Intelligenza Artificiale e Laboratorio

Anno accademico 2019/2020

Breve descrizione – dire che ci limitiamo ad accettare il primo risultato restituito come output in quanto ottimo e si eliminano eventuali percorsi alternativi tramite l’utilizzo del cut

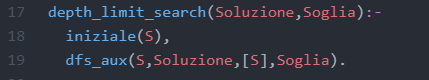
**Strategie non informate**

**Iterative deepening**

L’implementazione di questo algoritmo sfrutta l’utilizzo della ricerca in profondità limitata dove la soglia viene incrementata ad ogni iterazione in cui non raggiunge un risultato finale.

La ricerca in profondità limitata è divisa in due parti:

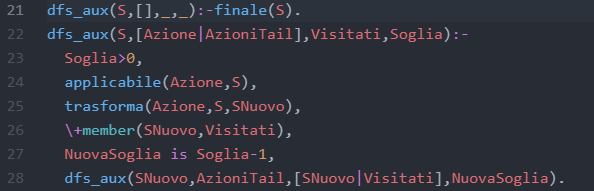
* Nella prima parte andiamo ad inizializzare con la variabile S la posizione iniziale da cui si partirà all’interno del labirinto e chiamiamo la seconda parte passandogli:
  + S: posizione iniziale
  + Soluzione: la variabile in cui ci aspettiamo di ricevere il risultato della ricerca
  + [S]: una lista contenente i nodi che abbiamo visitato, essendo all’inizio conterrà solo quello iniziale
  + Soglia: l’attuale valore di soglia per questa iterazione



* Nella seconda parte avverrà la ricerca ricorsiva in profondità dove la riga 21 rappresenta la condizione di successo: se si verifica finale(S), la nostra Soluzione inizia a ricostruirsi a ritroso partendo dalla lista vuota.

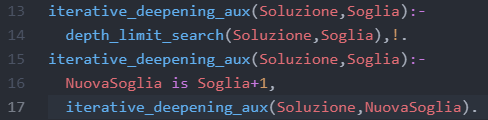
Altrimenti finché la Soglia è positiva continuiamo la discesa ricorsivamente prendendo la prima Azione applicabile nello stato S, costruendo il nuovo nodo ottenuto con lo spostamento Azione e verificando che non si tratti di un nodo già visitato.

La soluzione sarà costruita tramite backtracking aggiungendo in testa alla lista AzioniTail, l’Azione che è stata eseguita in questo step.



Ora che abbiamo visto la ricerca in profondità possiamo passare alla parte iterativa dell’algoritmo, ovvero dove incrementiamo la soglia ad ogni iterazione.

Per ottenere questo risultato semplicemente verifichiamo se è risolvibile con l’attuale soglia e in caso contrario quest’ultima viene incrementata di 1 e si riesegue la ricerca iniziale con il nuovo valore

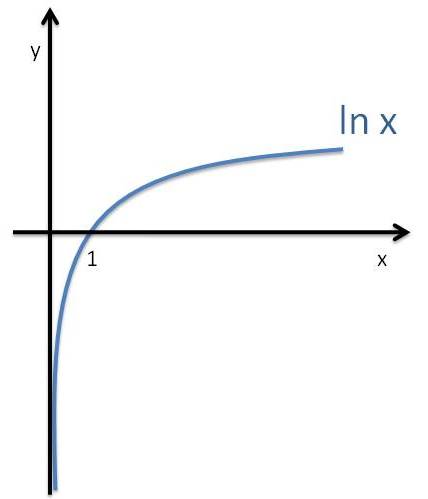


Tuttavia questo tipo di approccio risulta infinito nel momento in cui si presenta un caso in cui non è possibile raggiungere un risultato finale, per questo scopo abbiamo ricercato delle condizioni di uscita, ovvero un Upper bound che raggiunto il quale decidiamo che una soluzione non è più ottenibile in tempi ragionevoli.

La nostra idea è stata di giocare con le dimensioni dei labirinti in maniera tale da avere una condizione di uscita dinamica in base al tipo di problema.

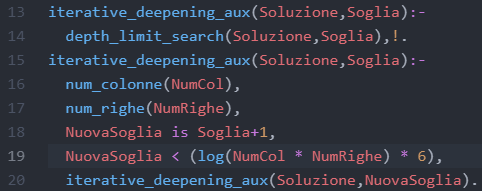
La prima idea è stata di semplicemente moltiplicare il numero di colonne per il numero di righe per assicurarci di non visitare più di quante caselle effettivamente esistano (ignorando la possibilità che alcune siano occupate) ma questa soluzione si è dimostrata insoddisfacente in quanto questo valore di soglia cresce proporzionalmente alle dimensioni del labirinto.

Abbiamo pensato di poter sfruttare la funzione del logaritmo naturale, come mostrato in figura NNN questo ci permette di avere una crescita controllata. Infatti all’aumentare delle dimensioni del labirinto la crescita rallenta permettendoci di ottenere un valore che per essere raggiunto non richiede tempi troppo elevati.



Il risultato della funzione logaritmica è poi moltiplicato per una costante che è stata calcolata sulla sperimentazione. Questa costante ci permette di avere un valore di UpperBound sufficientemente grande per labirinti piccoli ma abbastanza piccolo per labirinti troppo grandi che richiederebbero una notevole quantità in termini di tempo.

Applicata questa modifica il codice rappresentato in figura NNN diventa come segue:



Nota: il cut a riga 14 nella figura precedente elimina eventuali percorsi alternativi di costo pari o maggiore

Per concludere questo algoritmo può essere eseguito tramite la chiamata iterative\_deepening(Soluzione).

Come possiamo vedere in figura NNN questa chiamata va a iniziare la ricerca a partire dalla Soglia 1.



**Strategie informate**

**Algoritmo IDA\***

Come l’iterative deepening questo algoritmo viene ripetuto più volte con una soglia più grande. Tuttavia questa soglia non è più incrementatasoglia di un valore come nel caso precedente ma è calcolata cercando la minima funzione di valutazione tra i percorsi inesplorabili.

Per la funzione di valutazione abbiamo indicato con G la funzione costo, che ci rappresenta la lunghezza del percorso dallo stato iniziale allo stato attuale e con H la distanza di Manhattan ( o geometria del taxi) come nostra euristica. La nostra funzione di valutazione F sarà ottenuta tramite la somma di queste due. F = G + H

In questa nuova ricerca in profondità distingueremo tre casi particolari:

* Se ci troviamo nel nodo finale:

Siamo giunti alla soluzione del problema, la soluzione sarà la lista di azioni per giungere in questo nodo

* Se ci troviamo in un nodo la cui funzione F è minore della nostra attuale soglia:

Ci troviamo in un nodo da cui possiamo proseguire con la ricerca in profondità perché ancora non abbiamo superato la nostra soglia.

* Se ci troviamo in un nodo la cui funzione F è maggiore della nostra attuale soglia:

Ci troviamo in un nodo da cui non possiamo più proseguire in profondità ma dobbiamo controllare se la funzione di valutazione di questo nodo può essere usata come nuova soglia per la prossima iterazione

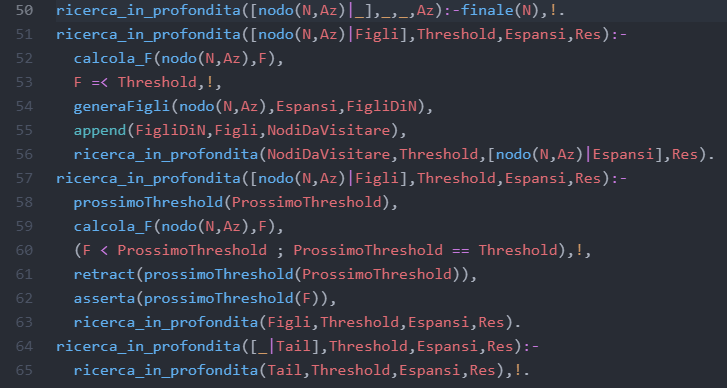
Per comodità abbiamo rappresentato i nodi come una clausola nodo(P, Az) dove P rappresenta la posizione del nodo e Az è una lista di azioni eseguite dalla radice per giungere a questa posizione.

Inoltre utilizziamo i predicati di prolog asserta/retract per salvare dinamicamente i valori di prossimoThreshold che rappresenta la soglia da utilizzare nella prossima iterazione e thresholdCheck che come vedremo sarà utilizzato per una condizione di uscita in una situazione senza un nodo finale raggiungibile.

Partiamo quindi a vedere in dettaglio la ricerca in profondità.



Come vediamo in figura NNNsotto, la riga n°50 rappresenta il primo caso che abbiamo accennato. Quando raggiungiamo un nodo che si unifica con la clausola di finale(N) allora unifichiamo il valore della soluzione (Res) con Az che rappresenta il percorso per giungere in tale nodo

A partire dalla riga 51 inizia il secondo caso particolare, quando dobbiamo continuare la ricerca in profondità. In questo caso generiamo i figli del nodo corrente e li aggiungiamo in testa ai nodi da visitare. Nella chiamata ricorsiva di riga 56 aggiorniamo il valore di Espansi che rappresenta la lista di nodi da cui siamo già passati e che utilizzeremo all’interno di generaFigli per non passare da nodi già visti con un costo di funzione minore rispetto al percorso attuale

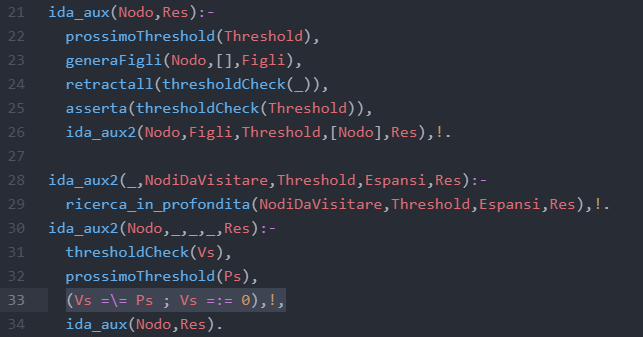
Infine nel terzo caso particolare, quando ci troviamo in un nodo la cui funzione F è maggiore del nostro threshold, dobbiamo aggiornare il prossimo valore di threshold. Questa si aggiorna se rientra in uno dei due casi a riga 60:

* Se la F è minore del ProssimoThreshold allora non è il minimo che stiamo cercando
* Se il ProssimoThreshold è uguale a quello attuale allora abbiamo appena iniziato la nuova iterazione e dobbiamo cambiare la prossima soglia.

Se nessuna delle due condizioni è verificata si esegue semplicemente la chiamata ricorsiva ignorando l’attuale nodo.

Questa ricerca sarà eseguita iterativamente, aggiornando ad ogni ciclo il nuovo valore di Soglia memorizzato in prossimoThreshold. Tuttavia come nel caso dell’algoritmo precedente è necessaria una condizione di uscita qual’ora il programma dovesse essere sottoposto a problemi senza soluzione.

Per questo abbiamo pensato di utilizzare un valore memorizzato dinamicamente in thresholdCheck, inizializzato a 0, contenente ad ogni giro il valore del vecchio Threshold cosicché se per due iterazioni otteniamo lo stesso valore di nuova soglia significa che siamo bloccati e abbiamo visto tutti i percorsi raggiungibili.

****

Come vediamo in figura NNNsopra a partire dalla riga 21 si inizializzano i valori da cui cominciare la ricerca, in particolare si aggiorna la clausola che traccia il valore di thresholdCheck con il valore di soglia che sarà utilizzata nella iterazione che sta per iniziare e in caso essa fallisca in riga 29, si passerà da riga 33 dove troviamo il controllo sopra accennato.

Per concludere questo algoritmo può essere eseguito tramite la chiamata ida\_star (Soluzione).

Come possiamo vedere in figura NNN questa chiamata va a iniziare la ricerca resettando eventuali clausole rimaste sul threshold ( in caso di esecuzioni ripetute) e inizializzandole pari a 0.

