Ricerca informata

Una ricerca informata sfrutta la conoscenza specifica del dominio applicativo per fornire suggerimenti su dove si potrebbe trovare l'obiettivo, più efficientemente di una non informata. I suggerimenti hanno la forma di una funzione euristica.

Ricerca Best-First Greedy

E' una forma di ricerca best-first che espande prima il nodo con il valore più basso $h(n)$, cioè quello che appare più vicino all'obiettivo $(f(n) = h(n))$. Non ottimo e non completo in spazio degli stati infinito.

A*

Una ricerca best-first con $f(n) = g(n) + h(n)$ dove $g(n)$ è il costo del cammino dal nodo iniziale al nodo n e $h(n)$ rappresenta il costo stimato del cammino più breve dal nodo n al nodo obiettivo.
$f(n)$ è il costo stimato del cammino migliore che continua da n fino al goal.

A* con $\epsilon > 0$ (limite inferiore al costo di ogni azione) è completo.
Infatti sia n^* un nodo generico in frontiera nel cammino soluzione, questo (a meno che non si trovi prima la soluzione) verrà espanso.
L'ottimalità dipende da alcune proprietà
Diciamo che un'euristica è ammissibile se non sovrastima mai il costo effettivo per raggiungere un obiettivo. Un'altra proprietà è la consistenza. Un'eurista $h(n)$ è consistente se per ogni nodo n e ogni suo successore n' generato da un'azione a, abbiamo:
$h(n) \leq c(s, a, s') + h(n')$

CONSISTENTE $\implies$ AMMISSIBILE.

Con un'euristica consistente non bisognerà modificare raggiunti o riaggiungere uno stato alla frontiera , perché una volta raggiunto uno stato questo è sicuramente in un cammino ottimo.
Con un'euristica inconsistente potremmo trovarci con più cammini che raggiungono lo stesso stato, e se ogni cammino nuovo ha costo inferiore al precedente finiremmo con avere più nodi corrispondenti allo stesso stato in frontiera, con un aggravio nei costi temporali.
Con un'euristica inammissibile, A* potrebbe essere ottimo in due casi:

- Se vi è anche un solo cammino ottimo rispetto al costo lungo cui $h(n)$ è ammissibile per tutti i nodi n nel cammino.

- Se la soluzione ottima ha costo C^* e la seconda migliore C_2 e se $h(n)$ sovrastima alcuni costi ma mai più di $C_2 - C^*$, allora A* restituisce sempre il cammino ottimo.

E' chiaro che quando si estende un cammino, i costi $g(n)$ sono monotoni: il costo di un cammino incrementa sempre mentre lo si percorre ($\epsilon > 0$). Quando si estende un cammino da n a n', il costo da $g(n) + h(n)$ diventa $g(n) + c(n, a, n') + h(n')$. Eliminando il termine $g(n)$ vediamo che il costo del cammino sarà monotono crescente se e solo se $h(n) \leq c(n, a, n') + h(n')$, quindi se l'euristica è consistente.

Se C^* è il costo del cammino ottimo:

- A* espande tutti i nodi che possono essere raggiunti dallo stato iniziale su un cammino in cui per ogni nodo n si ha che $f(n) < C^*$. Questi sono **certamente espansi**.
- A* potrebbe espandere alcuni nodi dove $f(n) = C^*$ prima di arrivare al nodo obiettivo.
- A* non espande alcun nodo $f(n) > C^*$.

A* è **ottimamente efficiente** (altri algoritmi che usano la stessa euristica di A*, espandono gli stessi nodi, espansi da A*).
A* è efficiente perché esegue la **potatura** dell'albero di ricerca dei nodi non necessari per trovare una soluzione ottima, ovvero scarta alcune possibilirà senza doverle nemmeno esaminare.
In più e completo e ottimo rispetto al costo. Ma per molti problemi il numero di nodi espansi può aumentare esponenzialmente con la lunghezza della soluzione. La memoria usata è $O(b^{d+1})$.

Beam Search

Mantiene solo i k nodi con i migliori costi f scartando ogni altro nodo espanso, ciò rende la ricerca subottima e incompleta.

IDA*

Nel ID la soglia è la profondità, che aumenta di 1 ad ogni iterazione. Nel IDA* la soglia è il costo $f(g + h)$ a ogni iterazione il valore soglia è il più piccolo costo f di qualsiasi nodo che abbia superato la soglia nella precedente iterazione. L'algoritmo è completo e ottimale se:

- le azioni hanno costo costante k e soglia viene incrementato di k.
- le azioni hanno costo variabile e l'incremeneto di soglia è $\leq \epsilon$ (limite inferiore costo azioni).
- la nuova soglia = min valore f nodi generati ed esclusi dall'iterazione precedente.

Memoria usata $O(bd)$.

Ricerca Best-First ricorsiva

l'algoritmo sembra la DF ricorsiva solo che invece di continuare a seguire il cammino corrente, utilizza la variabile soglia per tener traccia del valore $f(n)$ del miglior cammino alternativo che parte da uno qualsiasi degli antenati del nodo corrente. Se il nodo corrente supera questo limite la ricorsione torna indietro al cammino alternativo. Durante il ritorno viene sostituito il valore f di ogni nodo lungo il cammino con un valore di **backup**, il miglior valore f dei suoi nodi figli.
In questo modo RBFS ricorda il valore f della foglia migliore nel sottoalbero abbandonato e può quindi decidere in seguito di riespanderlo.

SMA*

Il problema dei due algoritmi è che usano troppa poca memoria.
SMA* procede esattamente come A* finché la memoria è piena. A questo punto se ne deve aggiungere un altro di nodo nell'albero scarta sempre il nodo foglia peggiore (quello con costo $f(n)$ più alto).
Memorizza nel nodo padre il valore del nodo dimenticato. Con questa informazione viene rigenerato il sottoalbero dimenticato solo quando tutti gli altri cammini premettono di comportarsi peggio.
A parità di f si sceglie il nodo migliore più recente e si dimentica il nodo peggiore più vecchio.
Completo se c'è una soluzione raggiungibile ovvero se $d <$ dimensione memoria.
La strategia è ottima se c'è una soluzione ottima altrimenti viene restitutita quella migliore.
Le limitazioni di memoria possono rendere un problema intrattabile dal punto di vista temporale.

Funzioni euristiche

Distanza di manatthan: la somma delle distanze tutti i tasselli dalla loro posizione corrente a quella nell'obiettivo, considerando movimenti orizzontali o verticali (h_2).

Il numero di tasselli fuori posto (h_1).

Un modo di caratterizzare la qualità di un'euristica è il fattore di ramificazione effettivo b^*. Se il numero totale di nodi generati da A* per un problema è N e la profondità è d,allora b^* è il fattore di ramificazione che un albero di profondità d dovrebbe avere per contenere $N + 1$ nodi.

$N + 1 = 1 + b^* + (b^*)^2 \grave{e} + ... + (b^*)^d$

Il fattore può cambiare da un'istanza all'altra ma solitamente ha un valore abbastanza costante su tutte le istanze di problemi non banali.

Un euristica ben progettata dovrebbe avere valore b^* vicino a 1.

Molte prove dimostrano che $h2$ è migliore di h_1 sul problema dei tasselli.

Dalle definizione delle due euristiche è facile vedere che per ogni nodo $n,\ h_2 \geq h_1$, diciamo in questo caso che h_2 domina h_1.

Teorema

A* con euristica h_2 non espanderà mai più nodi di A* che usa come euristica h_1.

Dim

$f(n) < C^*$ allora n sarà sicuramente espanso. Ovvero sarà sicuramente espanso ogni nodo n con $h(n) < C^* - g(n)$, quando h è consistente. Ma dato che h_2 è grande almeno quanto h_1 , ogni nodo espanso da A* con h_2 lo sarà anche con quella che usa h_1 e anche con h_1 potrà espandere ulteriori nodi. □

Questo significa che ogni nodo $f(n) < C^*$ verrà sicuramente espanso quindi siccome $h_1 \leq h_2$ ci con euristica h_2 A* espanderà meno nodi.

Se ne deduce che generalmente una funzione euristica con valori più alti è meglio a patto che sia consistente e non troppo onerosa da calcolare.

Rilassare i problemi

Per generare euristiche si possono rilassare i problemi. Un problema "rilassato" è un problema con meno restrizioni sulle azioni possibili.

Il costo di una soluzione ottima di un problema rilassato è un'euristica ammissibile per il problema originale. Siccome nel problema rilassato è il costo esatto nel problema originale, l'euristica derivata sarà sicuramente consistente.
Se per un problema abbiamo una collezione di euristiche ammissibili ma nessuna domina le altre $(h_1, h_2, ..., h_m)$, quale dovrebbero scegliere?

$h(n) = max\ \{\ h_1(n), h_2(n), ..., h_m(n)\ \}$

Questa euristica composita sceglie la funzione più accurata per ogni nodo. Dato che tutte le euristiche $h_1, h_2, ..., h_m$ sono ammissibili, lo è anche h. L'unico svantaggio è che $h(n)$ richiede più tempo di calcolo (si può scegliere a caso o predire la migliore).

Soluzioni di sottoproblemi

E' anche possibile derivare euristiche ammissibili dal costo della soluzione di un sottoproblema.

Il costo della soluzione ottima di un sottoproblema (considerare, nel problema degli 8 tasselli, solo i tasselli 1,2,3,4) è un limite inferiore al costo del problema completo.

L'idea alla base dei **database di pattern** è di memorizzare i costi esatti delle soluzioni di ogni possibile istanza.
E' possibile ora calcolare un'euristica ammissibile per ogni stato incontrato nella ricerca, semplicemente estraendo dal database la corrispondente configurazione del sottoproblema.
Potremmo usare anche altri tasselli (5,6,7,8), ogniuno di esse fornisce un'euristica ammissibile e avremmo potuto prendere il massimo.
Si potrebbe sommare l'euristica ottenuta dal DBP da (1,2,3,4) e (5,6,7,8), dato che i suoi sottoproblemi non sembrano sovrapporsi? No, l'euristica non sarebbe più ammissibile, perché le soluzioni ai due sottoproblemi potrebbero condividere delle mosse.
Potremmo considerare non già il costo totale della soluzione del sottoproblema 1,2,3,4 ma solo il numero di mosse che coinvolgono i primi 4 tasselli. Allora la somma dei due costi è ancora un limite inferiore del costo della soluzione dell'intero problema. Questa è l'idea alla base dei **pattern disgiunti**.

Imparare dall'esperienza

Imparare dall'esperienza significa risolvere parecchie istanze del rompicapo. Ogni soluzione ottima per un'istanza del rompicapo fornisce una coppia $< obiettivo, cammino >$ e da queste è possibile usare un algoritmo di apprendimento per costruire h, che possa approssimare il costo di cammino reale per altri stati.
Alcune tecniche di apprendimento funzionano se si forniscono delle **caratteristiche** di uno stato rilevanti per predire il valore dell'euristica di quello stato, invece della descrizione completa.
Naturalmente si possono usare ulteriori caratteristiche e combinarle tra di loro mediante una combinazione lineare:

$$h(n) = c_1 x_1(n) + c_2 x_2(n).$$