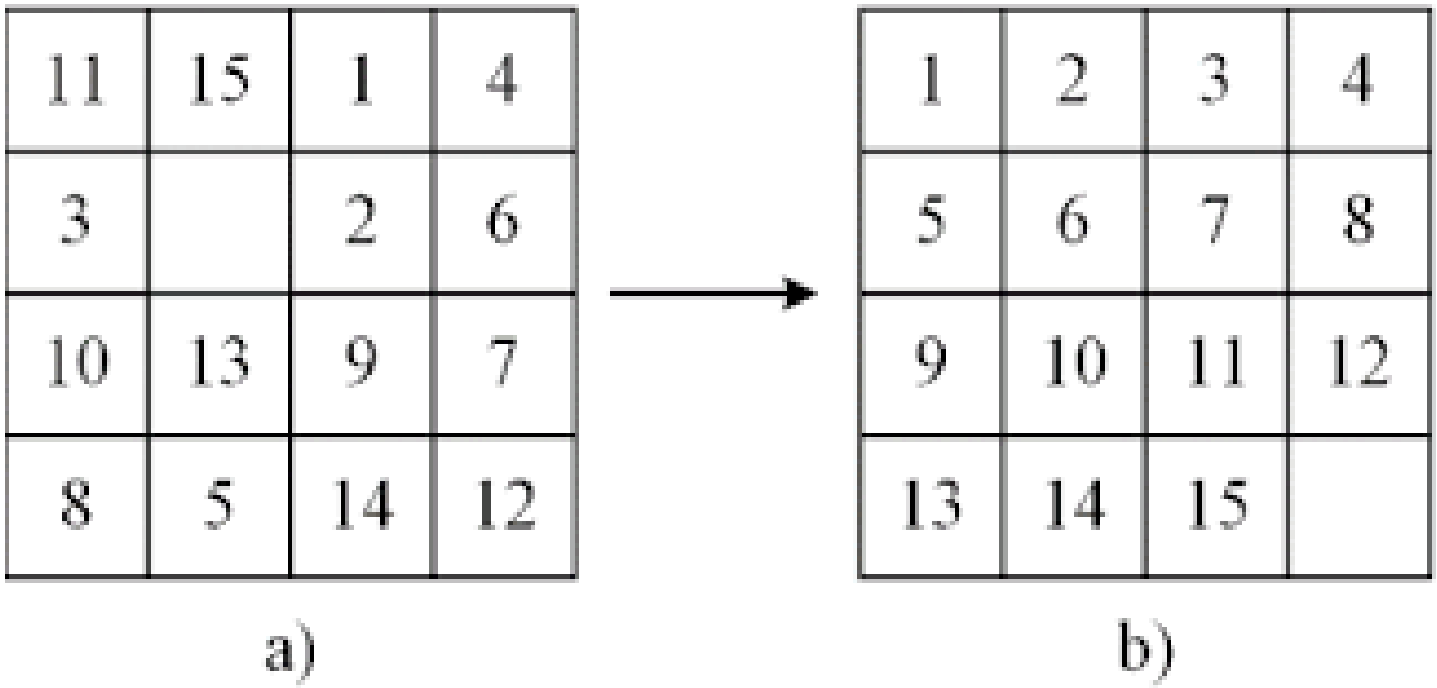
Nicola Barbaro (1070668)
Mario Bifulco (881727)
Franco Sansonetti (1087075)

Università degli studi di Torino
Intelligenza Artificiale e Laboratorio

A.A. 2022/2023

Il gioco, le regole

- La board
- In cosa consiste il gioco
- Come eseguire una mossa sulla board



11	15	1	4
3		2	6
10	13	9	7
8	5	14	12

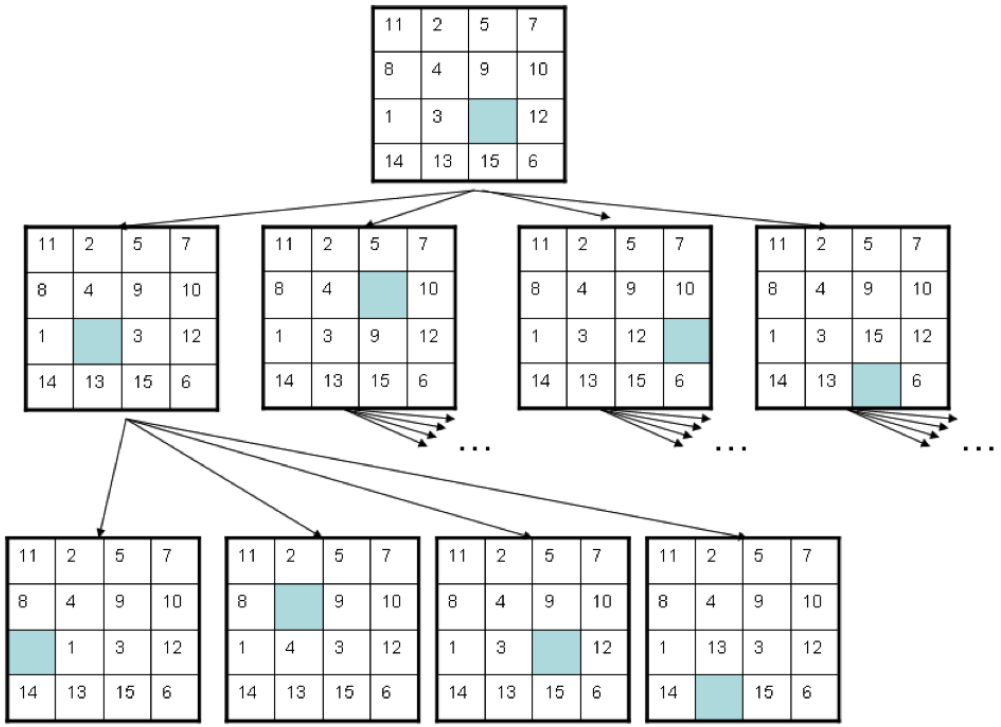
a)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

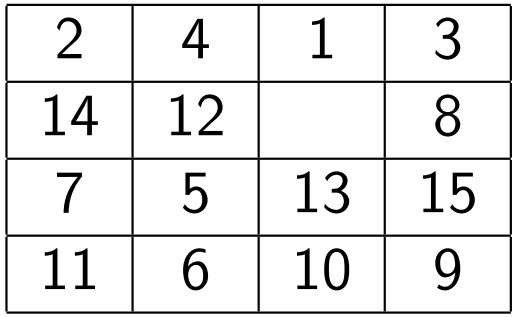
b)

Il problema come ricerca nello spazio degli stati

- Rappresentazione del problema come albero $T = (N, A)$
- Per ogni mossa possibile, un possibile sottoalbero radicato in quella nuova posizione
- Strategie di traversazione dell'albero valutate e testate (*ID*, *IDA**)
- Focus sulla efficienza computazionale nel progetto



Le posizioni in memoria

1. 

2	4	1	3
14	12		8
7	5	13	15
11	6	10	9

2. 2,4,1,3,14,12,v,8,7,5,13,15,11,6,10,9
3. la posizione viene parsata attraverso hashing esadecimale: $0x20401030e0c100807050d0f0b060a09$
4. conversione in intero decimale: 267924570344543520406288274141361204 1

Viene registrato un risparmio in memoria di svariati ordini: in genere, a profondità ≥ 15, da oltre 6GB a pochi MB.

Le mosse (con un esempio: la mossa U - Parte 1)

a partire dalla posizione $pos = 0x20401030e0c100807050d0f0b060a09$

1. Sapendo che la posizione della cella blank è 9, la cella di arrivo dopo la mossa U è in posizione $9 + 4 = 13$
2. La cifra che si trova in posizione 13 non è nota a priori, dato che conserviamo un singolo intero e non una lista, va estratto:
   - dato un numero binario $bucket = bin(15 << 8 * 13)$
   - l'algoritmo estrae $swap = (pos \& bucket) >> 8 * 13 = 1$, che è effettivamente il valore in Up rispetto al blank

Le mosse (con un esempio: la mossa U - Parte 2)

a partire dalla posizione $pos = 0x20401030e0c100807050d0f0b060a09$

5. calcoliamo $diff = blank - swap = 16 - 1 = 15$, la differenza tra il valore di blank (rappresentato come 16) e il valore da swappare
6. Con una strategia del tutto analoga all'accesso alla cella di destinazione, accediamo nuovamente alla posizione 13 per *aggiungere* il valore della variable *diff*, trasformando 1 in 16 (1+15)
7. Ancora, accediamo alla posizione 9 per *sottrarre* il valore della variable *diff*, trasformando 16 in 1 (16-15)
8. le altre celle restano invariate.

risulta: $new\_pos = 0x20410030e0c010807050d0f0b060a09$

Le Euristiche

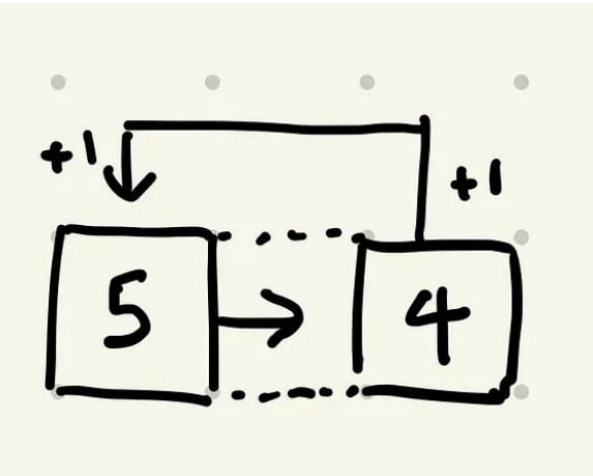
- È una funzione $h(n)$, che stima il costo minimo da ogni vertice n del grafo al goal.
- Ammissibilità: $g(n) \geq h(n)$, ossia non sovrastima mai il reale costo per raggiungere un goal.
- Consistenza: $c(n, n') + h(n') \geq h(n)$, ossia non sovrastima mai il costo verso un successore n' di n sommato a $h(n')$

Le Euristiche: La Distanza Di Manhattan

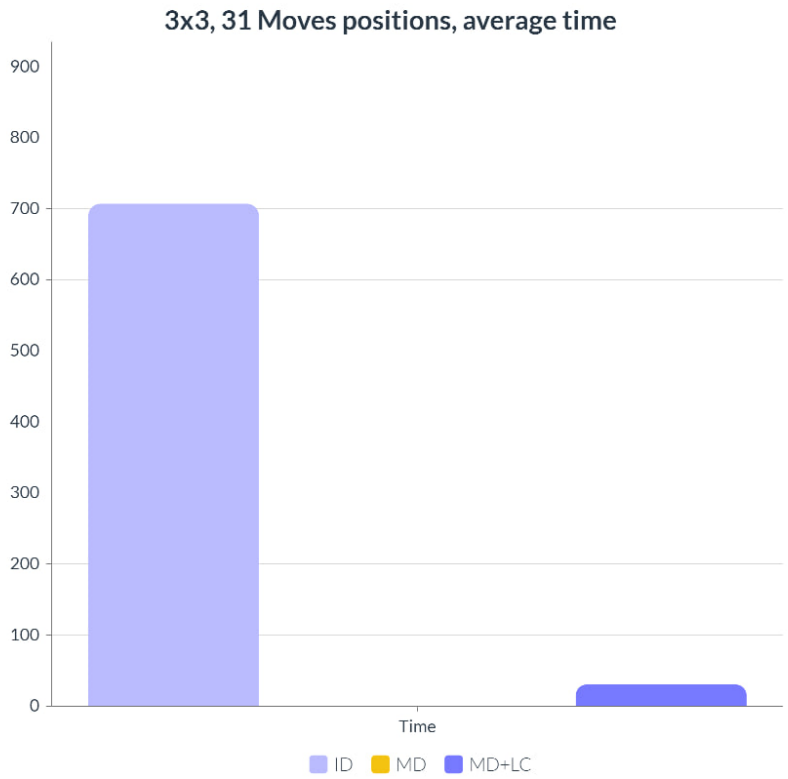
- Somma delle distanze di ogni singola cella sulla board rispetto alla sua posizione originaria
- Sia Ammissibile che Consistente
- L'efficienza risulta importante: Da $O(n^2)$ a $O(1)$
- Facile da Rilassare: overhead permette di risolvere anche le posizioni più complesse in pochi secondi, a discapito dell'ammissibilità

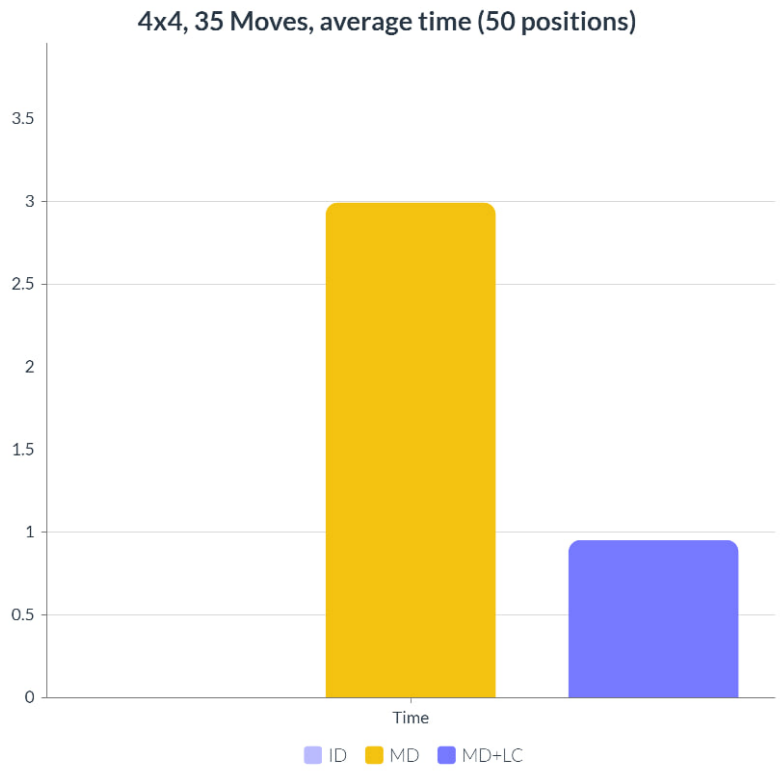
Le Euristiche: I Conflitti Lineari

- Due tiles sono in conflitto lineare se sono sulla riga/colonna corretta ma vanno swappate
- Sia Ammissibile che Consistente
- Complessità $O(n^2)$, necessario uno sviluppo ottimizzato
- Essendo un caso specifico, molto spesso l'euristica aggiunge un overhead di complessità senza ottenere benefici



Risultati





Per la versione del gioco del 15, istanze superiori a 30 mosse richiedevano tempistiche superiori alle ore, e quindi non vengono mostrate.

Risultati (con rilassamento della distanza di Manhattan, overhead = 1.5)

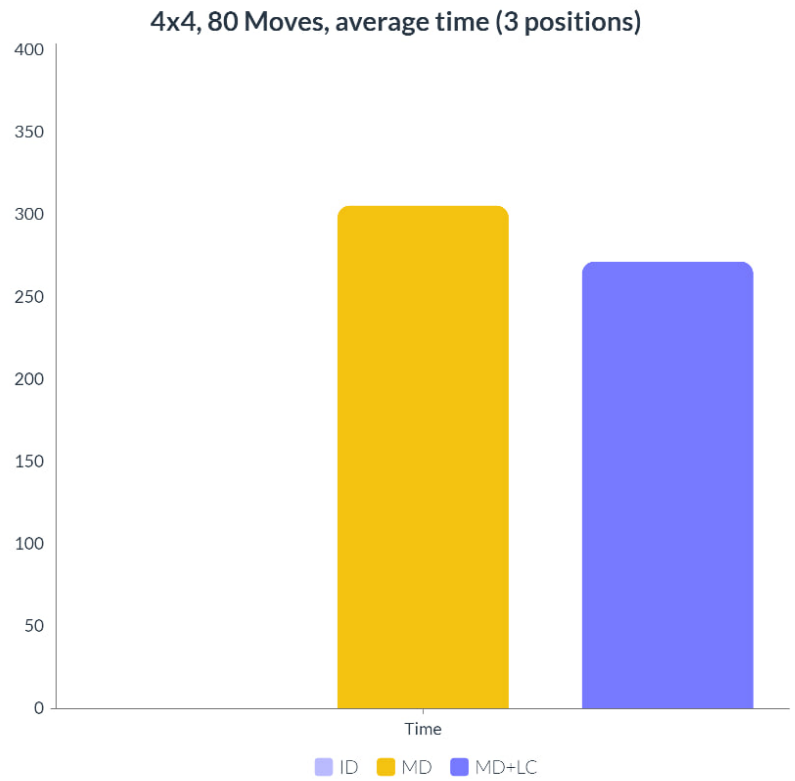


Figura: Pur perdendo di ammissibilità (sono sempre state trovate risoluzioni con +90 mosse contro le 80 teoriche), l'algoritmo in pochi minuti trova la soluzione corretta.

Fine