Relazione Esercizi Prolog e Clingo

Corso di Intelligenza Artificiale e Laboratorio A.A. 2012/2013
Docenti: Prof. Alberto Martelli, Prof. Pietro Torasso

Andrea Pacino – 774734
Giacomo Falcone - 751155

1. Strategie utilizzate

Le strategie implementate in prolog sono in totale 6. Di queste, 4 sono strategie non informate le restanti 2 sono informate.

1.2. Ricerca informata

La prima strategia di ricerca non informata implementata è quella in **profondità**, la quale espande sempre il nodo più profondo nell'albero di ricerca. La ricerca in profondità con backtracking usata dal prolog per la regola di risoluzione permette un risparmio notevole di memoria. Essendo la ricerca in profondità implementata in questo caso per gli alberi è soggetta sovente a loop infiniti che riempiono lo stack e restituiscono l'errore *"out of stack"*.

La strategia di ricerca in **profondità di un grafo** non differisce molto dalla precedente. A differenza della prima questa presenta un controllo che verifica, prima della chiamata ricorsiva sul nuovo nodo, se questo è uguale ad un nodo già visitato in precedenza e quindi se è necessario espanderlo o meno. In questo modo è possibile evitare gli errori di *"out of stack"*.

Per la strategia di ricerca in **ampiezza** abbiamo separato la logica di funzionamento della strategia e le operazioni che si occupano dell'espansione e dell'aggiunta dei nuovi nodi. Tali funzioni sono state modellate in clausole dedicate che prima cercano le possibili azioni applicabili ad uno stato e, successivamente, scorrono la lista delle azioni possibili aggiungendo i nuovi nodi in coda.

La strategia di ricerca ad **iterative deepening** evita loop infiniti e permette la ricerca completa ed ottima per i nostri domini. I requisiti di memoria di cui necessita questa ricerca sono al più uguali a quelli richiesti dalla ricerca in ampiezza, ma solitamente sono inferiori. Questo perché i nodi goal si trovano sulla frontiera e, grazie al backtracking, non si tiene traccia dei nodi generati nelle precedenti iterazioni.

1.2. Ricerca informata

Per diminuire lo spazio di memoria occupato e il tempo di esecuzione della ricerca in ampiezza abbiamo implementato la ricerca con **A***. Abbiamo, dunque, modificato la ricerca in ampiezza per lavorare con gli insiemi ordinati così da poter selezionare più facilmente il nodo da espandere in base al minimo costo stimato F. La modifica principale riguarda la clausola che aggiunge nuovi successori; per questi il nuovo costo stimato F è calcolato mediante una euristica, sommando la distanza dal nodo goal alla distanza che li separa dal nodo radice, ottenendo il costo stimato F per quel nodo. L'euristica ovviamente dipende dal dominio e per quella dei cammini consiste nella distanza Manhattan mentre per quella dei blocchi nel numero degli elementi dello stato del goal non presenti nello stato attuale. Entrambe sono euristiche ammissibili, cioè stimano il costo attuale e la distanza dal nodo goal sempre per difetto, ma quella utilizzata nel mondo dei blocchi non è consistente.

Per diminuire ulteriormente i requisiti di memoria della ricerca con A* abbiamo implementato **IDA***. Le qualità della ricerca ad approfondimento iterativo unite a un'euristica ammissibile permettono di raggiungere una soluzione con costi di memoria ragionevoli e tempi di esecuzione migliori della sola ricerca non informata nei domini da noi considerati.

2. Domini

In questa sezione verranno descritti i domini considerati e gli esempi di dominio utilizzati per testare le varie strategie di ricerca implementate. Per ogni esempio è stato analizzato il fattore del tempo di esecuzione di ogni strategia, confrontandoli. Tali test sono stati effettuati su un unico computer.

2.1. Cammini

Data una mappa rettangolare con presenza di ostacoli, bisogna elencare le azioni per raggiungere la posizione goal da una posizione iniziale. Le possibili azioni applicabili, a meno di non incontrare un ostacolo sono: est, sud, ovest e nord. Importante qui è l'ordine con cui è possibile applicare le azioni poiché questo influenza i tempi di esecuzione a seconda della strategia usata. Di seguito riportiamo gli esempi di dominio:

Primo esempio.

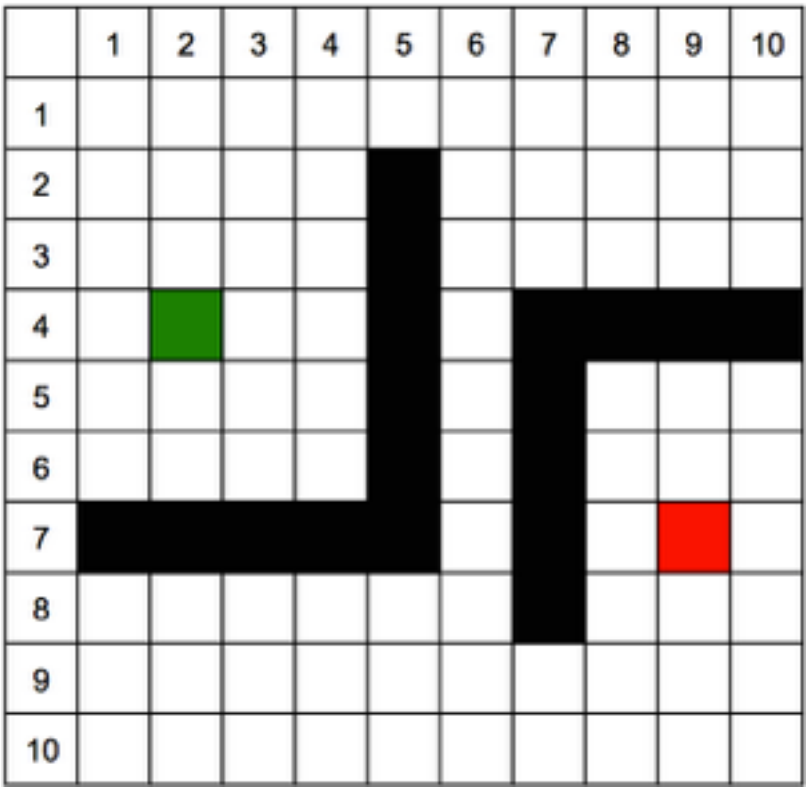

Figura 1: esempio di mappa 10x10. in nero vengono rappresentati gli ostacoli, in verde lo stato iniziale, in rosso lo stato finale

Secondo esempio

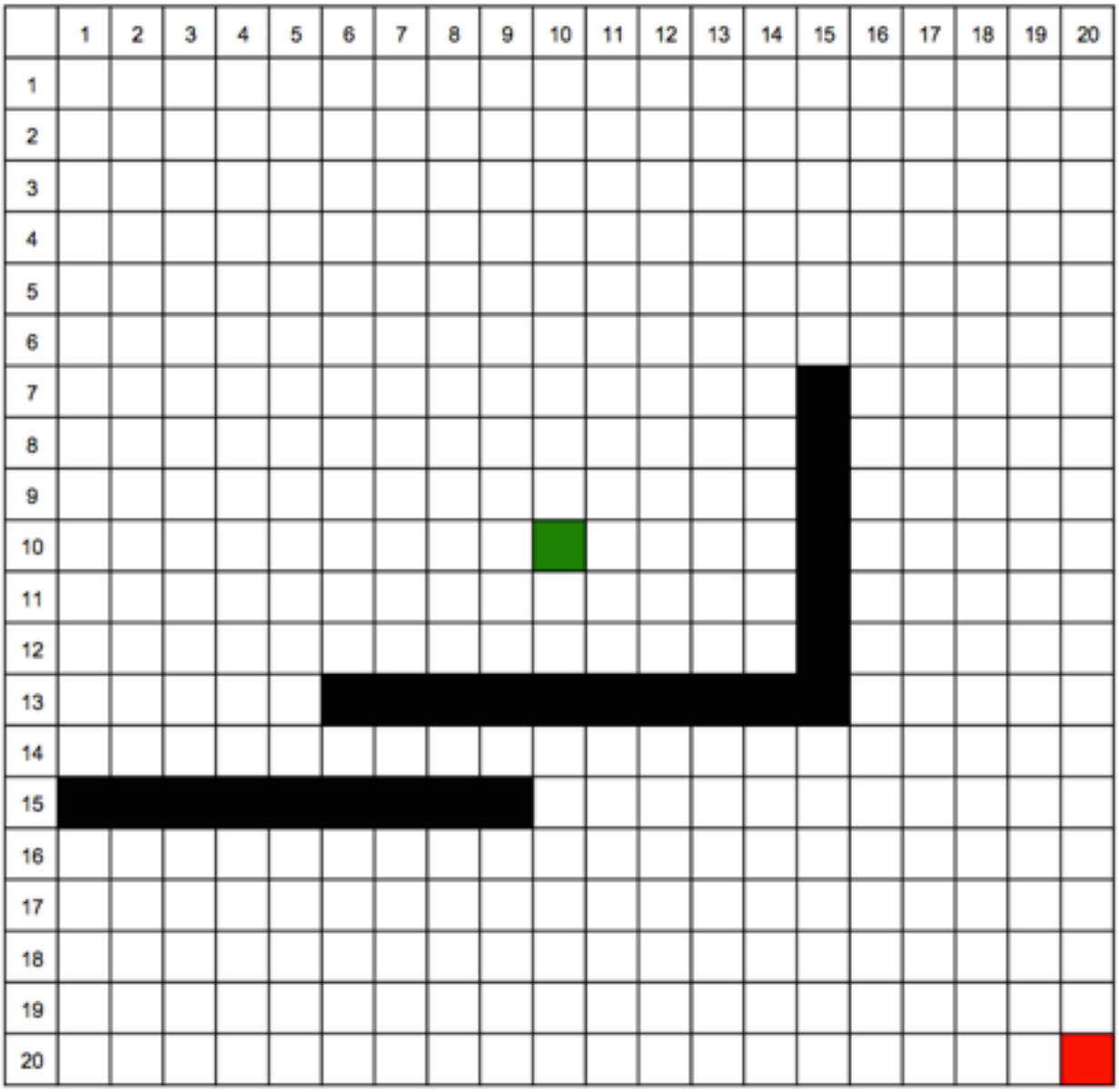

Figura 2: esempio di mappa 20x20. In nero vengono rappresentati gli ostacoli, in verde lo stato iniziale, in rosso lo stato finale

2.1. Blocchi

Dato una tavolo che si assume essere infinito con sopra un certo numero di blocchi e un braccio meccanico, elencare le azioni per raggiungere uno stato goal a partire da uno stato iniziale. Il braccio meccanico può avere in mano un solo blocco per volta e le possibili azioni applicabili sono: prendere un blocco dal tavolo, rilasciare un blocco sul tavolo, prendere un

blocco da sopra un altro blocco e rilasciare un blocco su un altro blocco. Di seguito riportiamo gli esempi di dominio:

Primo esempio.

Figura 3: Esempio di blocchi 1. La parte superiore della figura rappresenta lo stato iniziale, la parte inferiore della figura rappresenta lo stato finale.

Secondo esempio.

Figura 4: Esempio di blocchi 2. La parte superiore della figura rappresenta lo stato iniziale, la parte inferiore della figura rappresenta lo stato finale.

3. Risultati e valutazioni

In questa sezione mostreremo i risultati ottenuti dai test effettuati su entrambi gli esempi di dominio applicandovi tutte le strategie implementate ed evinceremo delle valutazioni conseguenti ad essi.

3.1. Cammini

Di seguito riportiamo i grafici rappresentanti il confronto dei tempi di esecuzione delle strategie di ricerca implementate sugli esempi di dominio per i cammini. I primi due grafici si riferiscono all'esempio di cammino 10x10 celle mentre il secondo grafico si riferisce all'esempio di cammino 20x20 celle.

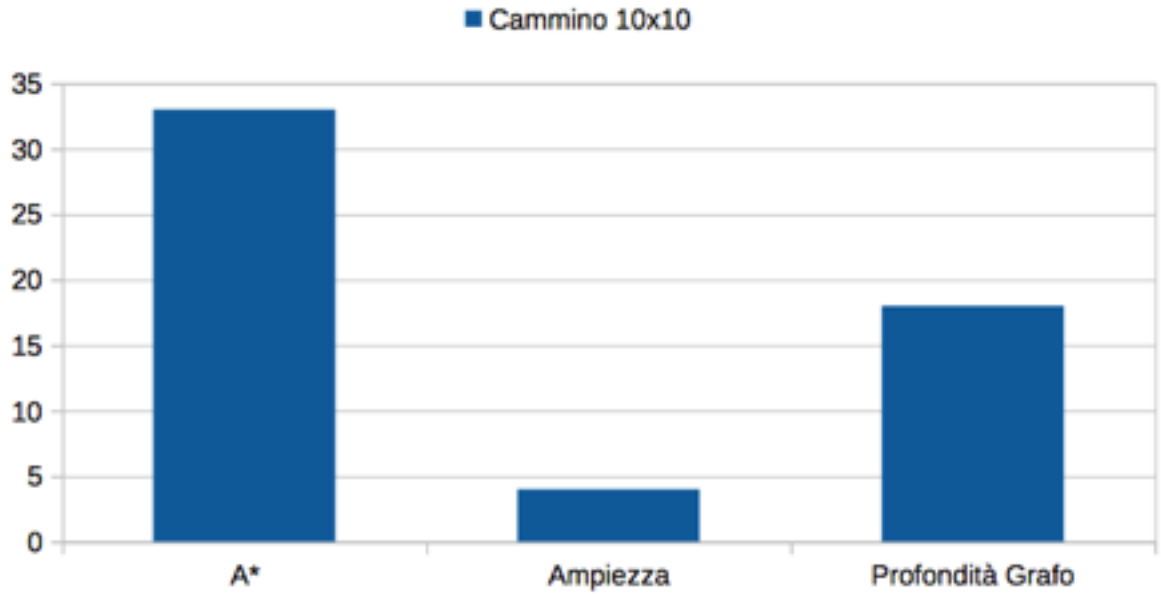

Figura 5: confronto dei tempi di esecuzione delle strategie nell'esempio del dominio dei cammini 10x10 celle

In Figura 5 sono messe a confronto le strategie di ricerca A*, ampiezza e profondità su grafo. Le restanti 3 strategie sono state escluse in quanto la ricerca in profondità classica entra in loop e causa un errore di *"out of stack"* mentre le restanti 2 strategie saranno messe a confronto nella figura successiva. In questo grafico possiamo notare come la ricerca in ampiezza sia la più veloce. Per quanto riguarda A* bisogna considerare che i tempi di esecuzione sono affetti dal calcolo dell'euristica che ne pregiudica il risultato.

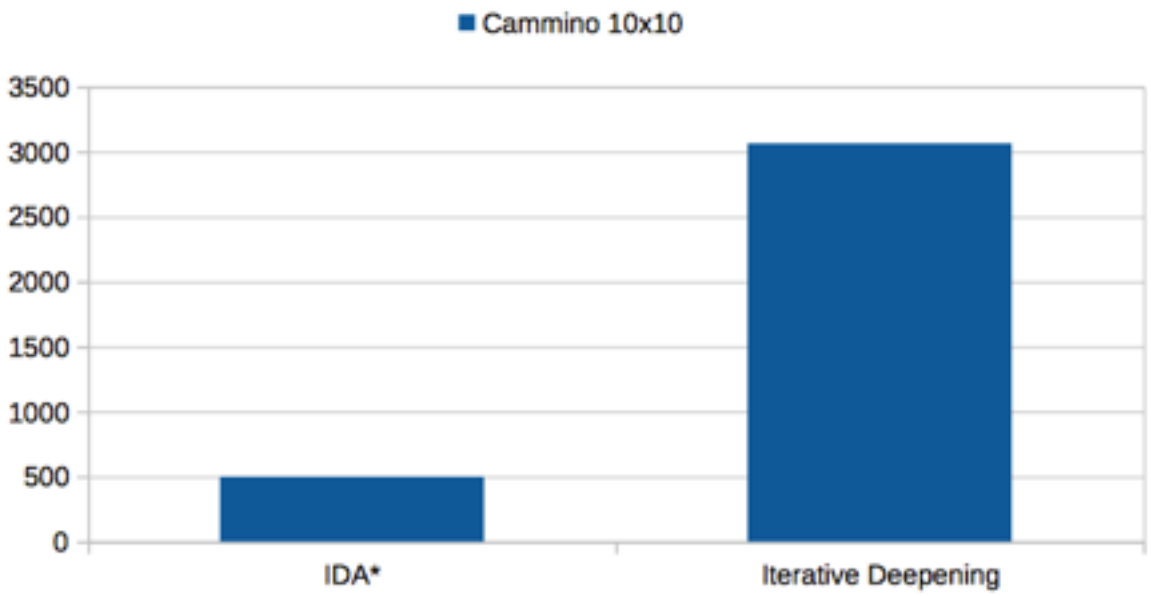

Figura 6: confronto dei tempi di esecuzione delle strategie di ricerca IDA ed iterative deepening nell'esempio del dominio dei cammini 10x10 celle*

In Figura 6 notiamo come i tempi di esecuzione non siano paragonabili con i precedenti. Questo è dovuto alla rigenerazione dei nodi ad ogni iterazione dell'algoritmo fino al soddisfacimento del goal. Possiamo tuttavia notare come la ricerca informata sia più performante della ricerca non informata.

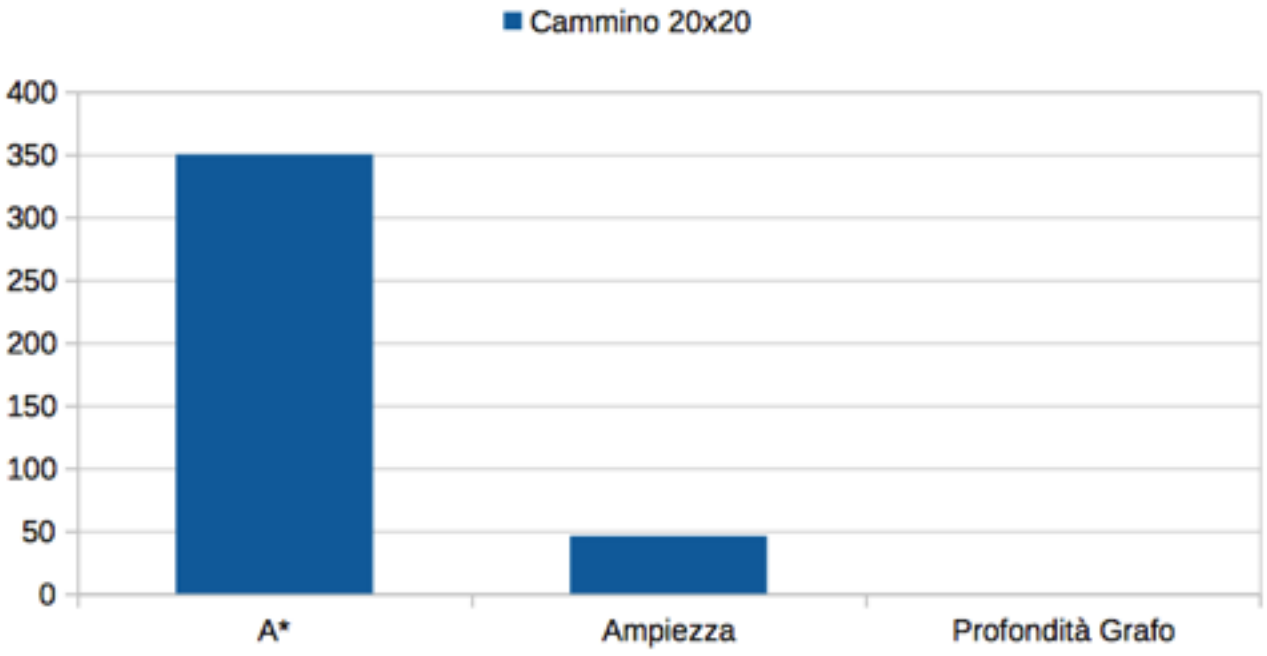

Figura 7: confronto dei tempi di esecuzione delle strategie nell'esempio del dominio dei cammini 20x20 celle

In Figura 7 notiamo come, nell'esempio 20x20, la strategia di ricerca in profondità si comporti in maniera anomala. Questo è dovuto all'implementazione dell'esempio e dall'ordine di esecuzione delle azioni applicabili.

3.1. Cammini

Di seguito riportiamo i grafici rappresentanti il confronto dei tempi di esecuzione delle strategie di ricerca implementate sugli esempi di dominio dei blocchi. I grafici si riferiscono al primo esempio del dominio dei Blocchi.

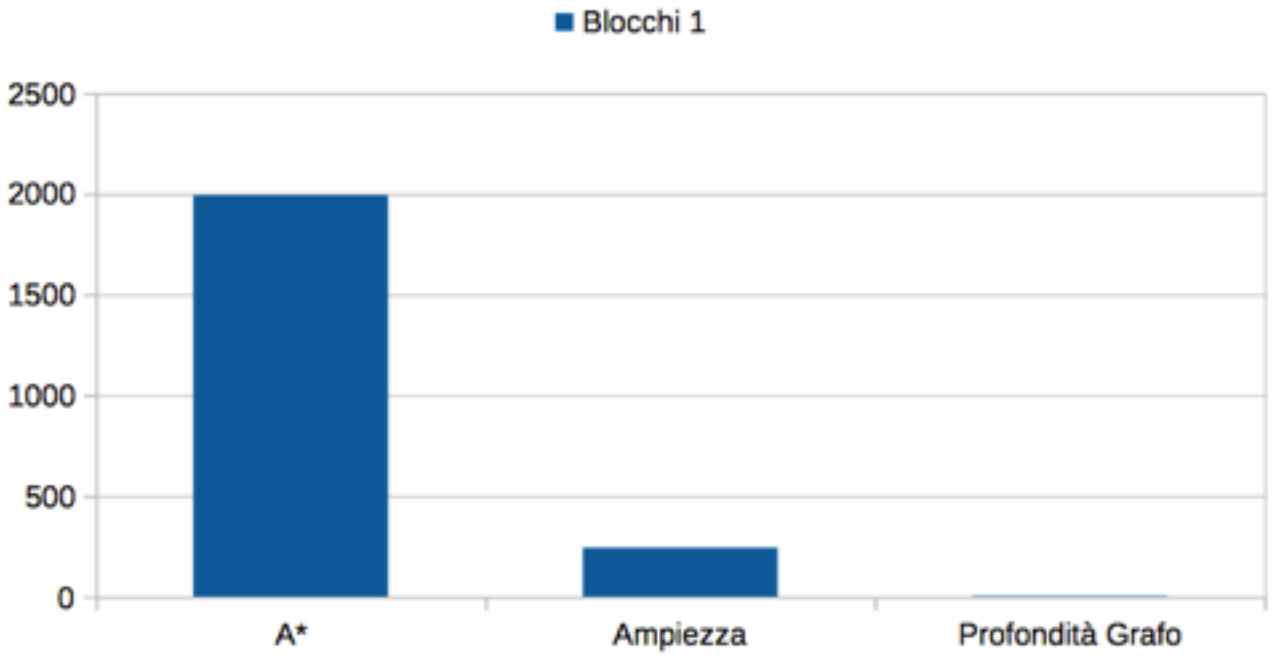

Figura 8: confronto dei tempi di esecuzione delle strategie dell'esempio del dominio dei blocchi

In Figura 8 possiamo notare come, anche in questo caso, la ricerca in profondità per grafi sia riuscita a raggiungere la posizione del goal al primo ramo di ricerca intrapreso. Questo è sempre dovuto all'ordine delle azioni applicabili. Mentre i lunghi tempi di completamento di A* sono probabilmente dovuti all'inconsistenza dell'euristica.

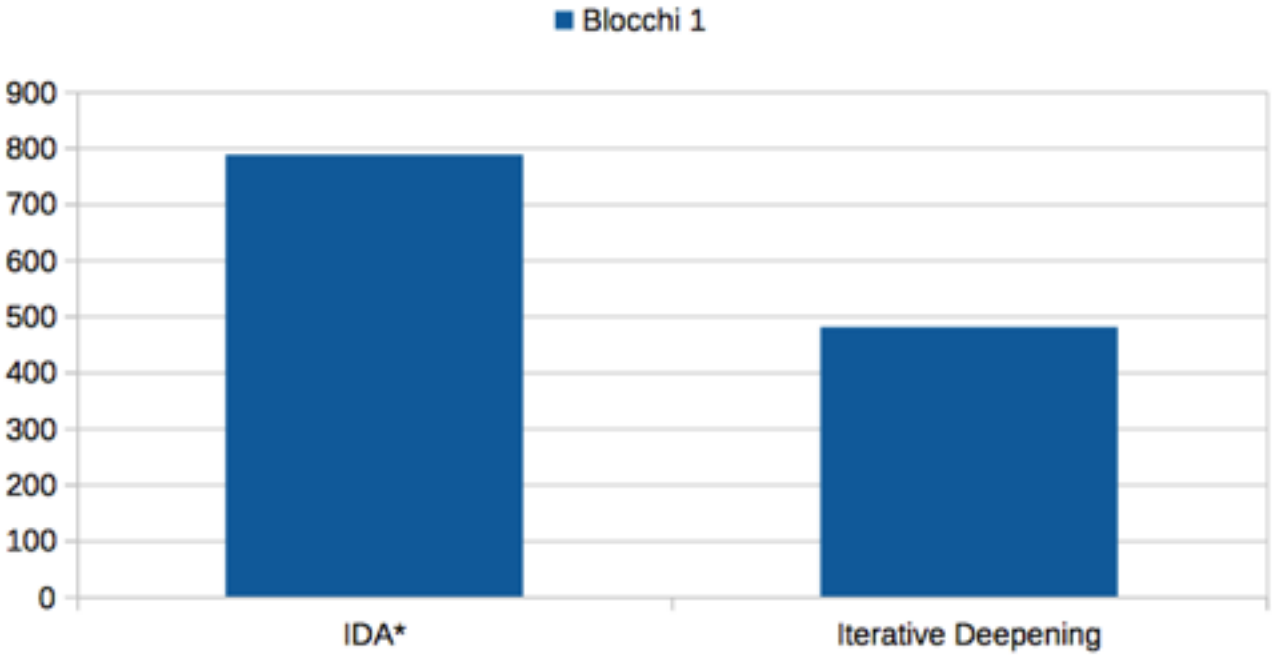

Figura 9: confronto dei tempi di esecuzione delle strategie di ricerca IDA ed Iterative deepening nell'esempio del dominio dei blocchi*

In Figura 9 viene confermata l'ipotesi che l'inconsistenza dell'euristica porti ad un prolungamento dei tempi di esecuzione delle strategie informate.

4. Esercizi in Clingo

Gli esercizi sviluppati utilizzando il paradigma ASP e lo strumento di sviluppo CLINGO sono stati due, il primo riguardante le cinque case, il secondo invece inerente al problema di pianificazione per il dominio del trasporto aereo di merci descritto nel capitolo 10.1 del Russell e Norvig.

4.1. Esercizio 1: CINQUE CASE

L'esercizio è stato svolto definendo inizialmente tutti i fatti del problema, cioè quelli riguardanti nazionalità, professione, animale, bevanda, colore e numero di case. Per i primi 5 tutti i termini possibili per ogni atomo sono stati raggruppati utilizzando il simbolo di pooling ";", per il numero di case abbiamo invece definito un intervallo da 1 a 5 utilizzando il simbolo "..".

Subito dopo abbiamo elencato le regole imposte dai 14 punti del problema. Infine sono stati aggiunti tutti i vincoli: il primo definisce il significato di adiacenza tra case (la distanza tra due case adiacenti è sempre pari ad 1), i secondi sono stati creati sfruttando gli aggregati (viene imposto che per ogni abitazione debbano apparire tutti gli atomi precedentemente definiti esclusivamente una ed una sola volta), ed i terzi sono vincoli di integrità (regole senza testa per permettere di trovare una soluzione corretta).

L'esecuzione di CLINGO è estremamente rapida come è possibile notare dall'output di seguito riportato:
Answer: 1

ownCol(3,rosso) ownNaz(4,inglese) ownNaz(2,spagnolo) ownAnim(2,cane) ownNaz(3,giapponese) ownProf(3,pittore) ownNaz(1,italiano) ownBev(5,te) ownCol(1,verde) ownBev(3,caffe) ownCol(2,bianco) ownProf(2,scultore) ownAnim(4,lumaca) ownCol(4,giallo) ownProf(5,diplomatico) ownNaz(5,norvegese) ownCol(5,blu) ownProf(4,violinista) ownBev(4,succo) ownProf(1,dottore) ownAnim(5,volpe) ownAnim(1,cavallo) ownAnim(3,zebra) ownBev(1,latte) ownBev(2,aranciata)

SATISFIABLE

Models : 1+

Calls : 1

Time : 0.061s (Solving: 0.05s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.031s

4.2. Esercizio 2: PIANIFICAZIONE SU DOMINIO AIR CARGO

L'esercizio è stato creato seguendo la struttura degli esempi proposti presenti su moodle.

Innanzitutto, per mezzo di 3 regole, sono state definite le azioni di *load*, *unload*, *fly* che possono essere compiute nel dominio; a queste è stata aggiunta una regola definita utilizzando un aggregato, per imporre che ci sia al più una azione per stato, anzi più specificatamente che ci sia esattamente una azione per stato. In seconda battuta sono state definite le regole che rappresentano gli effetti delle azioni. Queste regole impongono vincoli di mutex di modo che azioni diverse per ogni stato siano tra loro mutuamente esclusive. A questo punto abbiamo aggiunto le precondizioni, che definiscono quando è possibile applicare una determinata azione ad uno specifico stato. L'ultimo step per completare il pianificatore è riguardato l'aggiunta dei vincoli di persistenza per mezzo di regole non

monotone. Ciò al fine di garantire che un letterale negato (o non negato) rimanga tale anche nello stato successivo se nello stato corrente nessuna azione interviene ad apportare su di esso modifiche. Una volta definito il pianificatore, abbiamo aggiunto tutti i fatti riguardanti il dominio, quindi numero di aerei (intervallo definito tramite ".."), aeroporti e cargo (entrambi raggruppati utilizzando ";"), posizionamento degli aerei e dei cargo nei vari aeroporti, specifica di un particolare goal. Di seguito riportiamo gli output di alcuni degli esperimenti eseguito e le conseguenti valutazioni:

- **Esperimento 1 (10 aerei, 3 cargo, 3 aeroporti, stati 6)**

  Answer: 1

  occurs(load(c1,1),0) occurs(load(c0,10),1) occurs(fly(1,torino,roma),2)
  occurs(fly(10,torino,catania),3) occurs(unload(c1,1),4) occurs(unload(c0,10),5)

  SATISFIABLE

  Models : 1+

  Calls : 1

  Time : 0.059s (Solving: 0.03s 1st Model: 0.01s Unsat: 0.00s)

  CPU Time : 0.031s

- **Esperimento 2 (10 aerei, 3 cargo, 3 aeroporti, stati 31)**

  Answer: 1

  occurs(load(c1,9),3) occurs(load(c1,7),15) occurs(load(c1,6),23) occurs(load(c0,10),26)
  occurs(fly(3,catania,torino),0) occurs(fly(9,roma,torino),1) occurs(fly(3,torino,roma),2)
  occurs(fly(8,catania,torino),4) occurs(fly(7,catania,roma),5) occurs(fly(6,roma,torino),6)
  occurs(fly(5,roma,catania),7) occurs(fly(5,catania,torino),8) occurs(fly(5,torino,catania),
  9) occurs(fly(5,catania,torino),11) occurs(fly(7,roma,torino),12)
  occurs(fly(5,torino,catania),13) occurs(fly(5,catania,torino),14)
  occurs(fly(2,catania,torino),16) occurs(fly(1,torino,catania),17)
  occurs(fly(5,torino,roma),18) occurs(fly(5,roma,torino),19) occurs(fly(2,torino,catania),
  20) occurs(fly(5,torino,roma),21) occurs(fly(1,catania,torino),24)
  occurs(fly(5,roma,torino),25) occurs(fly(10,torino,catania),27) occurs(fly(6,torino,roma),
  28) occurs(unload(c1,9),10) occurs(unload(c1,7),22) occurs(unload(c0,10),29)
  occurs(unload(c1,6),30)

  SATISFIABLE

  Models : 1+

  Calls : 1

  Time : 0.421s (Solving: 0.39s 1st Model: 0.28s Unsat: 0.00s)

  CPU Time : 0.421s

- **Esperimento 3 (10 aerei, 5 cargo, 7 aeroporti, stati 6)**

  Answer: 1

  occurs(load(c1,10),0) occurs(load(c0,1),3) occurs(fly(10,torino,roma),1) occurs(

  fly(1,torino,catania),4) occurs(unload(c1,10),2) occurs(unload(c0,1),5)

  SATISFIABLE

  Models : 1+

  Calls : 1

  Time : 0.069s (Solving: 0.02s 1st Model: 0.00s Unsat: 0.00s)

  CPU Time : 0.062s

  Ciò che possiamo affermare osservando l'output degli esperimenti 1 e 3 è che a prescindere dal numero di aerei, cargo o aeroporti che aggiungiamo al problema, se questi non sono inclusi nel goal, gli answer set ritornati saranno comunque gli stessi e l'ordine di grandezza dei tempi di calcolo sarà praticamente uguale.

  Questo è appunto il caso dell'esperimento 3, in cui sia il numero di cargo sia il numero di aeroporti sono maggiori rispetto a quelli dell'esperimento 1, ma la soluzione resta comunque uguale al netto di possibili variazioni dovute alla scelta di uno o di un altro aereo per effettuare il trasporto.

  È possibile trarre una ulteriore conclusione analizzando l'output degli esperimenti 1 e 2. Il numero di answer set restituiti, infatti, così come il tempo di calcolo, sono fortemente dipendenti dal numero di stati che definiscono il problema. Ciò significa che CLINGO restituirà tutti i goal associati a tutti gli stati in cui questi si presentano. Per cui, a prescindere da quanto facile possa essere il goal o da quanto rapidamente questo venga raggiunto, se il numero di stati è ampio, la computazione impiegherà comunque tanto tempo, poiché CLINGO cercherà tutte le soluzioni in tutti gli stati in cui il goal è soddisfatto.