Intelligenza Artificiale e Laboratorio

Relazione del progetto di laboratorio

Stefano Locci, Gabriele Picco

Anno Accademico 2018 - 2019

Programmazione Logica

1. Implementazione delle strategie di ricerca
   1.1. Strategie non informate
      1.1.1. Iterative Deepening
   1.2. Strategie con Euristica
      1.2.1. A*
      1.2.2. IDA*
2. Descrizione del dominio
3. Funzioni euristiche
4. Confronto tra le strategie di ricerca
5. Analisi delle prestazioni con costi non unitari
6. Completezza e ottimalità delle strategie di ricerca
   6.1. Soluzione non appartenente allo spazio degli stati

1. Implementazione delle strategie di ricerca

Di seguito vengono riportate le principali scelte implementative e le considerazioni relative agli algoritmi Iterative Deepening, A*, IDA* in linguaggio Prolog.

1.1. Strategie non informate

Le strategie non informate cercano di esplorare in modo completo lo spazio degli stati nella ricerca della soluzione, mediante l'espansione combinatoria degli stati.

Esse non utilizzano alcun tipo di conoscenza del problema al di là della definizione del problema stesso.

1.1.1. Iterative Deepening

La strategia di ricerca Iterative Deepening sfrutta una ricerca in profondità limitata, incrementando il limite (inizialmente impostato a 1) iterativamente.

L'implementazione in prolog è simile ad una ricerca in profondità limitata, con l'aggiunta di un predicato ausiliario per l'incremento della soglia in caso di fallimento.

Viene inoltre gestito come caso particolare la situazione in cui la soluzione non appartenga allo spazio degli stati, come descritto nella sezione 6.1.

Il predicato ausiliario, che si occupa dell'incremento iterativo della soglia, asserisce il fatto maybe_solvable(false) all'inizio del corpo della regola, prima del predicato ricorsivo per la DFS limitata.
Il fatto maybe_solvable(false) viene rimosso e sostituito con maybe_solvable(true) dal primo ramo della DFS limitata che fallisce poiché limitato dalla soglia. E' infatti possibile che la soluzione si trovi a distanza maggiore della soglia, perciò ha senso proseguire la ricerca.

In caso contrario, se cioè nessun fallimento è stato causato dal limite, l'intero spazio degli stati è stato esplorato. La ricerca perciò termina restituendo il fallimento.

1.2. Strategie con euristica

Le strategie di ricerca con euristica, al contrario delle strategie non informate, sfruttano la conoscenza del dominio del problema, per guidare l'esplorazione dello spazio degli stati.

L'algoritmo di ricerca informata è quindi dotato sia della funzione di ricerca che di una funzione di conoscenza (funzione euristica). La funzione di conoscenza determina l'ordine di ricerca dell'algoritmo al fine di ridurre il tempo necessario e/o il costo della ricerca.

1.2.1. A*

L'algoritmo A* basa l'esplorazione dello spazio degli stati su una funzione di stima corrispondete alla somma dell'euristica e il costo del cammino per arrivare allo stato S:

F(S) = Euristica(S) + G(S).

La ricerca della soluzione utilizza la politica best-first, ovvero si espande sempre il nodo con F(S) minore. Tale strategie richiede che si memorizzi e si mantengano ordinati i nodi ed i relativi valori F(S) per tutti i nodi alternativi alla scelta corrente (frontiera).

L'implementazione risulta perciò simile ad una BFS, ma tenendo in considerazione i valori F(S) per l'esplorazione. L'implementazione in prolog memorizza la frontiera in una lista di nodi, dove ogni nodo corrisponde ad una quadrupla (F(S), G, Stato, Azioni).
La lista viene mantenuta ordinata dal predicato build-in sort, ad ogni espansione della frontiera.

L'algoritmo inoltre deve considerare che non vengano espansi nodi i cui successori appartengono alla frontiera o siano già presenti nella soluzione parziale, per evitare situazioni cicliche.

1.2.2. IDA*

L'algoritmo IDA* è una variante di Iterative Deepening, che sfrutta l'idea di di utilizzare una funzione stima per valutare il costo rimante per raggiungere l'obiettivo, come in A*.

Poiché si tratta di un algoritmo di ricerca in profondità, l'utilizzo della memoria è inferiore rispetto ad A*. Lo svantaggio dell'approccio è che lo stesso nodo può essere espando più volte, poiché non viene memorizzata la frontiera come in A*.

L'implementazione in Prolog è molto simile a quella descritta precedentemente per Iterative Deepening, con l'unica differenza che la soglia che limita l'esplorazione è basata sulla funzione F(S) (la medesima di A*) e la funzione ausiliaria che richiama la ricerca in profondità, dopo il fallimento, aggiorna la soglia basandosi sul minimo valore F(S) superiore alla soglia.
Questo valore può essere facilmente ricavato asserendo un predicato fvalue(F(S)) al momento del fallimento, nel caso in cui il predicato fvalue(X) abbia valore maggiore di F(S) o uguale alla soglia precedente.

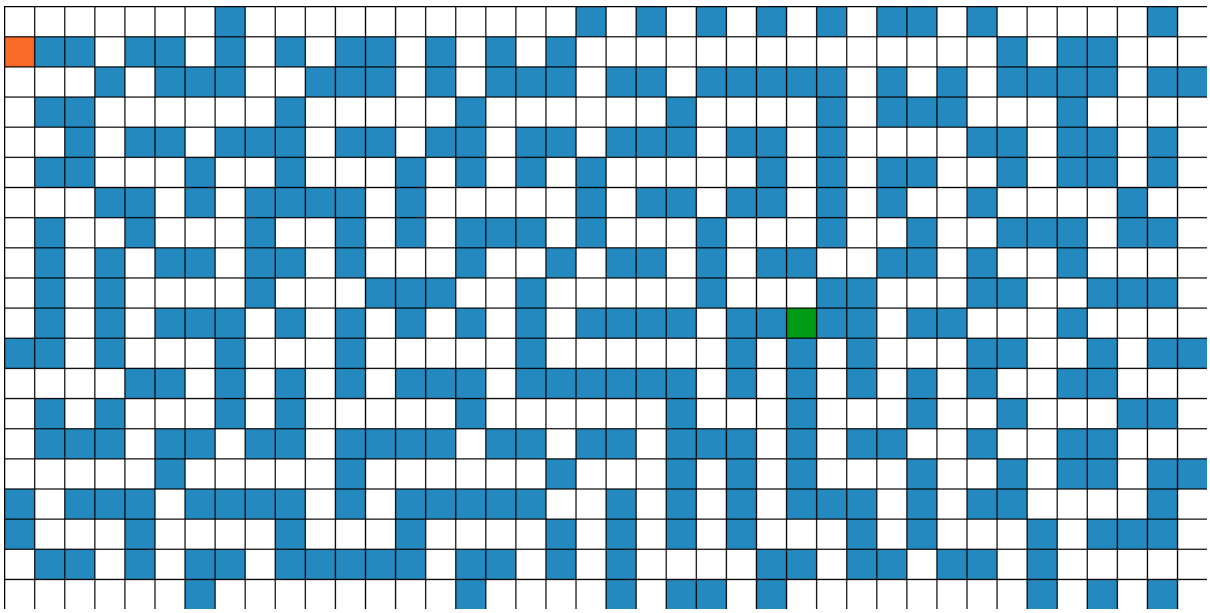
Basandosi inoltre sul fatto che la FSoglia è nota, prima di proseguire la ricerca impostando un nuovo valore di FSoglia, si può vericare che essi non corrispondano, evitando di ripetere una computazione uguale. Si può ricavare inoltre che tale condizione si verifica quando lo spazio degli stati è stato esplorando completamente, ma la soluzione non è stata trovata. In questo modo la non esistenza della soluzione risulterà in una esplorazione completa dello spazio degli stati ed un fallimento.

2. Descrizione del dominio

Il domino di interesse simula la ricerca di un percorso, all'interno di un labirinto, da uno stato iniziale ad uno stato finale. Ogni casella C è rappresentata da un termine pos(X,Y) dove X e Y sono le coordinate spaziali. Gli ostacoli sono modellati da un termine occupata(C). finale(C) e iniziale(C) rappresentano rispettivamente lo stato finale e iniziale. num_colonne(CL) e num_righe(NR) identificano la dimensione del labirinto.

Il predicati *applicabile*, *trasforma* e *costo* determinano rispettivamente l'applicabilità delle azioni, le transizioni di stato e il costo delle azioni.

Per effettuare la sperimentazione è stato implementato in Python un generatore algoritmico di labirinti, che costruisce un tabellone CL x NR fornendo il codice prolog corrispondente e la relativa immagine.



2. Funzioni Euristiche

Una funzione euristica in una strategia di ricerca fornisce informazioni all'algoritmo (sfruttando la conoscenza del dominio) per ridurre il processo di ricerca della soluzione.

Due proprietà importanti delle euristiche, che incidono sulla completezza e l'ottimalità degli algoritmi che le sfruttano sono ammissibilità e consistenza.

L'ammissibilità asserisce che la funzione euristica non stimi mai per eccesso il costo per arrivare all'obiettivo: $h(n) <= h^*(n)$.

La consistenza di un euristica è una condizione più forte (consistente => ammissibile), che corrisponde alla monotonicità della funzione.
Un'euristica è consistente se $h(n) <= c(n, a, n') + h(n')$.

Le euristiche possono essere costruite rilassando la definizione del problema, memorizzando in un database di pattern i costi precalcolati delle soluzioni, o apprendendo dall'esperienza.

Di seguito vengono riportate 3 euristiche facilmente definibili nel dominio del labirinto, tutte consistenti.

Distanza di Manhattan: la distanza tra due punti è la somma del valore assoluto delle differenze delle loro coordinate.

$D(C_1,C_1) = |x1 - x2| + |y1 - y2|$

Distanza Massima orizzontale o verticale (Max Distance):

$D(C_1,C_1) = \max(|x1 - x2|, |y1 - y2|)$

Distanza Euclidea: misura del segmento avente per estremi i due punti approssimata all'intero più vicino.

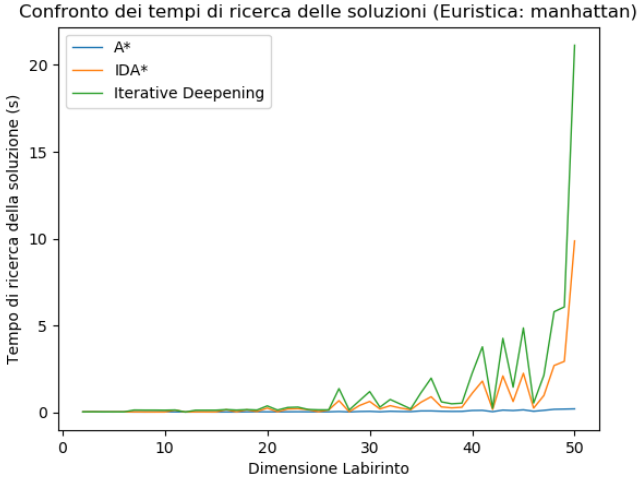
$D(C_1,C_1) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

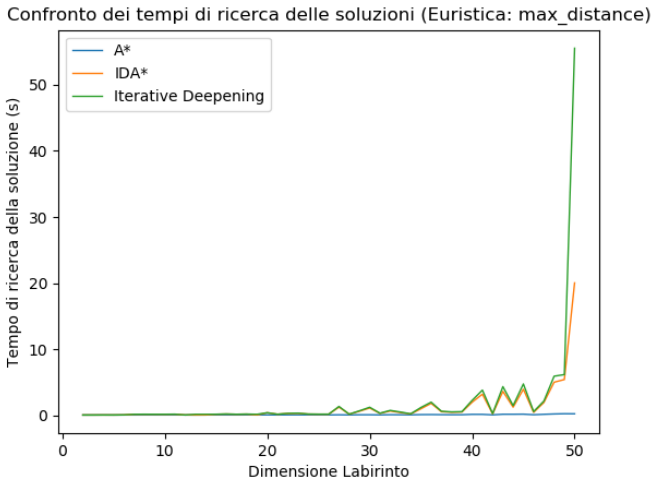
Le euristiche definite verrano utilizzate per confermare le difference di performance, poiché le prestazioni degli algoritmi di ricerca euristici dipendono dalla qualità delle funzioni euristiche.

4. Confronto tra le strategie di ricerca

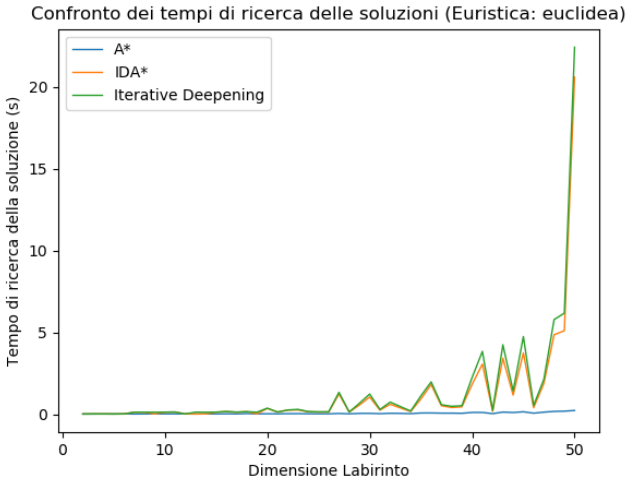
Di seguito verranno riportati i risultati e le considerazioni derivanti dall'esecuzione dei tre algoritmi su una serie di labirinti generati automaticamente. In particolare verranno forniti i grafici relativi alla lunghezza delle soluzione ed al tempo di ricerca.

Considerando azioni con costo unitario e utilizzando le euristiche di Manhattan e Max Distance si ottengono i seguenti grafici:



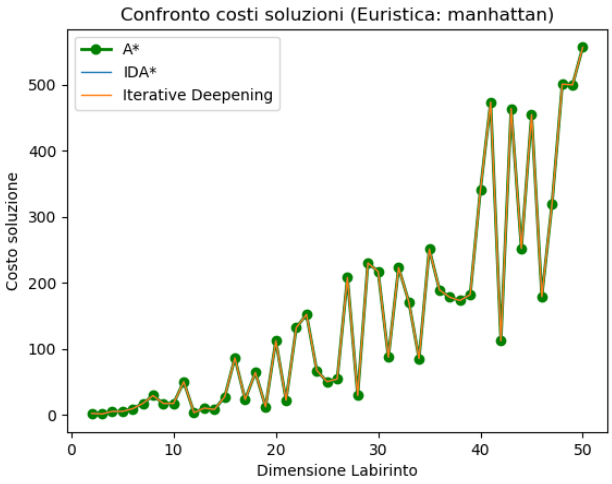


Osservando i grafici si può osservare che A*, come atteso, risulta sempre più performante rispetto a IDA* e Iterative Deepening. IDA* nel complesso è più veloce di Iterative Deepenig.
E' inoltre possibile osservare la qualità delle euristiche, ovvero quanto più si avvicinano all'euristica perfetta (costo reale per raggiungere l'obiettivo).



La distanza di Manhattan riduce considerevolmente i costi di ricerca.
Dall'evidenza sperimentale si osserva che la distanza euclidea comporta tempi di ricerca leggermente superiori, mentre la max distanza (essendo sempre minore di entrambe le euristiche) è la meno efficace.

Il costo delle soluzioni trovate risulta coincidere per tutte le strategie di ricerca, poiché con azioni di costo unitario tutti gli algoritmi trovano la soluzione ottima.



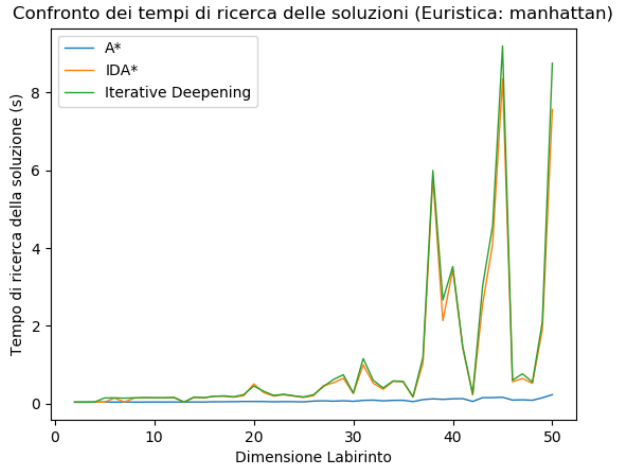
5. Analisi delle prestazioni con costi non unitari

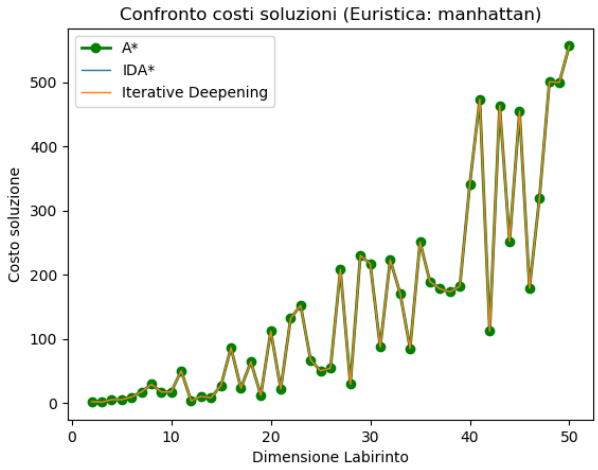
Di seguito vengono analizzate le strategie di ricerca ipotizzando una situazione in cui le azioni hanno costo non unitario, ad esempio il costo delle azione *est* e *ovest* potrebbe risultare maggiore rispetto alle azioni *nord* e *sud* a causa di un particolare servo sterzo.

I fatti che rappresentano il costo delle azioni risulteranno quindi:

costo(est,3). costo(ovest,3). costo(sud,1). costo(nord,1).

Si riporta di seguito il grafico relativo alle prestazioni degli algoritmi e al costo delle soluzioni trovate:





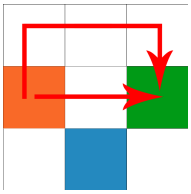
Dai grafici si può osservare che le osservazioni precedenti rimangono valide e che tutte le strategie di ricerca trovano la soluzione ottima. La discussione sulla completezza e ottimalità degli algoritmi viene rimandata alla sezione successiva.

6. Completezza e ottimalità delle strategie di ricerca

Nell'implementazione utilizzata gli algoritmi A* e IDA* sono sia completi che ottimi, se l'euristica è consistente.
Iterative Deepening è completa, risulta inoltre ottima quando il costo del percorso è una funzione non-decrescente (monotona) della profondità del nodo.

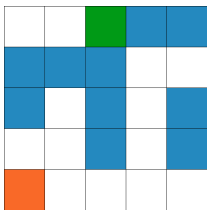
Per accorgersi delle differenze delle soluzioni trovate possiamo analizzare la seguente particolare situazione:
Le azioni nord e sud hanno costo negativo, in qualche modo ricaricano il robot (esempio pannelli solari), mentre le azioni est e ovest hanno costo 2.

costo(est,2). costo(ovest,2).
costo(sud,-1). costo(nord,-1).



Formulando una funzione euristica ammissibile per il problema riportato in figura:
H(S) = 2*|x1 - x2| - |y1 - y2|
gli algoritmi A* e IDA* individueranno la soluzione ottima *nord - est - est - sud* di costo 2, mentre l'algoritmo Iterative deepening troverà la soluzione *est - est* di costo 4.

6.1. Soluzione non appartenente allo spazio degli stati



Come già illustrato nei paragrafi precedenti, tutte le implementazioni delle strategie di ricerca gestiscono il caso in cui la soluzione non appartenga allo spazio degli stati.
A* per definizione termina quando lo spazio degli stati è stato esplorato completamente.
IDA* e Iterative Deepening terminano quando le asserzioni individuano situazioni in cui la ricerca non può più esplorare nuovi percorsi per la ricerca della soluzione e ripeterebbe la computazione precedente. Iterative Deepening ha complessità spaziale $O(b^d)$, dove b è il branching factor e d è la profondità della soluzione più vicina alla radice, nel caso peggiore quindi d corrisponde al path più lungo. IDA* rivisiterà meno volte i nodi sui cammini a causa della funzione euristica, ma il caso pessimo risulta comunque computazionalmente proibitivo.