Eight Puzzle

Intelligenza Artificiale

Giulia Forasassi

Aprile 2018

1 Abstract

Nel seguente elaborato, partendo da un'implementazione dell'algoritmo A*, si vuole implementare la strategia di generazione di euristiche ammissibili e analizzare sperimentalmente A* realizzata con l'euristica basata sulla distanza Manhattan con A* basata sui database pattern in termini di numero di nodi espansi, penetranza e branching factor.

2 Descrizione del problema

Il gioco dell'otto, versione ridotta del famoso rompicapo "Il Gioco del 15", si svolge su una tavola 3x3 divisa in righe e colonne su cui sono posizionate 9 tessere quadrate contenenti numeri dall'1 all'8 e una tessera vuota. Le tessere possono scorrere in verticale o in orizziontale ma il loro spostamento è limitato dall'esistenza del singolo spazio vuoto. Partendo da una configurazione iniziale casuale lo scopo del gioco è riuscire a riordinare le tessere facendo le opportune mosse in modo da giungere alla configurazione goal desiderata.

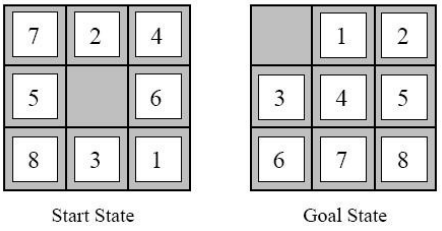

Figura 1: Esempio di stato iniziale e stato finale del gioco

3 Descrizione dell'esperimento

Il problema sopra descritto è possibile risolverlo con un semplice algoritmo di ricerca che prova tutte le possibili combinazioni di mosse eseguibili per trovare la soluzione. Questo tipo di ricerca si dice che opera alla cieca (*blind search*), perché non usa informazioni riguardo alla particolare disposizione delle caselle per indirizzare la ricerca.

Per rendere più efficiente questo procedimento si usano le funzioni euristiche, che servono per dare indicazioni su dove cercare la soluzione. In particolare in questo esperimento, vengono confrontate due euristiche, quella basata sulla distanza Manhattan e quella che sfrutta i Database di Pattern.

I parametri usati per valutare la prestazione degli algoritmi sono: numeri di nodi espansi, penetranza ed effective branching factor.

Un nodo viene considerato espanso quando i suoi figli vengono aggiunti alla frontiera. Con penetranza si intende il rapporto tra numero di nodi espansi (N) e profondità (d), ovvero il numero di passi dalla radice lungo il cammino che porta alla soluzione ottima. Il branching factor è il numero massimo di successori di un nodo, ma non è molto indicativo per l'analisi in quanto non considera che ogni nodo può avere un numero di successori inferiori al numero massimo. Quindi si valuta l'effective branching factor, che corrisponde ad un ipotetico branching factor nel caso in cui tutti i nodi avessero lo stesso numero di successori, ed è il numero b per cui vale la seguente formula:

$N + 1 = 1 + b + b^2 + \ldots + b^d$

Per entrambe le euristiche, si calcolano i tre valori come media su un certo numero di test, vengono infine mostrati i risultati.

Euristica con distanza Manhattan

Data una casella di indice i, j, si definisce per essa la distanza Manhattan (MD_{ij}), ovvero la quantità minima di spostamenti necessari per spostare la casella nella posizione obbiettivo, secondo la seguente formula:

$MD_{ij} = |i_g - i| + |j_g - j|$

Dove i_g e j_g sono gli indici di riga e colonna nello stato goal che deve avere la casella da spostare.

L'euristica basata sulla distanza Manhattan è data dalla sommatoria della distanza Manhattan di ogni casella.

Euristica con Database di Pattern

Si definisce un sottoproblema, identificato da un sottoinsieme delle caselle originali, come un problema del gioco dell'otto dove per raggiungere lo stato goal è sufficiente che siano nella corretta posizione soltano le caselle prese in considerazione.

L'idea di base di questa euristica è sfruttare il costo di uno o più sottoproblemi. Per far ciò si calcolano inizialmente i costi di tutte le istanze di un sottoproblema e memorizzandone le informazioni si genera un cosiddetto database di pattern.

In generale, nel caso di utilizzo di più sottoproblemi, come euristica si può usare il massimo dei costi dei vari sottoproblemi. Si può avere un'euristica più accurata se i sottoproblemi sono disgiunti, in tal caso si può fare la somma dei singoli costi.

In questo esperimento sono stati usati due sottoproblemi disgiunti: il primo formato dai numeri che vanno da 1 a 4, mentre il secondo è formato dai numeri che vanno da 5 a 8.

4 Aspettative

Ci si aspetta che le prestazioni dell'algoritmo di ricerca siano migliori quando si usa l'euristica basata sui database di pattern disgiunti, in quanto con l'euristica basata sulla distanza Manhattan non si possono rilevare particolari situazioni svantaggiose.

5 Implementazione

Si implementa quanto descritto precedentemente mediante il linguaggio di programmazione Python, versione 2.7. Il programma è composto da 5 file, di cui due sono stati presi dal sito http://aima.cs.berkeley.edu/python/readme.html e leggermente modificati:

- **Utils.py**: sono state prese soltanto le funzioni di interesse per il programma; inoltre sono state aggiunte una classe e una funzione, utili per calcolare l'effective branching factor trovando una giusta approssimazione della formula precedente mediante il metodo di Newton contenuto nella libreria scipy.optimize.
- **Search.py**: sono state prese le classi e le funzioni di interesse, inoltre nella funzione *graph_search* è stato aggiunto il conteggio dei nodi espansi.

Mentre gli altri tre file sono stati creati:

- **EightPuzzle.py**: in questo file sono presenti due classi:
  - *PuzzleState*: rappresenta lo stato del puzzle ovvero la disposizione delle caselle; il costruttore di essa crea inizialmente lo stato goal (o lo stato che viene passato come parametro) ed effettua il numero indicato di mosse causali per ottenere lo stato iniziale dal quale verranno testati gli algoritmi. Inoltre sono presenti due funzioni: la funzione *swap* che serve per scambiare due elementi del puzzle e la funzione *move* che viene utilizzata per muovere una casella nel puzzle.
  - *Puzzle*: definisce il problema del puzzle; il costruttore permette di specificare lo stato iniziale, che altrimenti viene creato casualmente, e lo stato goal. Viene ridefinita la funzione *successor* in modo da sapere quali sono le azioni possibili di ogni stato, e inoltre viene ridefinita la funzione *goal_test*.

  Inoltre è presente una funzione *matrixToString* che serve per rappresentare la matrice in formato stringa, utile per controllare se uno stato è già stato visitato ed eventualmente per visualizzarlo.
- **HeuristicSearch.py**: vengono implementate la funzione euristica basata sulla distanza Manhattan e quella basata sui Database di Pattern. Per realizzare quest'ultima si è definita una classe *SubPuzzle* che definisce un sottoproblema della classe principale *Puzzle*. Un'ultima funzione presente è la *createDictionary* che serve per creare un dizionario nel quale viene associato ad ogni sottoproblema il costo della sua soluzione. Sono stati creati due dizionari usando la libreria pickle, eseguendo questo file.
- **Main.py**: inizialmente vengono definite delle variabili che permettono di controllare l'esecuzione dell'esperimento: *n*, lato del quadrato del puzzle, *nScrambles*, numero massimo di scambi fatti per mischiare il puzzle, *numTests*, numero di test da eseguire.

  Per ogni test viene creata un istanza del problema del gioco dell'otto e una copia di esso: sul problema originale viene eseguita A* con l'euristica distanza Manhattan mentre sulla copia si esegue A* con l'altra euristica, e su entrambe vengono calcolati i parametri di prestazione facendone una media sul numero di test specificato da *numTests*. Infine vengono mostrati i risultati ottenuti sia su console sia mediante degli istogrammi.

6 Utilizzo del programma

Per eseguire l'esperimento i passi sono i seguenti:

1. *Opzionale*: Modificare i parametri relativi alla creazione dei dizionari, che si trovano all'inizio del file HeuristicSearch.py.
2. Creare i dizionari eseguendo il file HeuristicSearch.py, oppure usare quelli forniti. Nel caso si scelga la prima opzione, il tempo di creazione totale è di circa 1 ora.

3. *Opzionale*: Modificare parametri relativi all'esecuzione dei test, che si trovano all'inizio del file Main.py.
4. Eseguire il file Main.py.

7 Risultati sperimentali

Si riportano di seguito i risultati approssimati ottenuti nell'esperimento con numero massimo di scambi pari a 50 e un numero di test uguale a 10:

Tabella 1: Confronto delle due euristiche

	Manhattan heuristic	Pattern databases heuristic
Average number of expanded nodes	1749.4	91.2
Average effective branching factor	1.2650	1.0781
Average penetrance	0.0515	0.4181

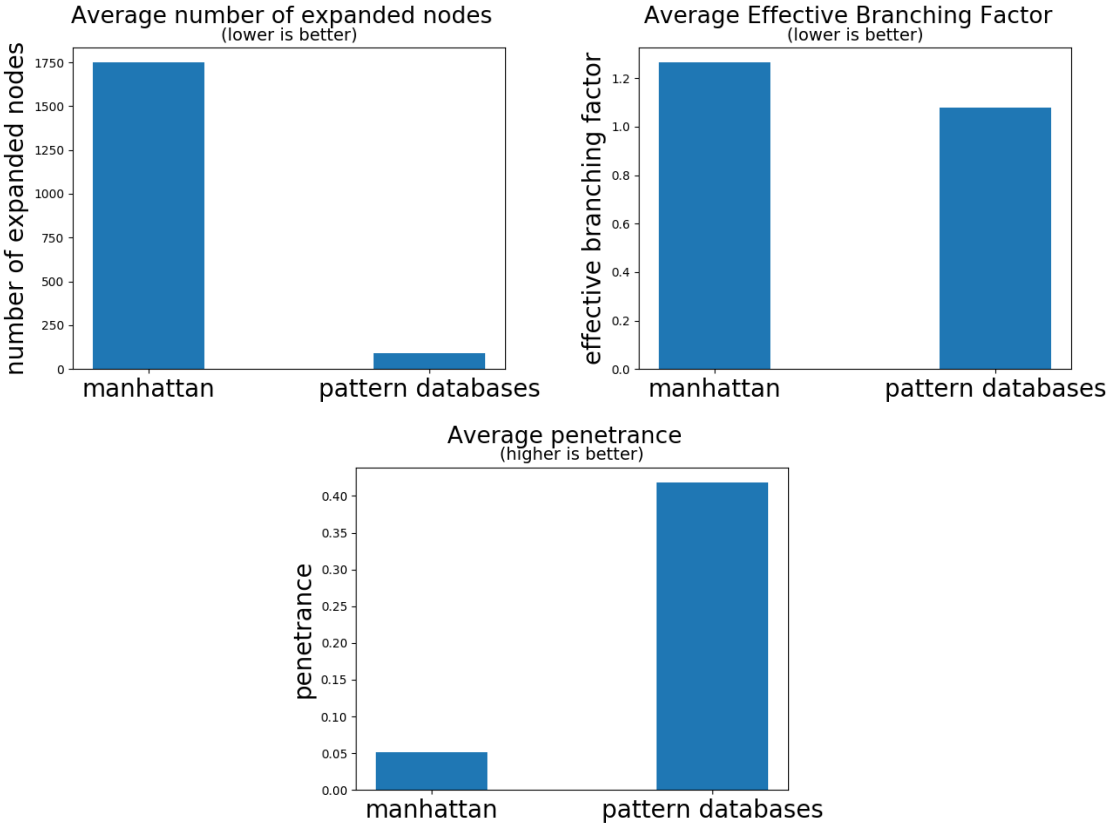

Figura 2: Comparazione delle prestazioni dei due algoritmi

8 Analisi risultati e conclusione

I risultati ottenuti sono concordi con ciò che ci si aspettava e quindi si osserva che, nel caso in cui i database siano disgiunti, l'euristica basata sui database di pattern risulta molto più accurata rispetto a quella basata sulla distanza Manhattan.

Infatti osservando i grafici notiamo che l'euristica basata sui database di pattern in media espande un numero minore di nodi, ha un branching factor di valore minore ed ha una maggior penetranza, per questo risulta migliore.