Sliding Puzzle - Disjoint Pattern Databases

Thomas Del Moro

June 1, 2022

1 Introduzione

Il gioco dello sliding puzzle è uno degli esempi più famosi di problemi di ricerca nell'ambito dell'intelligenza artificiale. Può essere utilizzato infatti per testare molti algoritmi di ricerca, tra cui quello che vedremo ovvero A*.

1.1 Sliding Puzzle

Lo sliding puzzle è un insieme di tessere numerate disposte su uno stesso piano. Inizialmente tali tessere sono disposte in ordine casuale e, grazie alla presenza di uno spazio vuoto, esse possono scorrere per cambiare posizione e arrivare a una disposizione vincente, ovvero quella in cui tutte le tessere sono ordinate (detta goal state). Tuttavia non sempre è possibile arrivare al goal state, ci sono infatti delle configurazioni iniziali da cui è totalmente impossibile arrivare alla soluzione finale. Esistono molte varianti di questo gioco, che si differenziano dal numero di tessere presenti. In particolare in questo esperimento vedremo 15-puzzle e 8-puzzle.

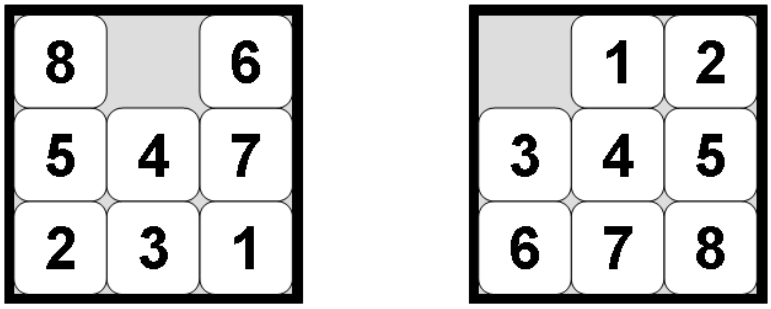

Figure 1: Esempio di configurazione di un 8-puzzle

1.2 Algoritmo di ricerca A*

A* è un algoritmo di ricerca molto simile al Best-First-Search [2], con l'unica differenza che la funzione di valutazione è data da $f(n) = g(n) + h(n)$ dove $g(n)$ è il costo del cammino dal nodo iniziale al nodo n, mentre $h(n)$ è la funzione euristica, cioè la stima del costo minimo del cammino dal nodo n al goal state calcolata secondo un certo criterio.
Per l'esecuzione dell'algoritmo si utilizza una coda con priorità detta frontiera, in cui la priorità più alta è data ai nodi con una funzione $f(n)$ minore.
A ogni iterazione viene quindi estratto il nodo a priorità più alta e vengono generati i suoi vicini, dopodiché si valuta $f(n)$ per ognuno di essi e si inseriscono nella frontiera, finché non si raggiunge lo stato di goal.

2 Euristiche

La funzione euristica rappresenta la stima del costo minimo del cammino da un certo nodo n al goal state e può essere di varie tipologie.
Un'euristica si dice ammissibile se non sovrastima mai il costo di tale cammino.
Inoltre si dice che un'euristica domina l'altra se è sempre maggiore di essa. Un euristica maggiore, se ammissibile, rappresenta generalmente una stima migliore del costo e permette di arrivare al goal state attraversando meno nodi intermedi.
In questo articolo analizzeremo cinque euristiche diverse applicate allo slinding puzzle:

2.1 Manhattan Distance

La distanza Manhattan è l'euristica più semplice che si può realizzare per questo tipo di problema. Si prende una tessera per volta e si considera che tutte le altre posizioni siano vuote. Si conta quindi il numero minimo di passi per spostare ogni tessera nella posizione di goal e alla fine di questa procedura di sommano tutti i valori ottenuti.
L'euristica è sicuramente ammissibile poiché per ogni tessera abbiamo considerato la distanza minima dal goal state; d'altra parte il valore ottenuto sottostima molto il costo reale poiché non si è considerata alcuna interazione tra le tessere.

2.2 Linear Conflicts

La linear conflicts può essere considerata una versione migliorata della Manhattan Distance. In particolare, una volta calcolata la distanza Manhattan, si vanno a controllare le tessere che sono già nella riga o nella colonna di destinazione. Se, ad esempio, due tessere si trovano già sulla riga di destinazione ma le loro due posizioni sono scambiate, possiamo aggiungere due mosse in più all'euristica poiché come minimo dovremo spostare almeno una delle due tessere fuori da tale riga per poterle scambiare. Analogamente per le colonne.
Anche in questo caso non si sovrastima mai il costo ottimo ma si ha un'euristica migliore della precedente.

2.3 Non Additive Databases

I Non Additive Databases rientrano in una tipologia di euristiche diversa dalle precedenti. Utilizzano infatti dei database che vengono costruiti prima dell'esecuzione dell'algoritmo di ricerca, attraverso un BFS all'indietro, a partire dalla configurazione di goal [1].
Andando più nello specifico, si tratta di prendere solo una parte delle tessere e salvare nel database il costo del cammino verso lo stato di goal a partire da tutte le possibili configurazioni iniziali. Un qualsiasi stato è quindi identificato univocamente dalle tessere prese in esame e dalla posizione vuota. Il nome non additive deriva dal fatto che, poiché si contano anche le interazioni con le tessere non coinvolte, non si possono sommare valori dell'euristica presi da gruppi di tessere diversi.

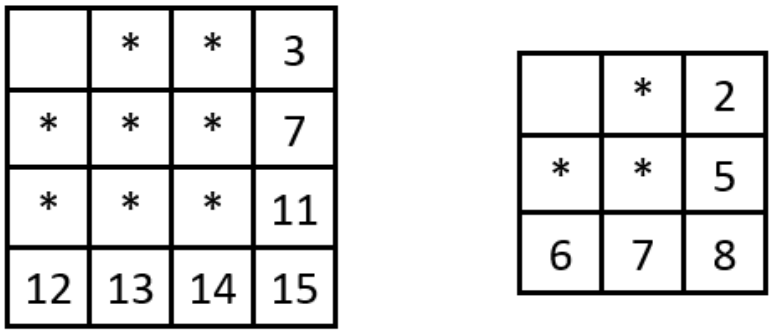

Figure 2: Non Additive Databases su 15-puzzle e 8-puzzle

2.4 Disjoint Pattern Databases

Anche i Disjoint Patter Databases (o disjoint databases) si basano sull'utilizzo di database che vengono costruiti prima dell'esecuzione dell'algoritmo di ricerca, ma a differenza della precedente, questi rappresentano un'euristica sommabile poiché la posizione vuota non viene mai considerata, così come le interazioni tra le tessere di un gruppo e quelle di un altro gruppo. Le tessere vengono prima di tutto divise in due o più gruppi distinti, dopodiché, separatamente per ognuno dei due gruppi, viene calcolato il costo minimo per raggiungere il goal state per ognuna delle possibili configurazioni iniziali. Questo costo viene quindi associato alla propria configurazione e salvato in un database.
Una volta compilati i database si procede all'esecuzione dell'algoritmo, quindi, ad ogni iterazione, il nodo corrente verrà confrontato con i due database, saranno quindi presi i due valori dell'euristica corrispondenti e sommati tra di loro, ottenendo un'euristica ammissibile.

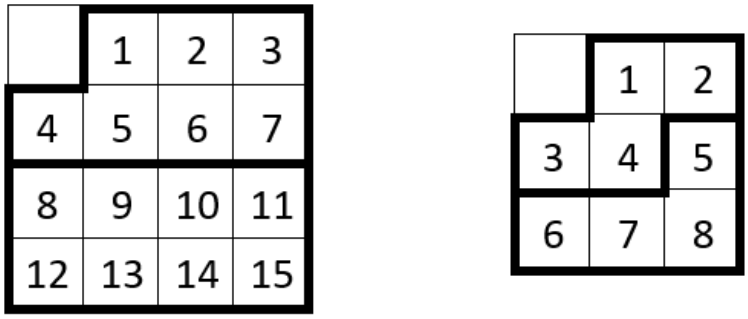

Figure 3: Disjoint Databases su 15-puzzle e su 8-puzzle

2.5 Reflected Pattern Databases

L'ultima euristica che andremo a testare è un ulteriore miglioramento dei disjoint databases. In questo caso, una volta creati i due database descritti prima, se ne costruiscono altri due, prendendo dei gruppi di tessere che sono lo specchio dei precedenti. Durante l'esecuzione quindi, il nodo sarà confrontato sia con i database classici, sia con i reflected database e ovviamente si sceglierà il valore dell'euristica maggiore tra i due ottenuti. Con i reflected database si ottiene un'euristica che in qualsiasi caso è almeno pari alla precedente, ma spesso può essere maggiore.

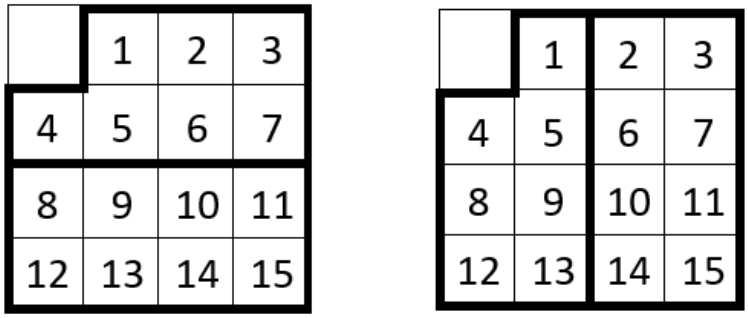

Figure 4: Disjoint Databases e Reflected Databases su 15-puzzle

3 Risultati attesi

In questo esperimento andremo a testare tutte le euristiche descritte sopra. Ovviamente ci aspettiamo che la Manhattan Distance sia l'euristica peggiore sia dal punto di vista del tempo che da quello del numero di nodi generati.
La Linear Conflicts dovrebbe invece mostrare un leggero miglioramento rispetto alla sua precedente, poiché i conflitti sulle righe e sulle colonne alzeranno il valore dell'euristica e saranno quindi generati meno nodi.
I Disjoint Database invece avranno lo svantaggio di dover creare i database prima dell'esecuzione, ma genereranno sicuramente molti meno nodi rispetto alle euristiche precedenti.
I reflected infine dovrebbero risultare ancora più veloci dei disjoint poiché il valore dell'euristica sarà in media più alto. Infine saranno testi i Non Additive Databases, i quali dovrebbero mostrare come i precedenti di essere migliori rispetto alla Manhattan Distance.

4 Esecuzione dei Test

Tutti i test sono stati effettuati su un computer fisso con le seguenti specifiche:

- SO: Windows 10 PRO
- CPU: Intel Core i5-6400 2.70GHz
- GPU: Intel HD Graphics 530 integrata
- RAM: 8GB

I test sulle varie euristiche sono stati eseguiti in sequenza sulle stesse 1000 configurazioni iniziali casuali. Per gestire il caso il cui lo sliding puzzle non è risolvibile e allo stesso tempo garantire la diversità e la casualità delle disposizioni iniziali è stato utilizzato un metodo $shuffle()$ che effettua un numero random di mosse a partire dal goal state.
Purtroppo a causa di limitazioni hardware ed eventuali inefficienze degli algoritmi implementati non è stato possibile eseguire i test sul 15-puzzle, ma è stato utilizzato un 8-puzzle. Questo ha garantito sicuramente una maggiore velocità di risoluzione, ma allo stesso tempo potrebbe aver diminuiti l'efficacia di certe euristiche rispetto ad altre.

La Tabella 1 contiene tutti i risultati ottenuti dai test. Ogni riga corrisponde a un'euristica diversa che è stata testata.
La prima colonna mostra il valore medio della funzione euristica sulle 1000 configurazioni iniziali. Come ci aspettavamo la Manhattan Distance ha un valore di $f(n)$ minore delle altre e questo indica un maggior numero di nodi generati dall'algoritmo A*. A seguire, contrariamente a quanto ci aspettavamo, ci sono i Disjoint Pattern Databases che hanno un valore iniziale di $f(n)$ molto simile ai Linear Conflicts. Questo molto probabilmente è dovuto al fatto di aver utilizzato un 8-puzzle a differenza di un 15-puzzle in cui invece i Disjoint Databases avrebbero avuto un valore maggiore. I Non Additive Databases sono risultati di gran lunga più efficienti rispetto alle altre euristiche per quanto riguarda il valore iniziale dell'euristica.

Funzione euristica	Valore Euristica	Nodi generati	nodi/s	Tempo(s)	Tutte le soluzioni
Manhattan Distance	14.192	1574.389	53199.091	0.0296	1575.768
Linear Coflicts	15.286	855.771	24167.830	0.0354	856.825
Disjoint Databases	15.278	940.076	5143.032	0.1827	941.078
Disjoint + Reflected	16.1	573.178	2427.276	0.2361	574.226
Non Additive Databases	21.014	95.05	1822.616	0.0521	96.048

Table 1: Risultati dei test

La seconda colonna mostra invece il numero di nodi generati all'algoritmo A* per ogni euristica. Anche in questo caso la migliore è risultata essere la Non Additive.
La terza e la quarta colonna rappresentano rispettivamente la media dei nodi generati al secondo e il tempo impiegato dall'algoritmo per arrivare alla soluzione. Possiamo notare che i tempi di esecuzione dei Disjoint Databases, dei Reflected Database e dei Non Additive Databases sono relativamente grandi rispetto alle più semplici Manhattan Distance. Dalla sezione 4.1 possiamo però evincere che i nodi generati sono all'incirca quelli previsti, dunque questa inefficienza dei database è sicuramente dovuta all'implementazione non ottima dei confronti dei nodi con i database.
L'ultima colonna infine mostra il numero totale di nodi generati finché non sono state trovate tutte le possibili soluzioni ottime. I risultati sono diversi da quello che ci aspettavamo, poiché questi valori sono vicini al numero di nodi generati per trovare la prima soluzione ottima. Anche questo molto probabilmente dipende dal fatto di aver utilizzato un 8-puzzle al posto di un 15-puzzle, il quale garantisce un numero di mosse possibili molto maggiore a ogni iterazione.

4.1 Nodi generati

Il numero di nodi generati all'algoritmo A* è un valore che dipende molto dall'efficienza dell'euristica. Se un'euristica h_1 domina una seconda euristica h_2 allora h_1 dovrebbe garantire generalmente la generazione di un numero minore di nodi rispetto a h_2. Il goal state dovrebbe quindi essere raggiunto in meno passi e il tempo di esecuzione dovrebbe diminuire. Come possiamo notare dal grafico 1

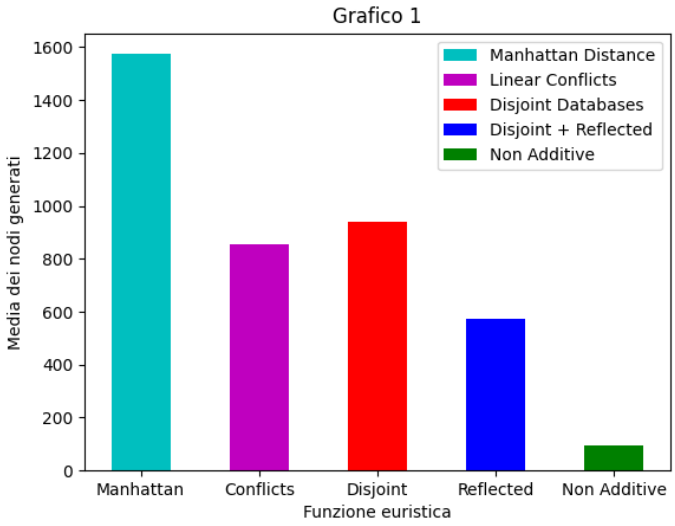

Figure 5: Media dei nodi generati da ogni euristica

(Figura 5) la Manhattan Distance genera molti più nodi delle altre euristiche. Contrariamente a quanto ci aspettavamo i Linear Conflicts risultano migliori dei Disjoint Databases. Questo risultato ovviamente è evidenziato dal fatto di aver utilizzato un 8-puzzle per i test, ma può anche essere dovuto alla scelta dei gruppi di tessere dei due database, che magari non era quella ottima. Come possiamo vedere dalla colonna successiva infatti, i Disjoint + Reflected mostrano già un netto miglioramento rispetto ai precedenti.
Infine è evidente che i Non Additive Databases sono stati di gran lunga più efficienti rispetto a tutte le altre euristiche, come era possibile prevedere dal valore iniziale dell'euristica molto maggiore.

Riferimenti

[1] R.E. Korf, A. Felner, Disjoint Pattern Database heuristic, 2002
[2] S. Russell, P. Novig, Artificial Intelligence, A modern approach (2020)