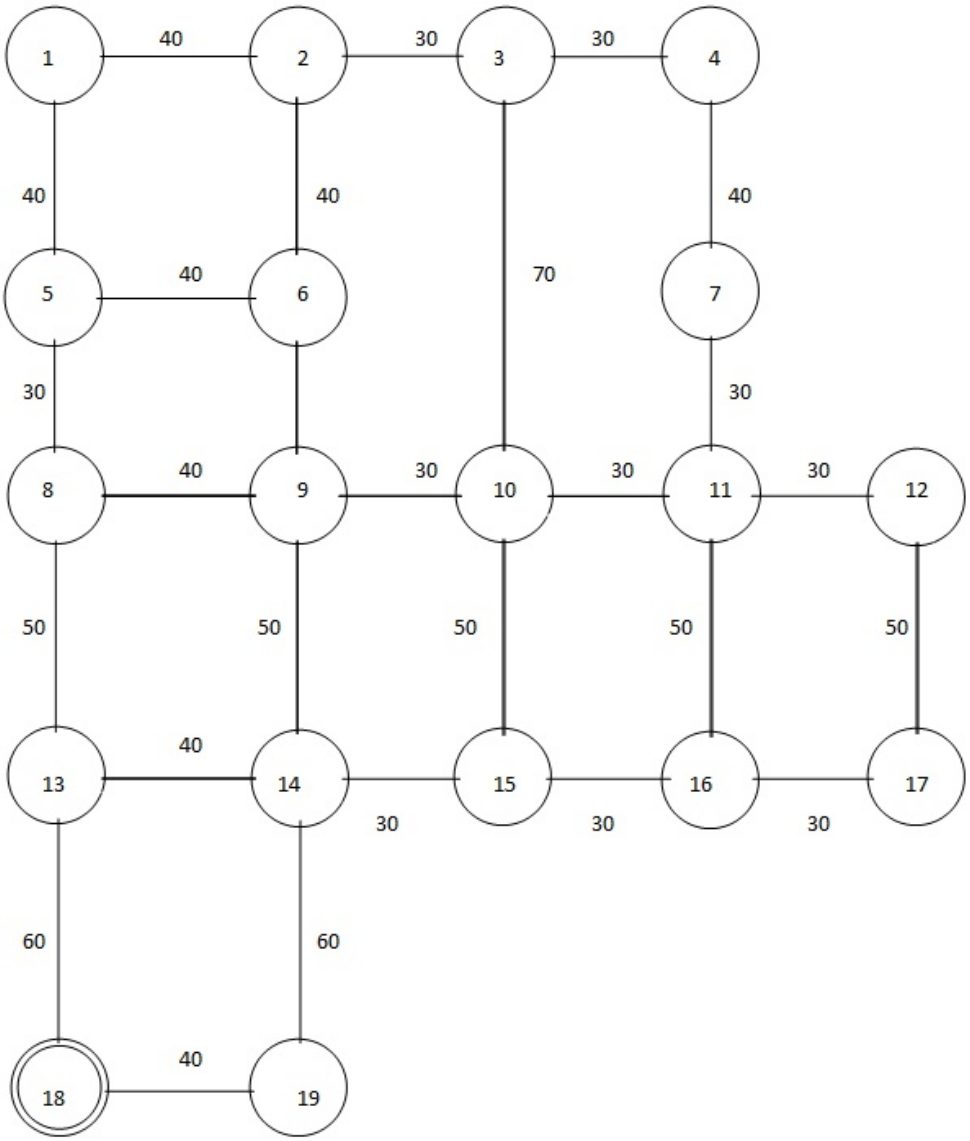
Andrea Pisoni - Matricola: 775696
Luca Palmulli - Matricola: 789156

Anno Accademico 2011-2012

Relazione sul progetto per il corso di Intelligenza Artificiale I

Lo scopo del programma è trovare il percorso più breve per evacuare due gruppi da una città verso un incrocio sicuro.
Il reticolo cittadino è stato rappresentato come un grafo pesato in cui i nodi rappresentano gli incroci, e gli archi le strade che li collegano.
I due gruppi si muovono indipendentemente e il loro viaggio è influenzato da alcuni fattori esterni quali il traffico incontrato, l'inagibilità di alcuni incroci, i mezzi a disposizione e il clima.



Lo scenario è stato modellato come problema di ricerca utilizzando la rappresentazione dello spazio degli stati in cui ogni stato rappresenta la situazione di entrambi i gruppi in un determinato momento.

Predicati e tipi utilizzati

stato → in(situazione,situazione):

il tipo stato è formato, come detto, dalle situazioni dei due gruppi. Questi possono essere essenzialmente in due situazioni differenti : essere arrivati in un incrocio, o viaggiare verso di esso. Nel primo caso la situazione sarà rappresentata da arrivato(X), dove X è l'incrocio in cui risiede attualmente il gruppo, nel secondo caso il predicato sarà formato da viaggio(X,L), dove X è l'incrocio verso il quale il gruppo si sta dirigendo, e L è la quantità mancante al raggiungimento di questo.

mossa(stato,stato):

Il predicato mossa permette il passaggio da uno stato ad un altro. Il predicato è stato svolto esplicitando il prodotto cartesiano delle mosse separate di entrambi i gruppi, e rappresenta la partenza da un incrocio oppure la continuazione di uno spostamento.
Sono stati applicati alcuni cut tra mosse, in modo da non eseguire backtracking quando superfluo, ad esempio se un gruppo fosse già arrivato nell' incrocio sicuro non sarebbe necessario cercare alternative di movimento più generali.

trovato(stato):

Rappresenta la situazione in cui i gruppi sono entrambi arrivati al sicuro.
Lo stato che prende come parametro, per restituire true, avrà una forma in(arrivato(X),arrivato(Y)) dove sia X che Y unificano con l'incrocio sicuro.

vicini(stato, list(stato)):

vicini(X,L) contiene in L tutti gli stati raggiungibili da X tramite una mossa.
è costruito utilizzando il predicato built-in di prolog setof per riempire la lista.

costo(stato,stato,int):

Il costo della mossa tra due stati è dato dalla somma dei costi delle singole mosse di ciascun gruppo, inoltre è soggetto ad aumenti dati da condizioni esterne (es. traffico sugli incroci). Queste condizioni sono riassunte in due coefficienti, uno per ciascun gruppo, che contengono la somma di tutti i costi ulteriori dovuti ai costraint.

connesso(inc,inc,int),agibile(inc,inc,int):

Due incroci sono connessi se esiste una strada che li collega. Data l'assunzione di mondo chiuso di Prolog (CWA), le connessioni vanno esplicitate tutte nella parte di database del programma. L'ultimo int rappresenta la lunghezza della strada che collega gli incroci, e andrà dunque a influenzare mosse e costi.
Una strada che collega due incroci è agibile, se i due incroci sono connessi, e non esiste un pericolo (definito dal predicato di servizio **pericolo**) nell'incrocio di arrivo.

Interfaccia Utente

Per semplificare l'esecuzione dei comandi in fase di test sono stati utilizzati dei wrapper, successivamente corredati di interfaccia utente guidata e adatta alle necessità di un utente meno esperto.

Algoritmi ed euristiche utilizzate

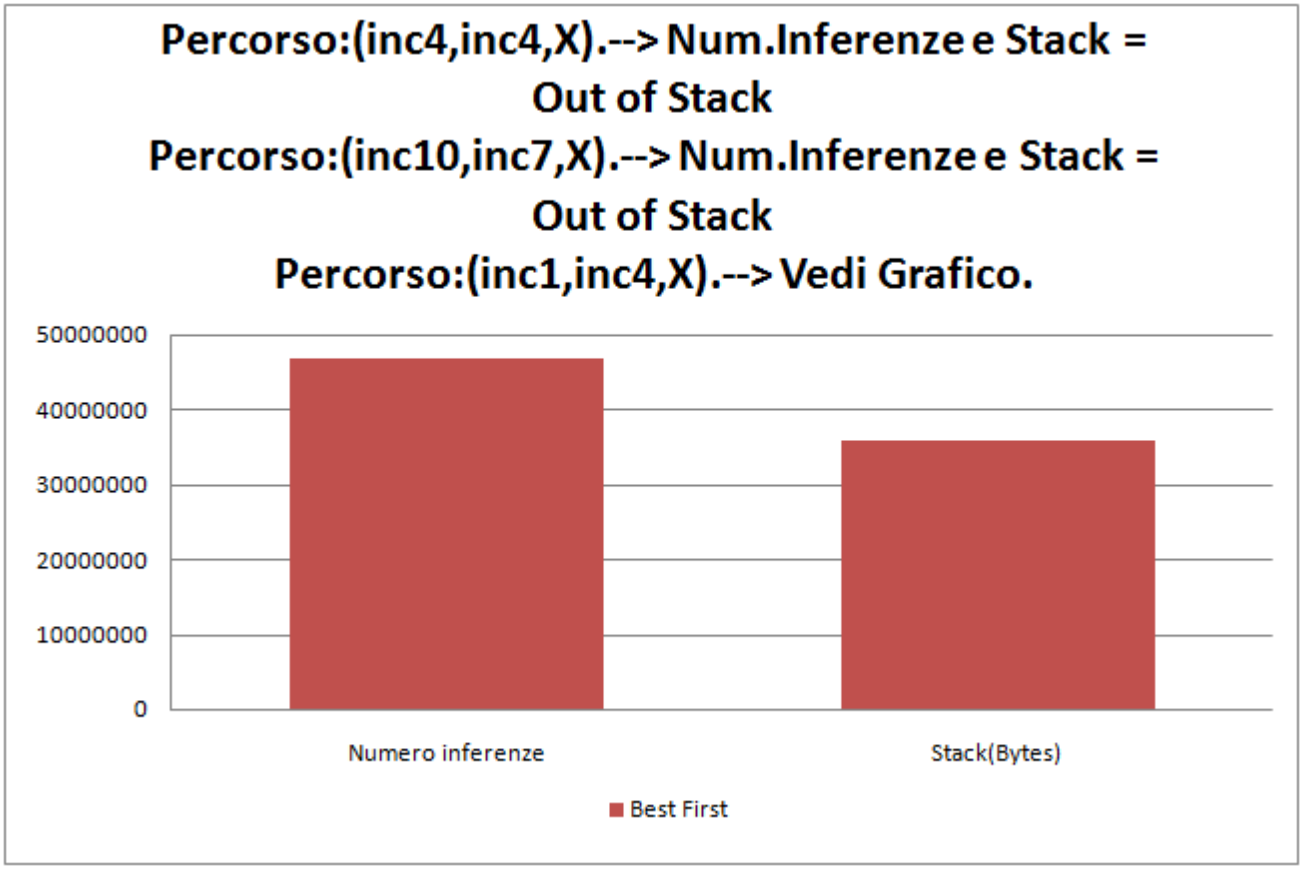
Per testare il programma abbiamo utilizzato principalmente due algoritmi: Best First e A*, per il quale sono state formulate due euristiche differenti: una basata sulla distanza in linea d'aria dal goal, l'altra basata sulla distanza di Manhattan, che in un reticolo che si sviluppa solamente in direzione verticale e orizzontale corrisponde alla somma dei costi per arrivare al sicuro.

Prestazioni

Con l'utilizzo dei sopra citati algoritmi abbiamo riscontrato un netto miglioramento prestazionale preferendo A* insieme alle euristiche. Inoltre l'euristica basata sulla distanza di Manhattan sembra essere migliore di quella in linea d'aria stando a quanto emerge dai dati ricavati da alcune sessioni di Benchmark, mediante le quali abbiamo rilevato il suo minor impatto su Tempo, Memoria utilizzata e Inferenze logiche eseguite.
Proponiamo di seguito alcuni grafici esplicativi derivati dai Benchmark eseguiti.

Primi risultati ottenuti utilizzando Best-First:

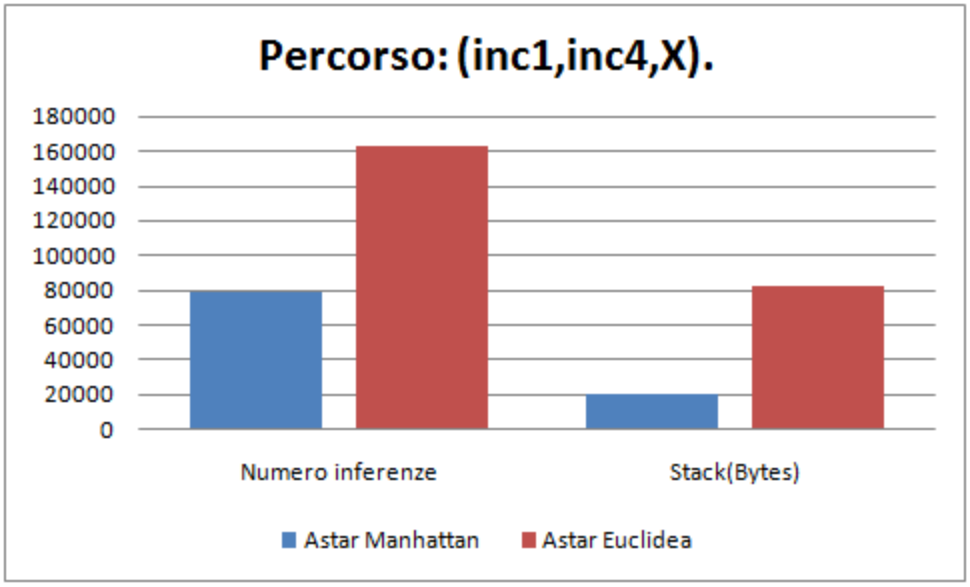
Percorso: (inc1,inc4,X).	Numero inferenze	Stack(Bytes)
Best First	46922722	35760992



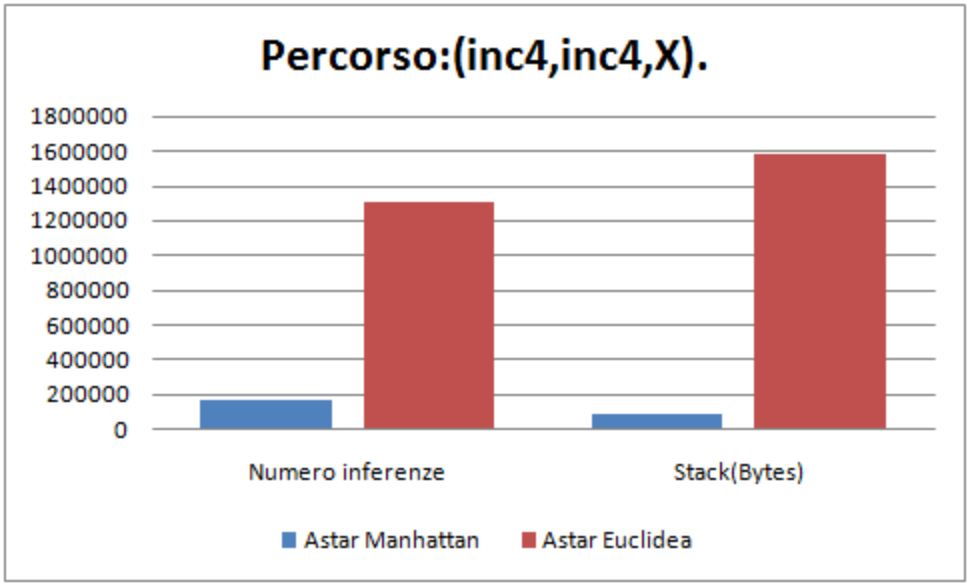
Come emerge dal titolo del grafico che specifica i punti di partenza dei due gruppi, solamente nel caso più “semplice” è stato possibile ottenere un riscontro prestazionale in quanto negli altri test l’esecuzione è terminata con Errore “Out of Stack”.

Ecco i test che mostrano chiaramente vantaggi e svantaggi dello scegliere tra le euristiche che sono state inserite, divise per chiarezza secondo i differenti stati iniziali dei due gruppi.

Percorso: (inc1,inc4,X).	Numero inferenze	Stack(Bytes)
Astar Manhattan	78694	20944
Astar Euclidea	163206	82272



Percorso: (inc4,inc4,X).	Numero inferenze	Stack(Bytes)
Astar Manhattan	170087	91272
Astar Euclidea	1305344	1578216



Percorso: (inc10,inc7,X).	Numero inferenze	Stack(Bytes)
Astar Manhattan	97913	27176
Astar Euclidea	661914	299176