Risoluzione del labirinto a più uscite mediante Prolog

Dipartimento di Informatica: Università di Torino

Autori: Alessandro Saracco, Leonardo Magliolo , Mattia Marra

Intelligenza artificiale e laboratorio

A.A. 2021/2022

Generazione dei labirinti

La generazione dei labirinti è stata effettuata mediante uno script in Python considerando i seguenti input:

- Numero di righe
- Numero di colonne
- Posizione iniziale
- Posizioni finali

I muri vengono generati casualmente, la probabilità che un muro venga inserito all'interno di una specifica casella (eccezion fatta per caselle assegnate a posizioni iniziali e finali) è stata impostata pari a una costante compresa tra 0% e 50%.

Scelta dell'algoritmo

Come algoritmo da implementare in Prolog è stato scelto A* considerate le seguenti caratteristiche:

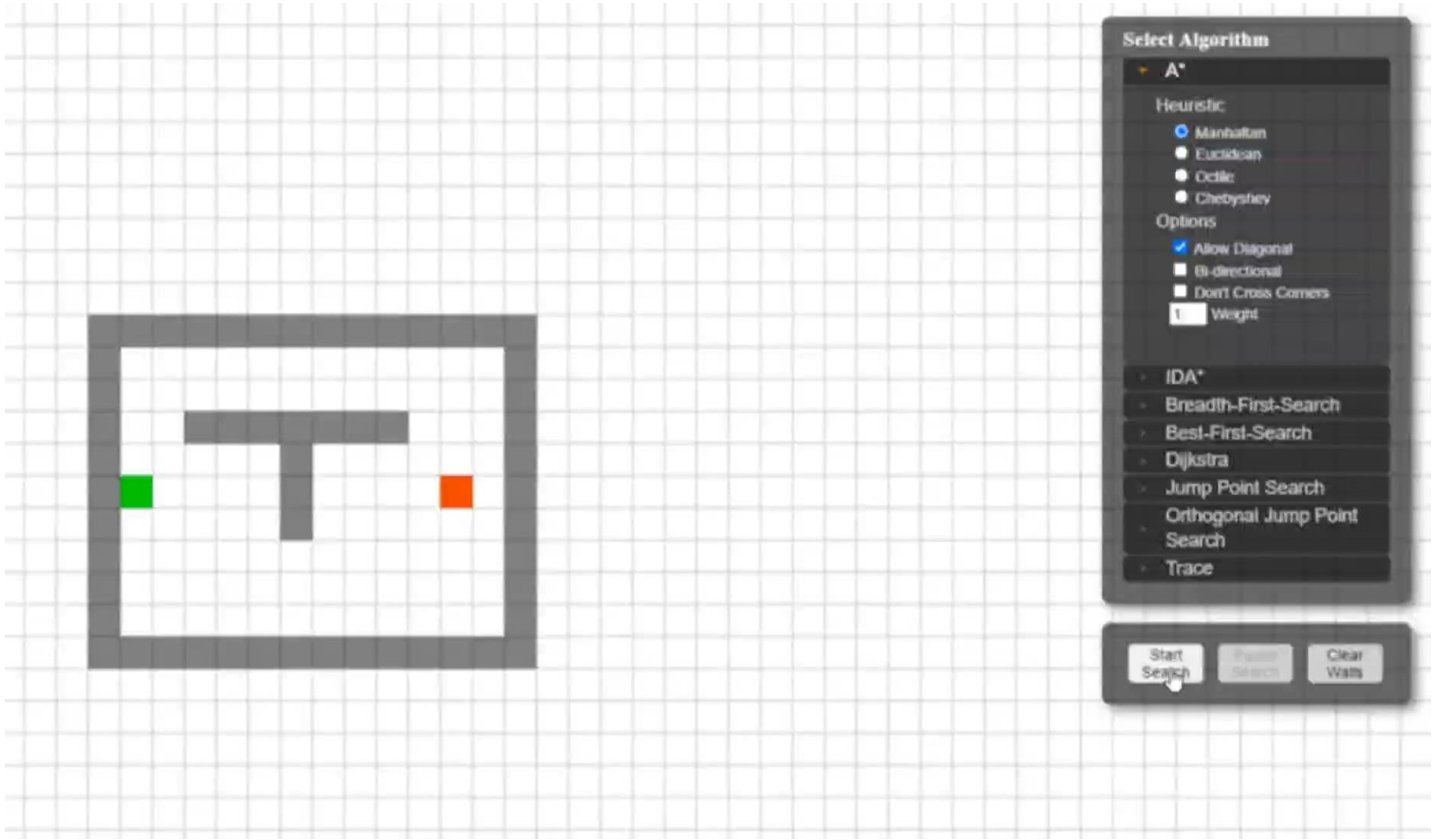
- Effettua ricerca informata
- Completo nel dominio considerato
- Se l'euristica utilizzata per la ricerca informata è monotona allora è ottimale
- Non esiste un altro algoritmo che garantisce un minor numero di nodi espansi per trovare una soluzione ottimale

Scelta dell'algoritmo

Prima dell'implementazione di A* sono state effettuate ricerche circa l'efficacia dei possibili algoritmi applicati al problema di riferimento.
Il seguente sito è stato particolarmente utile al fine di confrontare A* e IDA*.

Considerata la macchina su cui l'algoritmo è stato eseguito, la dimensione del problema che si è scelto di trattare (non più di labirinti 1000x1000) ed eventuali tempistiche dovute alla fase di test, è stato scelto di implementare A* piuttosto che IDA*.

Esempio IDA* vs A* con euristica Manhattan



Scelta delle euristiche

A seguito delle ricerche effettuate è stato scelto di implementare due versioni di A*, una utilizzando l'euristica di Manhattan e una con distanza euclidea.
Sono state implementate in quanto largamente studiate in letteratura in relazione alla risoluzione del problema del labirinto a più uscite.

Le euristiche scelte non sono monotone nello spazio di ricerca considerato (in quanto è possibile muoversi in diagonale con una sola azione), pertanto le soluzioni calcolate dall'algoritmo implementato risultano essere subottimali.

Analisi prestazionale: Tipologie di labirinto

Per l'esecuzione dell'analisi prestazionale effettuata è stato scelto eseguire l'algoritmo su labirinti quadrati aventi lunghezza dei lati: 25, 50, 100, 500 e 1000.

Sono state generate tre mappe per ogni lato elencato in precedenza:

- una senza muri con soluzioni (indicata come NM)
- una con muri con soluzioni (indicata come MS); la probabilità che un muro sia inserito all'interno di una determinata casella è del 50%
- una con caratteristiche identiche a quella indicata al punto precedente ma senza soluzioni (indicata come NS)

Analisi prestazionale: Specifiche hardware

Specifiche hardware:

- CPU: AMD Ryzen 7 5800X
- RAM: Corsair VENGEANCELPX16GB DDR4
- SSD: Samsung MZ-V8V1T0 980

Analisi delle prestazioni: Dati raccolti

A* con euristica di Manhattan

Labirinto	Time	Branching Factor	Depth	Cost	Out of Stack
25 x 25 MS	< 1 s	29.94	35	1048	False
25 x 25 NM	< 1 s	13.66	24	328	False
25 x 25 NS	< 10 s	/	/	/	False
50x50 MS	< 1 s	8.65	55	476	False
50x50 NM	< 1 s	13.83	49	678	False
50x50 NS	< 3 m	/	/	/	True 10 GB
100x100 MS	< 1 s	10.77	103	1110	False
100x100 NM	< 1 s	13.91	99	1378	False
500x500 MS	< 2 s	14.03	516	7240	False
500x500 NM	< 1 s	13.98	499	6978	False
1000x1000 MS	< 40 s	26.34	1065	28060	True 1GB, False 8 GB
1000x1000 NM	< 4 s	13.99	999	13978	False

A* con euristica distanza euclidea

Labirinto	Time	Branching Factor	Depth	Cost	Out of Stack
25 x 25 MS	< 0.5 s	19.5	34	663	False
25 x 25 NM	< 0.5 s	12.91	24	310	False
25 x 25 NS	< 2 s	/	/	/	False
50x50 MS	< 1 s	14.8	51	755	False
50x50 NM	< 0.5 s	13.1	49	642	False
50x50 NS	> 5 m	/	/	/	True 10 GB
100x100 MS	< 1 s	12.43	99	1231	False
100x100 NM	< 0.5 s	13.16	99	1304	False
500x500 MS	< 2 s	17.43	500	8716	False
500x500 NM	< 1 s	13.22	499	6604	False
1000x1000 MS	< 4 s	11.66	999	11649	False
1000x1000 NM	< 2.5 s	13.24	999	13229	False

Analisi delle prestazioni: Considerazioni sui risultati

Dai dati raccolti è possibile notare come l'euristica della distanza euclidea non risulti decisamente più vantaggiosa rispetto alla distanza di Manhattan, per la maggior parte delle mappe con lato ridotto considerate in fase di test.
E' risultata particolarmente vantaggiosa, invece, nella della ricerca all'interno nei labirinti il cui lato ammonta a 1000, sia in termini di tempistiche che nell'utilizzo di memoria.
Nel caso del labirinto 1000x1000 MS viene trovata una soluzione in meno di quattro secondi attingendo a meno di un GB di memoria RAM, utilizzando invece l'euristica di Manhattan il calcolo della soluzione ha impiegato circa 40 secondi e 8 GB di RAM al picco massimo d'acquisizione.

Analisi delle prestazioni: Considerazioni sui risultati

Sebbene l'algoritmo A* abbia ottenuto discreti risultati circa le tempistiche con cui le soluzioni sono state computate, è evidente che nei labirinti in cui la soluzione non fosse presente la quantità di memoria RAM utilizzata per l'espasione dei nodi sia risultata inadeguata rispetto alla macchina che lo ha eseguito, anche per mappe piuttosto ridotte.

Nelle casistiche in cui un determinato labirinto causi stack overflow con A*, per determinare se una soluzione, esiste avrebbe più senso utilizzare algoritmi che riducono i nodi espansi presenti in frontiera, come IDA*.

Analisi delle prestazioni: Considerazioni sui risultati

Ponendo il lato della mappa pari a l allora la profondità del ramo più profondo esplorato sarebbe <= di l^2.
Ponendo il branching factor pari a 8 (numero di mosse effettuabili nel caso peggiore), allora il numero massimo di nodi trattenuti in memoria da A* sarebbe $\in \mathcal{O}(8^{l^2})$ mentre nel caso di IDA* la stessa quantità ammonterebbe a $\mathcal{O}(l^2)$.