A* Con Disjoint Pattern Databases

Femia Davide

Indice

1 Introduzione

In questo esercizio si parte da un'implementazione dell'algoritmo A* per poi implementare dunque le euristiche descritte in (Korf and Felner, 2002) tentando di riprodurre i risultati riportati in Table 1 di tale articolo.
Purtroppo ci dovremo accontentare di riprodurre i risultati sul gioco dell'8 anzichè del 15 per motivi che verranno descritti nella sezione Scelte Implementative.

2 Descrizione Esperimenti

2.1 Dati e Misure

Gli stati iniziali da cui iniziare la ricerca verso il goal con le varie euristiche vengono generati casualmente attraverso il metodo RandomState della classe TilesProblem che a sua volta utilizza la funzione randrange della libreria random di python per scegliere quale azione applicare a partire dallo stato goal(vengono effettuate al più 100 azioni di default, il parametro 100 può essere modificato attraverso il costruttore della classe TilesProblem).

Le misure vengono effettuate all'interno dell'algoritmo AStar e vengono salvate in un oggetto Solution(la dichiarazione della classe Solution si trova nel modulo Solver.py). A quel punto un oggetto AStarSolver utilizza l'oggetto Solution per mostrare a video appunto le misure.

I secondi vengono misurati grazie alla funzione default_timer della libreria timeit. Il numero di nodi generati vengono misurati invece grazie alle dimensioni delle strutture dati che vengono utilizzate per memorizzare appunto i nodi stessi, tali strutture dati sono un dizionario python per i nodi incontrati durante la ricerca(explored) e un oggetto AStarFrontier la cui dichiarazione della relativa classe si trova nel modulo Frontier.py

2.2 Specifiche Della Piattaforma di Test

La piattaforma di test è un HP Notebook con Sistema Operativo Windows 10 Home, 12 GB di RAM e con il seguente processore:
Intel Core i5-7200U CPU @ 2.50 GHz 2.70 GHz

3 Documentazione Codice

3.1 Interazione fra Moduli

Nel progetto sono disponibili 2 eseguibili. uno si trova nel modulo DBLoader.py che crea i Pattern Database, li serializza e li scrive su dei file di testo('DB1.txt','DB2.txt','DB3.txt','DB4.txt') attraverso la libreria pickle di python; l'altro si trova nel modulo Main.py ed effettua la ricerca su 100 istanze random del gioco dell'8, generate come descritto in precedenza, attraverso le euristiche descritte in (Korf and Felner, 2002).

3.2 Scelte Implementative

Nella realizzazione degli esperimenti sul gioco del 15 mi sono imbattuto in un Memory Error dovuto alla dimensione della memoria RAM relativa alla piattaforma di Test e probabilmente ad una implementazione degli algoritmi non molto efficiente.
In effetti l'implementazione dell'algoritmo CreateDB nel modulo DBLoader.py prevede di creare un dizionario python(quindi contenente coppie chiave, valore) con chiavi delle serializzazioni di tutti i possibili stati di qualche sottoproblema(alcuni sottoproblemi arrivano ad avere $16!/8! = 518.918.400$ stati diversi) e come valori le soluzioni ottime di ogni possibile stato del sottoproblema; per poi serializzare il dizionario e scriverlo su file grazie alla libreria pickle di python.
Facendo un rapido conto e osservando che la chiave viene memorizzata con 32B dato che si tratta di una stringa di almeno 32 caratteri, mentre il valore invece viene memorizzato con 4B dato che è un intero, dunque abbiamo:

$$(32 + 4) * 518.918.400 = 18.681.062.400B$$

ci rendiamo conto dunque che per questa piattaforma di test e questa implementazione non è possibile replicare i risultati descritti in (Korf and Felner, 2002).

Ho scelto di caricare i Pattern Databases al momento della creazione di un oggetto TilesProblem, ma questo da problemi nel caso in cui si decida di tentare di riprodurre la creazione dei database(già forniti nei file di testo) per conto proprio. Per riprodurre la creazione dei database bisogna dunque innanzitutto cancellare i contenuti dei file di testo, commentare la riga di codice che si occupa della creazione dei database(la chiamata del metodo LoadDatabases nel costruttore di Tiles Problem) e in seguito eseguire il mo-

dulo DBLoader.py.

Per l'euristica dei Disjoint Database ho scelto come sottoproblemi quelli relativi alle celle (1,2,3,4) e (5,6,7,8).

Mentre per l'euristica Disjoint + reflected ho scelto come altri sottoproblemi aggiuntivi ai primi 2 descritti per l'euristica Disjoint Database altri 2 sottoproblemi relativi alle celle (2,5,7,8) e (1,3,4,6)

4 Presentazione Risultati

Heuristic Function	Value	Nodes	Nodes/Sec	Seconds	All Solutions
Manhattan distance	10.53	378	10576	35.78 10^{-3} s	398
Linear Conflicts	10.99	246	5720	42.94 10^{-3} s	256
Disjoint Database	13.23	70	9088	7.65 10^{-3} s	74
Disjoint + reflected	13.63	55	6681	8.24 10^{-3} s	58

Tabella 1: Risultati sperimentali sul gioco dell'8

Nella Tabella 1 possiamo vedere i risultati sperimentali delle diverse euristiche a confronto sulle 100 istanze random del gioco dell'8.
Il campo Value indica il valor medio della relativa euristica sulle 100 istanze generate casualmente.
Il campo Nodes indica il numero medio di nodi generati per trovare il primo stato goal ottimo.
il campo Nodes/Sec indica la frequenza di nodi generati al secondo per avere un'idea della velocità delle varie euristiche.
Il campo Seconds indica il tempo impiegato dall'algoritmo per trovare il primo stato goal ottimo.
Il campo All Solutions indica il numero medio di nodi generati per trovare tutti gli stati goal ottimi.

5 Analisi Risultati

Come ci si aspettava dalla teoria i database di pattern si rivelano euristiche molto efficaci malgrado il tempo necessario alla creazione dei database stessi, infatti confrontati alle altre euristiche in termini di tempo risultano molto più rapidi, anche in termini di nodi generati per trovare gli stati goal si rivelano molto efficaci. I risultati invece un pò inattesi sono la lentezza dell'euristica

Linear Conflicts rispetto all'euristica Manhattan malgrado la prima generi meno nodi rispetto alla seconda per trovare le soluzioni ottime, la seconda invece batte la prima in termini di tempo di esecuzione, tutto ciò probabilmente è dovuto all'implementazione della funzione euristica Linear conflicts che richiede più tempo rispetto alla funzione Manhattan distance(entrambe le funzioni si trovano nel modulo TilesProblem.py) e molto spesso è inconcludente rispetto alla prima infatti il loro valore medio sullo stato iniziale è quasi uguale.
Un'analoga analisi la possiamo fare sulle euristiche Disjoint Databases e Disjoint+Reflected.