Relazione per gli esercizi PROLOG e CLINGO

Andrea Latella, Roberto Pesando, Davide Furno

March 6, 2015

Contents

Part I

Prolog

Chapter 1

Strategie non informate

In questo capitolo verrà trattata l' implementazione delle due strategie non informate viste a lezione ovvero la ricerca in ampiezza e la ricerca in profondità; per ognuna di queste strategie verrà proposta l'implementazione generale e l'applicazione specifica per due domini che sono il dominio dei cammini e il dominio del mondo dei blocchi.

1.1 Ricerca in ampiezza

La ricerca in ampiezza è stata implementata in modo classico, fornendo una regola ric_amp([nodo(S,LISTA_AZ)|_],LISTA_AZ) per il caso base e una ric_amp([nodo(S,LISTA_AZ)|RESTO],SOL) per il caso generico. La prima non fa altro che controllare che lo stato S sia lo stato finale, mentre la seconda invoca le regole espandi(nodo(S,LISTA_AZ),LISTA_SUCC) e append(RESTO,LISTA_SUCC,CODA), per poi richiamare ricorsivamente ric_amp. La regola espandi non fa altro che prendere lo stato S, espanderlo e cercare tutte le azioni applicabili(con findall da quello stato; tramite la regola successori () costruisce la lista dei nuovi stati disponibili a partire dallo stato S, che saranno poi aggiunti agli stati ancora da espandere tramite la regola append(). Infine richiama ricorsivamente la regola per controllare di aver raggiunto lo stato finale e in caso contrario proseguire con l'espansione dei nodi. Dato che vengono espansi tutti i nodi e dato che un nodo può essere espanso più volte questa strategia può non terminare, perchè fa raggiungere il limite di memoria disponibile.

1.1.1 Dominio dei Cammini

Per quanto riguarda il dominio dei cammini, la ricerca in ampiezza risulta sensata solo se la soluzione si trova ad un livello di profondità relativamente basso, altrimenti si incorre nell'esaurimento di memoria. Per quanto riguarda l'esempio specifico fornito dal Prof. Martelli la ricerca della soluzione

ha esito negativo perchè la soluzione si trova ad un livello di profondità che la ricerca in ampiezza non è in grado di raggiungere.

1.1.2 Dominio del Mondo Dei Blocchi

Anche per quanto riguarda il dominio del mondo dei blocchi il risultato non cambia. Infatti anche in questo caso l'esempio fornito dal professore ha esito negativo perchè nuovamente la soluzione risulta troppo in profondità per essere raggiunta.

1.2 Ricerca in profondità

La ricerca in profondità è stato sviluppata in 2 versione; la prima riceve in input una profondità D oltre la quale la ricerca non può andare; la seconda invece parte con una profondità di 1 e man mano che vengono visitati tutti i nodi entro quella profondità senza trovare soluzione, la profondità viene aumentata di un livello. La seconda procedura è denominata Iterative Deepening. L'implementazione della prima versione è molto semplice. Come per la ricerca in ampiezza abbiamo 2 regole, una per il caso base e una per il caso generico. Nuovamente la regola per il caso base non fa altro che verificare che lo stato corrente S sia lo stato finale. La regola del passo generico è stata sviluppata nel seguente modo: se la profondità non è minore di zero applico allo stato corrente S la prima delle azioni applicabili disponibili, dopodiche controllo che il nuovo stato non sia stato già visitato, decremento la profondità di 1 ed infine richiamo la regola ricorsivamente sul nuovo stato. Se la profondità risulta minore di zero, significa che ho superato il limite imposto e di conseguenza non posso scendere oltre nell'espansione. Prolog farà backtracking e procederà con l'applicazione di un'altra delle azioni applicabili per quel nodo e si procede nuovamente come descritto sopra. Se la soluzione è oltre il limite imposto a priori Prolog resituirà false, altrimenti viene restituita la lista delle azioni per raggiungerla. L'implementazione della seconda versione sfrutta la prima. Utilizza le due regole per scendere in profondità nei vari stati, ai quali viene aggiunta una regola per aumentare il limite di un livello, quando la ricerca non trova soluzione all'interno del limite corrente. Il limite che viene fornito a priori è di 1 e ci si ferma solo quando si trova la soluzione.

1.2.1 Dominio dei Cammini

A differenza della ricerca in ampiezza, la ricerca a profondità limitata con approfondimento iterativo (iterative deepening) è in grado di fornire una soluzione ottima e di lunghezza minima per i problemi relativi al dominio dei cammini, dato che tutte le azioni relative a questo dominio hanno costo unitario. Per quanto riguarda l'esempio specifico fornito dal Prof. Martelli

la ricerca in profondità limitata con approfondimento iterativo è in grado di trovare una delle soluzioni di lunghezza minima (se si hanno più soluzioni con la stessa lunghezza possono essere visualizzate tramite il comando; dopo la visualizzazione della prima soluzione)

1.2.2 Dominio del Mondo dei Blocchi

Anche in questo caso la strategia implementata è in grado di fornire la soluzione ottima a lunghezza minima sia per un problema generico del mondo dei blocchi, sia per l'esempio fornito dal professore.

Chapter 2

Strategie informate

In questo capitolo verrà trattata l'implementazione delle strategie informate viste a lezione ovvero la ricerca in ampiezza sui grafi, la ricerca in profondità ad approfondimento iterativo con stima (IDA*) e la ricerca in ampiezza sui grafi con stima (A*); per ognuna di queste strategie verrà proposta l'implementazione generale e l'euristica specifica utilizzata per i due domini proposti che sono il dominio dei cammini e il dominio del mondo dei blocchi.

2.1 Ricerca in profondità ad approfondimento iterativo con stima (IDA*)

L'implementazione dell'algoritmo è stata fatta in modo da sfruttare il lavoro già fatto con la ricerca in profondità con approfondimento iterativo, alla quale andiamo ad aggiungere un euristica specifica per ogni dominio che fornirà un valore di soglia per la nostra ricerca in profondità. La soglia viene inizializzata con un valore di default molto grande, ma man mano che l'esecuzione prosegue, viene sostituita di volta in volta con la f_val minima presa dai nodi che hanno superato la soglia nell'iterazione precedente. Più nel dettaglio l'algoritmo inizia il suo lavoro settando la f_val che verrà utilizzata come limite di ricerca e richiamando la ricerca in profondità già sviluppata precedentemente con alcune piccole modifiche. Nel caso base è stato aggiunto un controllo sulla profondità, ovvero che la nostra f_val deve essere minore o uguale alla profondità; se è cosi si procede con il semplice controllo di S come stato finale, altrimenti si passa alla regola per aggiornare la soglia. Per quanto riguarda il caso generico, abbiamo sempre il controllo della f_val sulla profondità e in più abbiamo il suo ricalcolo tramite l'euristica specifica del dominio. La regola try_prof(F) ha una doppia funzione: oltre ad aggiornare la soglia quando questa risulta maggiore della nostra f_val corrente, ci permette di bloccare la ricerca quando siamo nel caso opposto, ovvero che la f_val risulta maggiore del nostro valore di soglia fissato. Infine quando la ricerca in profondità fallisce abbiamo una regola per riportare al valore di

default la soglia e ricominciare nuovamente la ricerca in profondità.

2.1.1 Dominio dei Cammini

Per quanto riguarda il dominio dei cammini l'euristica utilizzata per generare i valori di f_val è stata la classica distanza di Manhattan. Vengono presi in inpunt le coordinate x e y del nodo corrente e del nodo goal e si esegue questo semplice calcolo:

$$L1(p1, p2) = |x1 - x2| + |y1 - y2|$$

$$F = G + L1$$

G viene passata come parametro nella regola, mentre il valore F che viene calcolato viene asserto come f_val. Per quanto riguarda l'esempio fornito dal Prof. Martelli l'algoritmo precedentemente descritto, sfruttando quest'euristica, è in grado di fornire la soluzione ottima a costo minimo.

2.1.2 Dominio del Mondo dei Blocchi

L'euristica utilizzata per il mondo dei blocchi fornisce il nostro valore di f_val sommando il costo del cammino trovato dal nodo iniziale al nodo corrente, con il numero dei blocchi che si trovano nella posizione della configurazione finale. Questo viene fatto generando la lista dei nodi in posizione corretta tramite ord_subtract(List1,List2,Res). Ord_subtract è una regola nativa di Prolog e non fa altro che sottrarre la List1 dalla List2 due e mettere la lista risultante in Res. Infine tramite la regola calcola_h contiamo gli elementi di Res per trovare il numero dei blocchi in posizione correta che saranno poi sommati a G per avere la nostra f_val. Anche in questo caso l'algoritmo è in grado di fornire la soluzione ottima a costo minimo.

2.2 Ricerca in ampiezza sui grafi con stima (A*)

L'implementazione dell'algoritmo di ricerca astar è stato fatto in modo molto semplice. Sfruttando le regole native di Prolog ord_union() e ord_add_element possiamo costruire delle liste di nodi ordinate secondo il loro f_val() cosi da avere la certezza di analizzare sempre lo stato migliore fra quelli all'interno della lista degli stati da visitare. Vediamo più nel dettaglio l'implementazione. Come al solito la ricerca astar è composta di 2 regole principali, una per il caso base che non fa altro che controllare che lo stato in input S sia lo stato finale e una per caso generico. La regola per il caso generico controlla prima di tutto che lo stato in inpunt non faccia parte della lista dei nodi chiusi e quindi già visitato; in caso affermativo, richiamiamo la ricerca astar sul nodo successivo nella lista dei nodi aperti, mentre in caso negativo lo apriamo con

la regola open_node(). La regola non fa altro che cercare tutte le azioni applicabili dal nodo corrente, inserendole in una lista di azioni che verrà data in pasto alla regola best_node(). La regola best_node() non fa altro che costruire una lista ordinata di figli basata sul loro f_val() in ordine crescente. La costruzione viene fatta grazie a ord_add_element (Set1,Elem,SetRes) il quale prende in input una lista ordinata di elementi (Set1) alla quale aggiunge in modo ordinato il nuovo elemento (Elem) mettendo infine il risultato nella lista ordinata SetRes. ResTerminata la costruzione di questa lista ordinata di nodi figlio, la aggiungiamo alla lista dei nodi aperti tramite la regola ord_union(Set1,Set2,SetRes) la quale non fa altro che unire ordinatamente le due liste mettendo nuovamente il risultato in SetRes. Come ultimo passo richiamiamo la ricerca astar sul primo nodo della lista dei nodi aperti e aggiungiamo il nodo corrente alla lista dei nodi chiusi.

2.2.1 Dominio dei Cammini

Per quanto riguarda il dominio dei cammini è stata utilizzata la stessa euristica descritta per la ricerca ad approfondimento iterativo con stima (IDA*) ovvero la distanza di Manhattan. Anche in questo caso l'algoritmo è in grado di fornire la soluzione ottima a costo minimo sull'esempio fornito dal professore.

2.2.2 Dominio del Mondo dei Blocchi

Anche per il dominio del mondo dei blocchi l'euristica utilizzata è la stessa della ricerca ad approfondimento iterativo con stima (IDA*), riuscendo anche in questo a fornire una soluzione ottima a costo minimo sull'esempio fornito.

Chapter 3

Metropolitana di Londra

Il dominio della metropolitana di Londra risulta leggermente differente dai due domini presi in esame precedentemente, perchè le azioni non hanno un costo unitario. Oltre a dover sviluppare un euristica adatta a calcolare un f_val() per poter eseguire la ricerca, bisogna anche far fronte che non basta più un semplice incremento di uno del nostro g_val dopo ogni azione, ma serve un incremento diverso in base all'azione che stiamo per intraprendere. Si è deciso quindi di dare 2 valori di incremento differenti al g_val: +10 per le azioni di sali/scendi dalla metropolitana e un +5 per le azioni di vai. Con questi pesi le azioni di sali/scendi risultano molto più costose dell'azione di vai, facendo prediligere quindi percorsi più lunghi con pochi cambi di metro a percorsi più brevi ma con cambi più frequenti. Per quanto riguarda l'euristica scelta si è optati per una semplice distanza euclidea, visto che venivano fornite le coordinate delle stazioni. La distanza euclidea viene calcolata come segue:

$$L1(p1, p2) = sqrt((x1 - x2) + (y1 - y2))$$

Il risultato di questa semplice operazione sarà l'f_val associata ad ogni nodo nella ricerca. Infine per quanto riguarda la ricerca vera e propria è stato utilizzato l'algoritmo astar precedentemente sviluppato per il dominio dei cammini e del mondo dei blocchi. Dato che l'algoritmo era stato sviluppato nel mondo più indipendente possibile dal dominio è stato possibile riutilizzarlo anche in questo dominio, aggiungendo ovviamente le regole per l'incremento del g_val e l'euristica specifica.

Part II

Clingo