PROGETTO D'ESAME PER LA PARTE DI PROLOG

IALab A.A. 19/20

Amedeo Racanati 928995

Angelo Pio Sansonetti 928869

Introduzione al problema del labirinto

Il problema del labirinto consiste nel far trovare da parte di un sistema intelligente un percorso che permetta l'uscita dal labirinto stesso. Il sistema parte da una delle celle del labirinto e, muovendosi all'interno dello stesso, deve raggiungere una casella di uscita. Il sistema può muoversi nelle quattro direzioni (nord, sud, est, ovest) e né in diagonale né in una cella contenente un ostacolo.

Per la risoluzione del problema del labirinto sono stati implementati le seguenti strategie di ricerca: *Iterative Deepening, IDA*, A*.*

Nell'*Iterative Deepening* viene eseguita ripetutamente una ricerca a profondità limitata. Il limite della profondità viene incrementato sistematicamente finché non si trova l'uscita. La ricerca è *non informata,* cioè viene esplorato l'universo degli stati possibili senza avvalersi di alcuna conoscenza aggiuntiva del problema. L'algoritmo termina quando si trova l'uscita dal labirinto o quando è stato incrementato il limite ma non si è riusciti ad esplorare nuove celle.

Gli algoritmi *IDA* e A** sono algoritmi di *ricerca informata* dotati di una funzione euristica. La funzione euristica rilassa il problema e permette di indirizzare la ricerca verso quei percorsi più "promettenti". Entrambi gli algoritmi terminano quando si è trovata l'uscita o quando non è possibile esplorare nessun altro nodo nel labirinto.

La *funzione euristica* scelta per gli algoritmi IDA* e A* è la *distanza di Manhattan.* In pratica la distanza tra due punti è definita come la somma delle lunghezze delle proiezioni sugli assi cartesiani dei segmenti che congiungono i due punti, ovvero: $D(P_1,P_2) = |x_1-x_2|+|y_1-y_2|$. Quando siamo in presenza di più vie di uscita, l'euristica restituisce la distanza di Manhattan dell'uscita più vicina, assicurando pertanto la sua ammissibilità.

Per A* è bene citare che è stato adottato il Multiple Path Pruning[1]. L'euristica da noi definita è consistente, ossia per ciascun nodo N e ciascun successore P di N, il costo stimato per raggiungere il goal da N è minore o uguale del costo per arrivare da N a P più il costo stimato per raggiungere il goal da P. Quando viene ampliato un percorso, questa euristica ci permettere di scartare qualsiasi percorso che abbia come ultima cella visitata, una cella che sia stata già visitata da un percorso precedente. Questo perché l'euristica ci assicura che quando una cella viene visitata per la prima volta, il primo percorso trovato che porta a quella cella è il più breve possibile.

[1] https://artint.info/html/ArtInt_61.html

OBIETTIVI

Gli obiettivi di questo progetto sono stati l'implementazione dei tre algoritmi di ricerca, confrontandone le prestazioni rispetto al *tempo* e al *numero di inferenze prodotte.* Abbiamo definito differenti tipi di labirinto considerando:

- Diverse dimensioni del labirinto, ossia 10x10 o 13x13;
- Diverse disposizioni degli ostacoli, ossia pochi ostacoli che permettono diversi percorsi alternativi o molti ostacoli che permettono pochi percorsi alternativi;
- La possibilità che ci siano una o più uscite dal labirinto;
- La possibilità che nessuna delle uscite risulti raggiungibile.

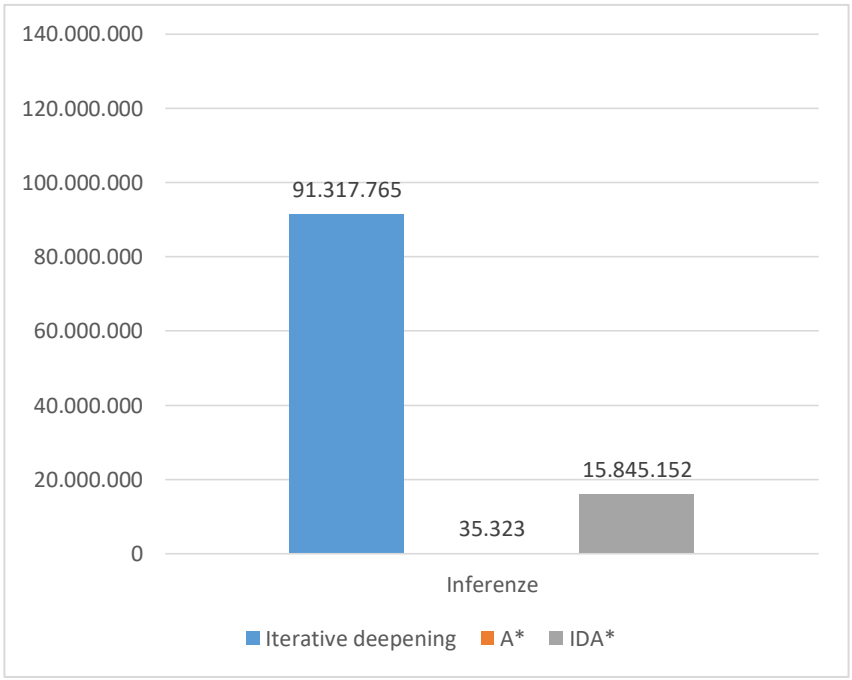
Di seguito verranno confrontate le prestazioni dei diversi algoritmi, eseguiti su un processore Intel dual core 2.50 Ghz..

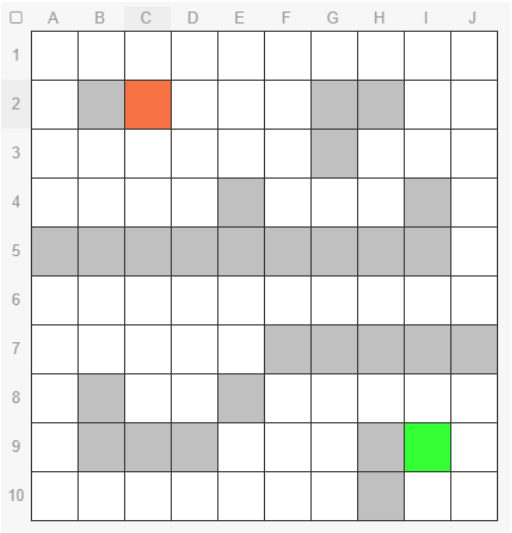
Note sul progetto

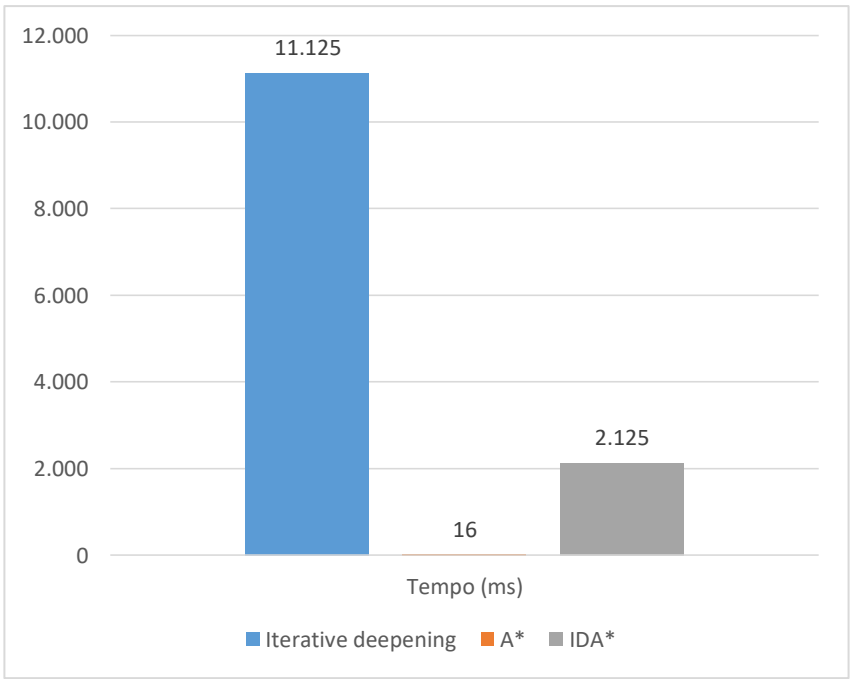
Il codice Prolog è strutturato in maniera tale da suddividere da una parte i diversi algoritmi di ricerca e dall'altra di definire in ciascun file un particolare labirinto.

Per eseguire il codice bisogna effettuare la consult del file "loader.pl", grazie alla quale viene effettuata l'importazione degli altri file che servono per la risoluzione del problema. All'interno del file loader è possibile definire l'algoritmo di ricerca e il labirinto. Una volta effettuata la consult, per avviare la ricerca bisogna digitare il comando "start.", che provvederà a stampare una soluzione del problema, qualora sia presente.

Labirinto 10x10 con pochi ostacoli[2]

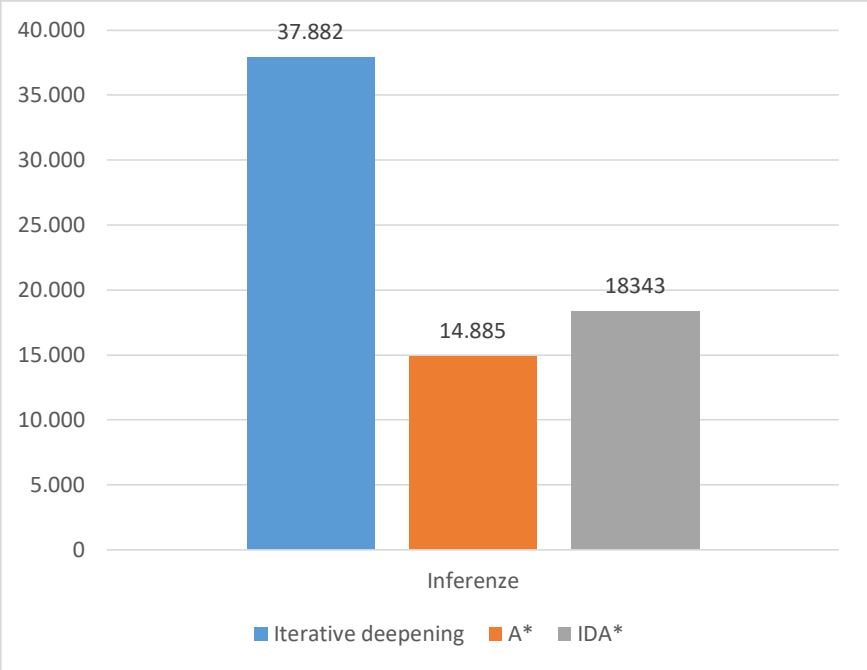


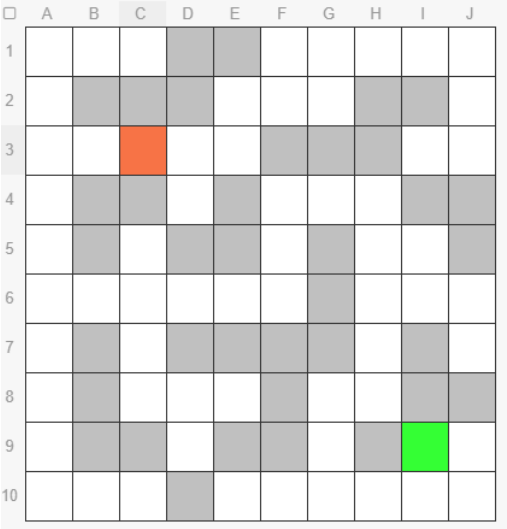


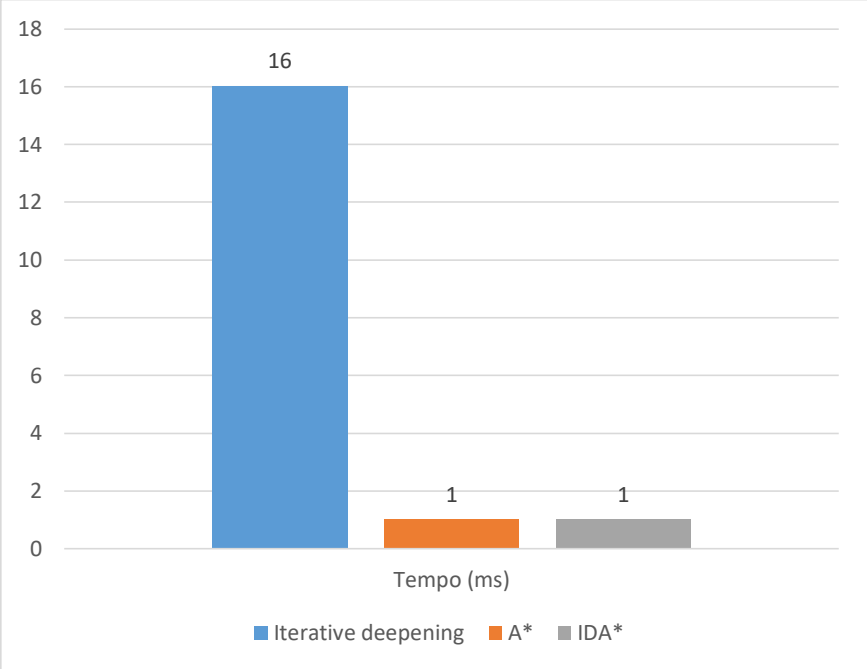


[2] La cella rossa indica la posizione di partenza, la cella verde indica l'uscita. Le celle più scure indicano gli ostacoli

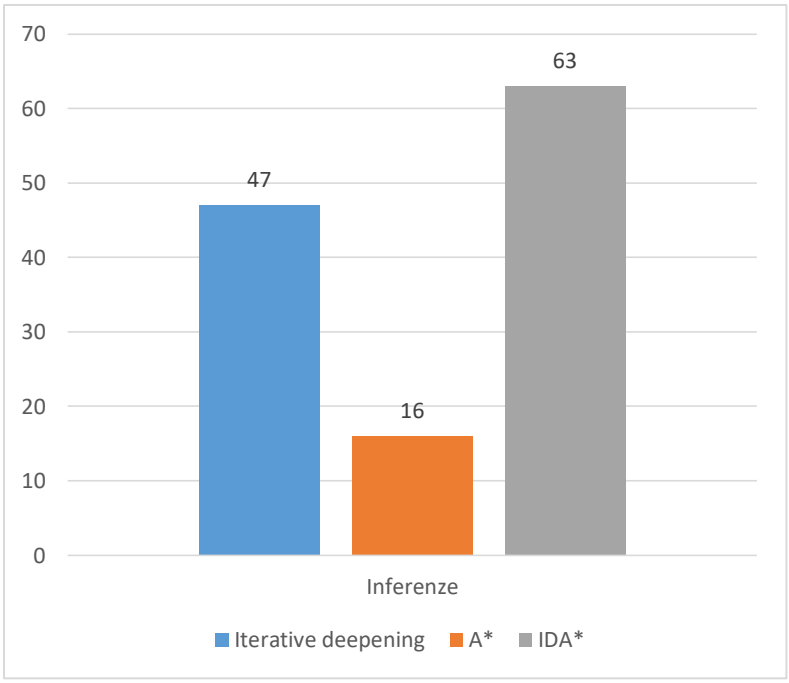
Labirinto 10x10 con molti ostacoli

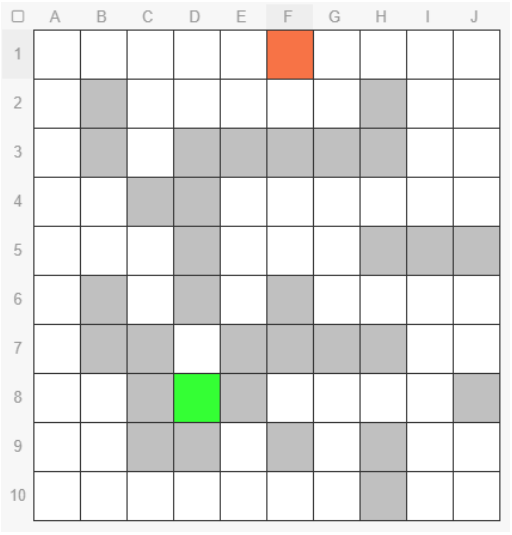


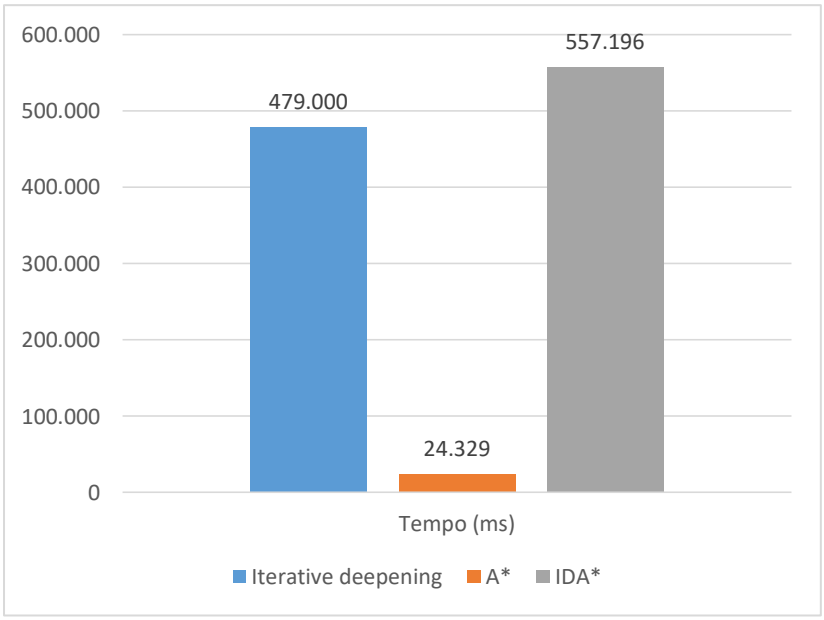




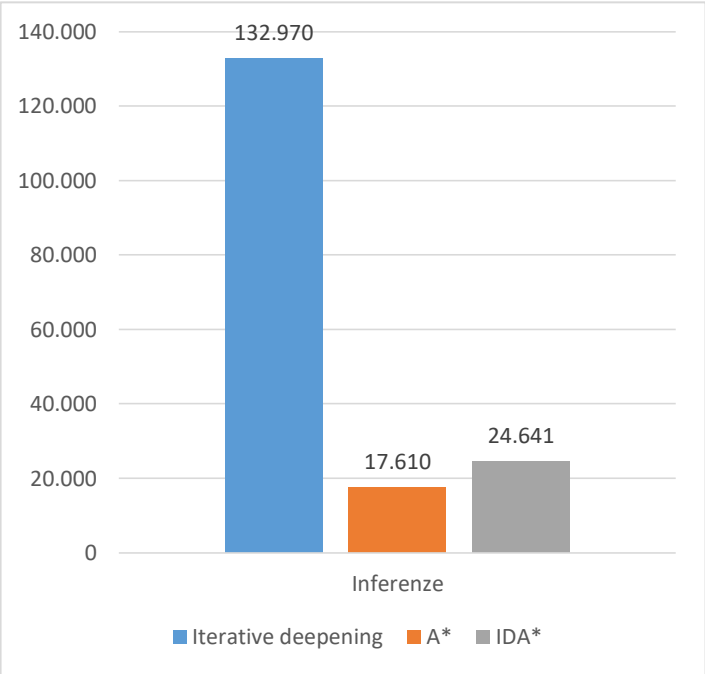
Labirinto 10x10 con nessuna uscita

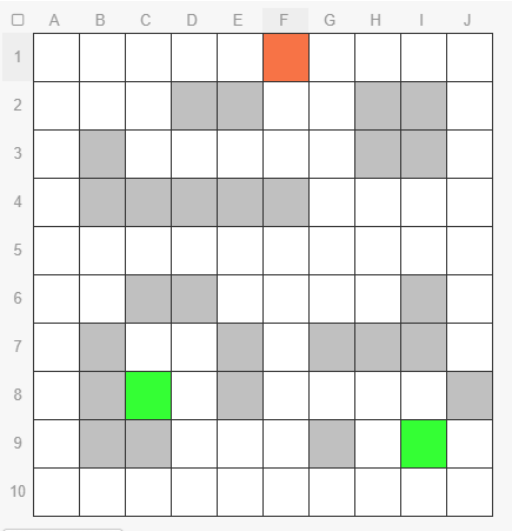


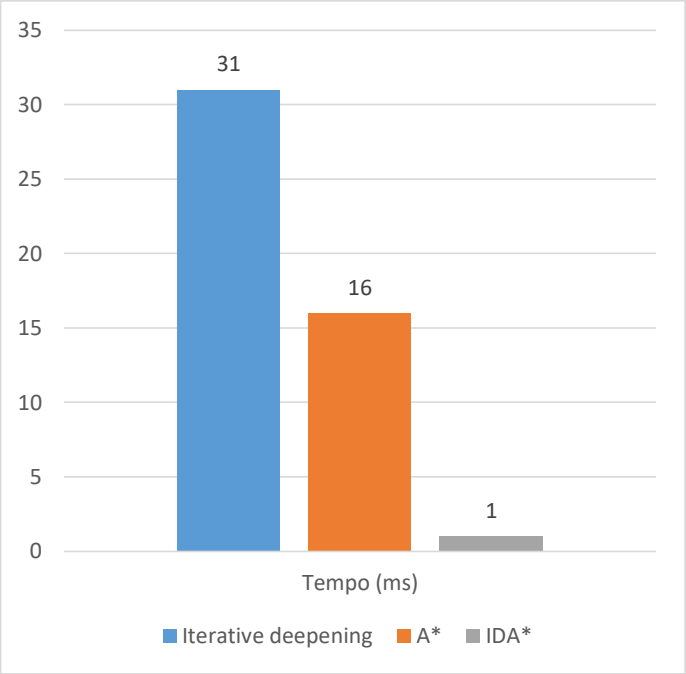




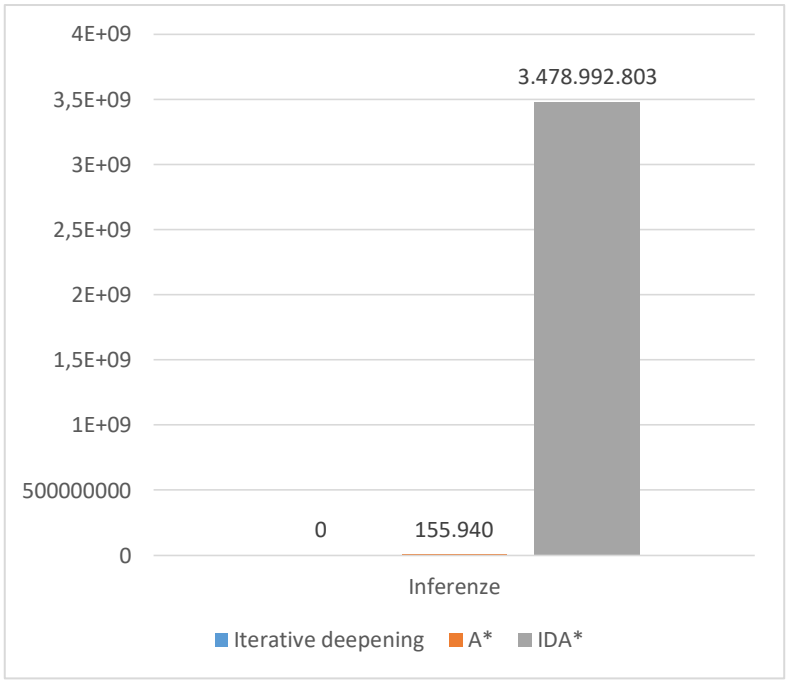
Labirinto 10x10 con diverse uscite

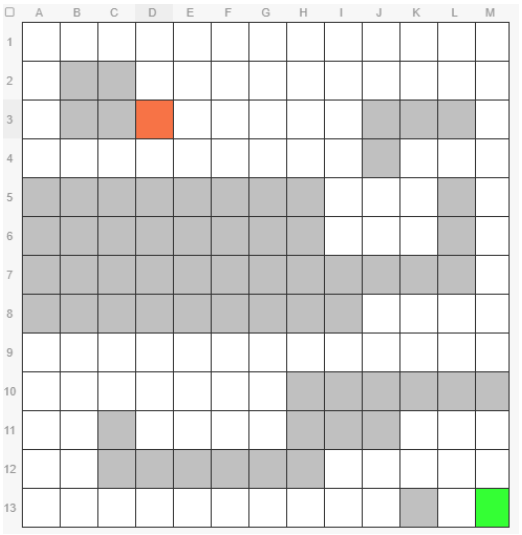


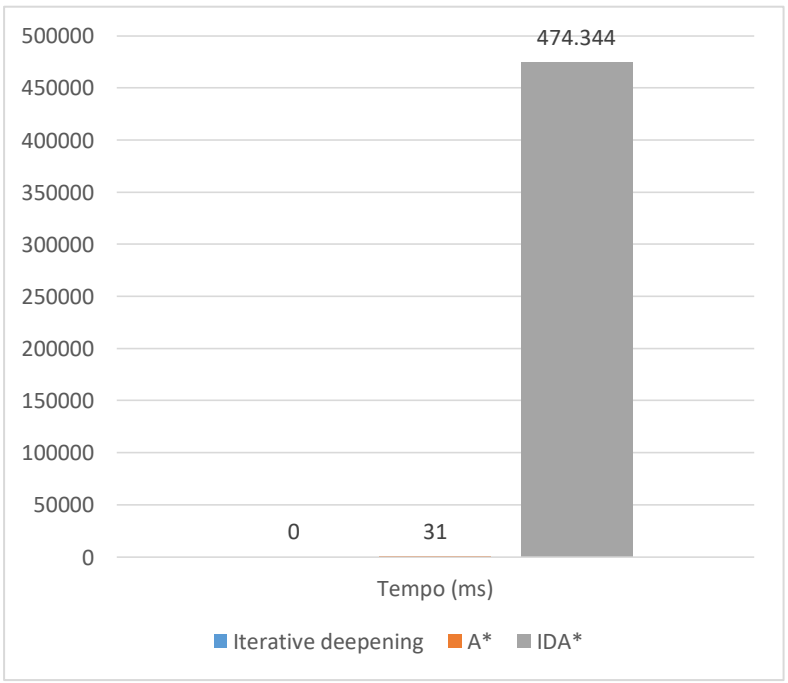




Labirinto 13x13 con pochi ostacoli[3]

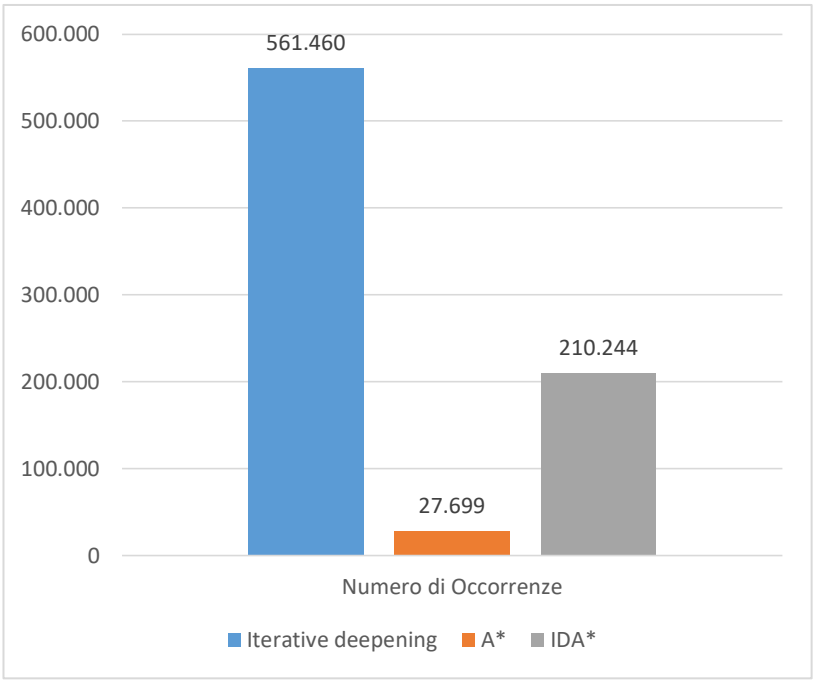


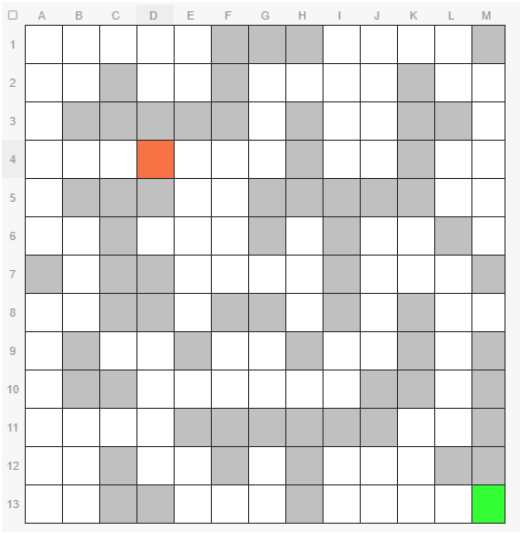


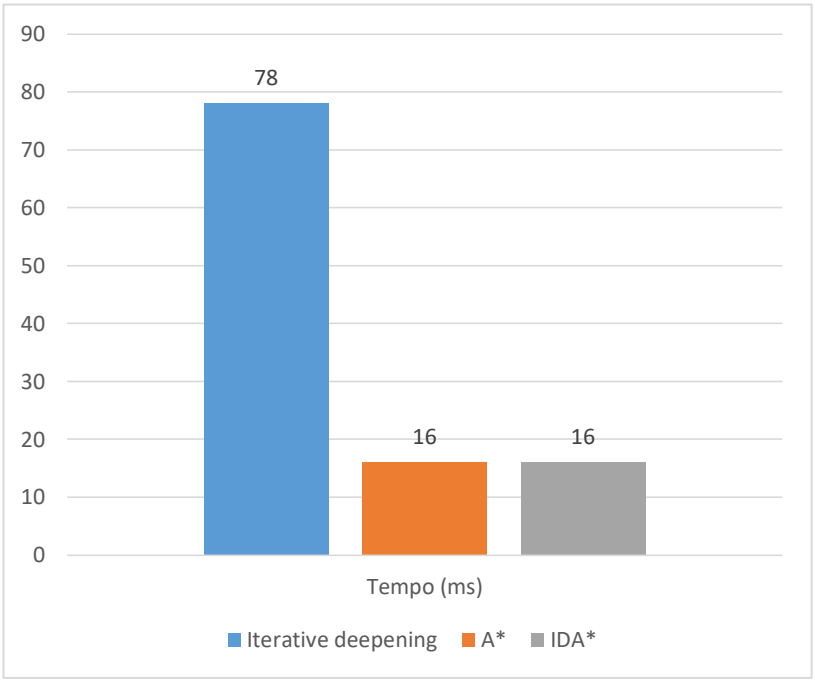


[3] L'algoritmo Iterative depening non ha trovato la soluzione al problema entro 8 ore, pertanto non abbiamo potuto inserire i risultati nel grafico.

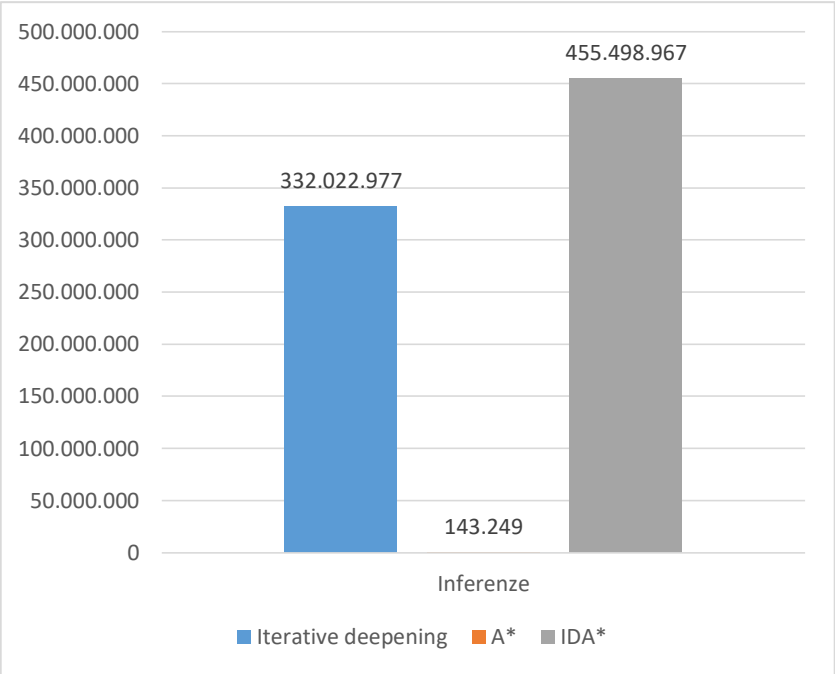
Labirinto 13x13 con molti ostacoli

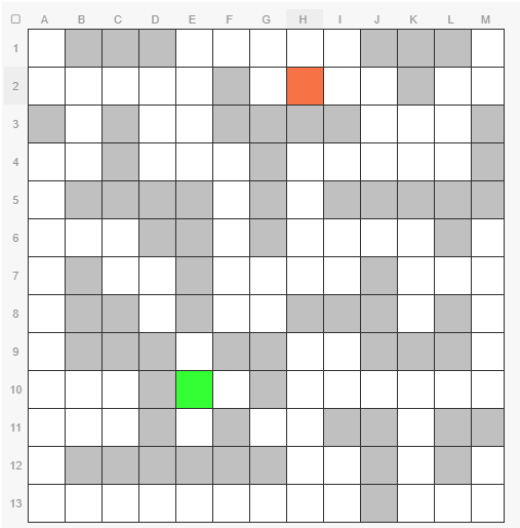


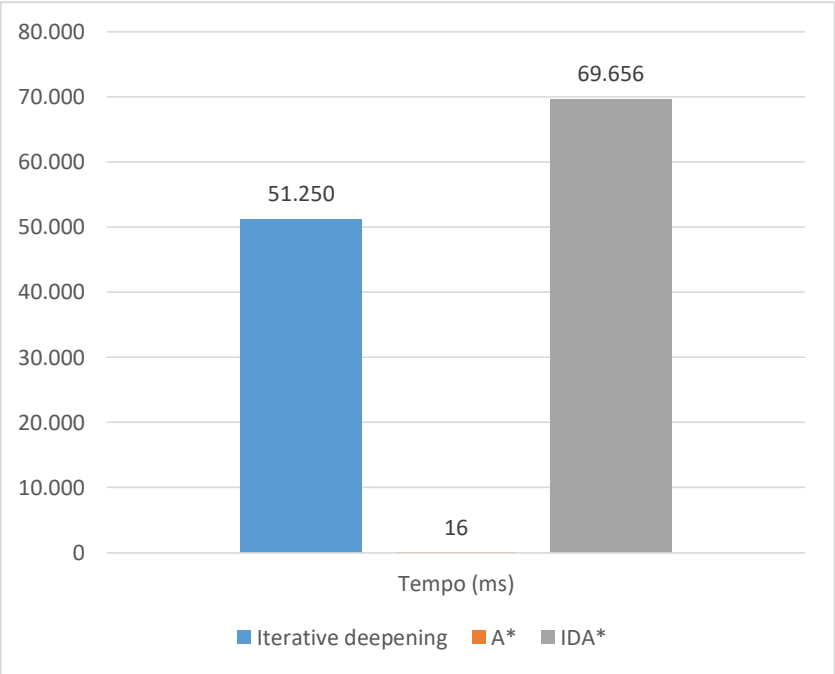




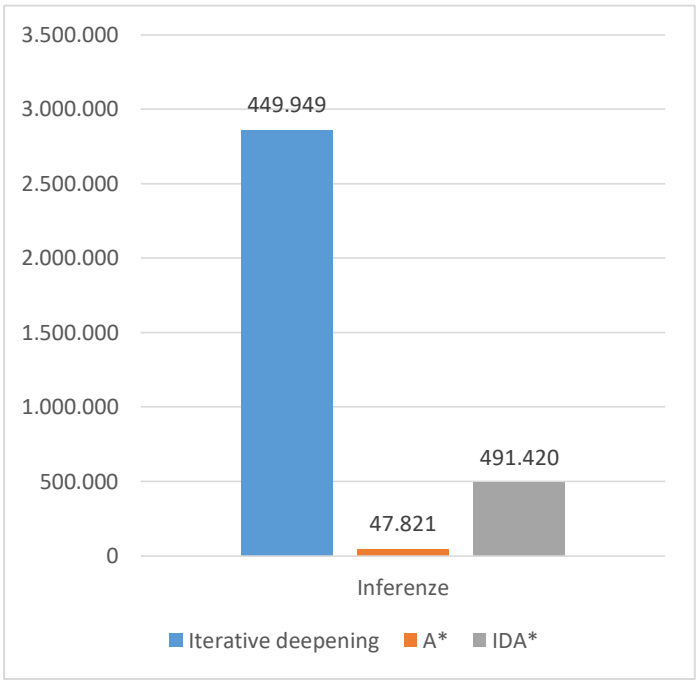
Labirinto 13x13 con nessuna uscita

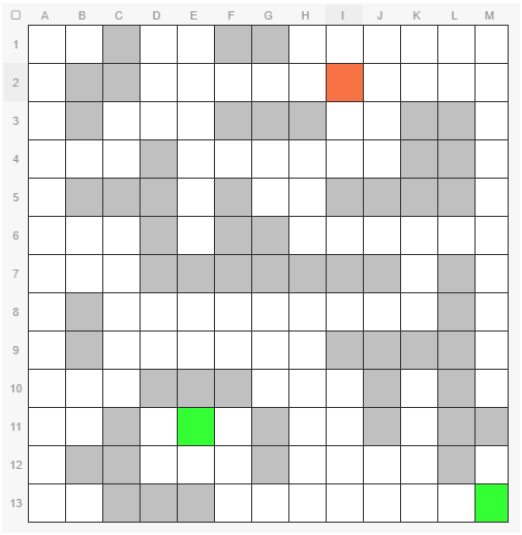


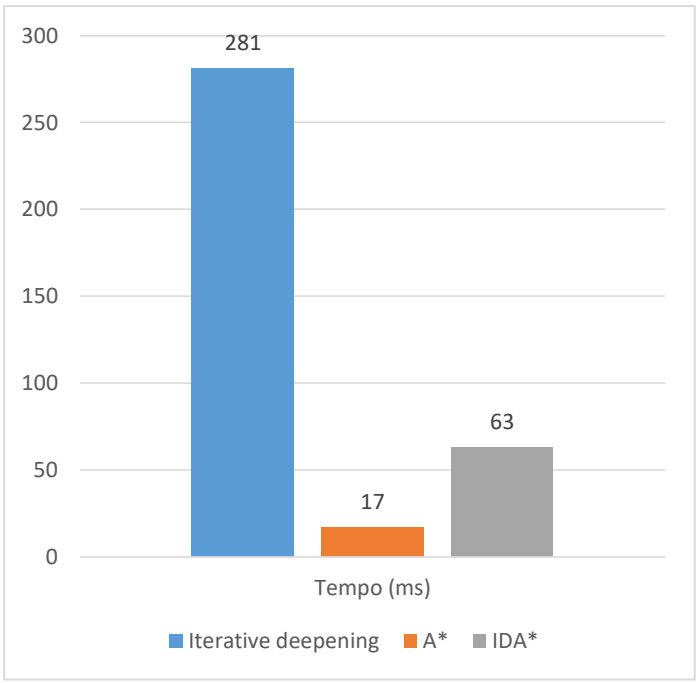




Labirinto 13x13 con diverse uscite







CONSIDERAZIONI

In generale l'algoritmo A* è risultato più veloce rispetto agli altri due algoritmi e in alcuni casi, in particolare nei labirinti aventi pochi ostacoli, la differenza è notevole. Di solito il tempo impiegato da A* si mantiene sull'ordine delle decine di millisecondi, sebbene il numero di inferenze varia a seconda dei casi.

In presenza di molti ostacoli IDA* si avvicina alle tempistiche di A*, in quanto IDA* si trova "costretto" ad esplorare pochi percorsi alternativi.

Quando invece l'uscita non è accessibile, gli algoritmi iterativi presentano un grosso difetto dato che impiegano più tempo del solito per capire se non esiste nessuna soluzione. Inoltre in questi casi IDA* è più lento di Iterative Deepening, a causa dell'overhead legato al mantenimento dell'euristica.

Quando la grandezza del labirinto aumenta, mentre per A* le prestazioni rimangono pressoché uguali, per ID e IDA* c'è un aumento dei tempi che è sovralineare rispetto all'aumento dei blocchi (che nel 13x13 corrisponde a circa il 70% di blocchi in più). Mentre in presenza di diverse uscite o di molti ostacoli questi due algoritmi si mantengono su dei tempi accettabili, quando il labirinto presenza pochi ostacoli i tempi di risoluzione aumentano in maniera spropositata. Questo è dovuto al fatto che questi algoritmi, prima di incrementare la soglia di esplorazione, devono prima percorrere tutti i possibili percorsi con la soglia corrente: pertanto in presenza di pochi ostacoli il numero dei percorsi alternativi è piuttosto elevato e di conseguenza porta via parecchio tempo.

Sono stati ottimizzati gli algoritmi ID e IDA* per far sì che si arrestino qualora tra una iterazione e un'altra non hanno esplorato nuove celle. Questo ha permesso di far terminare l'algoritmo in tempi più ragionevoli qualora la soluzione al problema non sia presente.