Ricerca Euristica progetto esame di Intelligenza Artificiale

Mandelli Lorenzo

2018-2019

Abstract

É stato realizzato un problema di ricerca di stato elaborando e definendo una specifica metrica euristica e implementando l'algoritmo di ricerca A*. Infine sono stati confrontati i risultati della performance ottenuti con la ricerca cieca e la ricerca informata.

1 Assegnamento: assemblaggio di prodotti

L'assemblaggio di un certo prodotto richiede lo svolgimento di **N** compiti {c1,...,cn}, ognuno dei quali richiede un tempo (discreto) ti,i = 1,..., n per essere completato. Alcuni compiti richiedono certi prerequisiti, ovvero il compito ci richiede il completamento dei compiti Pi $\subset${c1,...,cn} prima di poter iniziare (Pi è l'insieme vuoto se non ha prerequisiti). Sono disponibili **M** impiegati ai quali si possono assegnare i compiti e ogni impiegato può solo svolgere un compito ad un determinato istante di tempo. Supponendo di dover assemblare **K** copie identiche del prodotto, va trovata la sequenza temporale di assegnamento dei compiti ai dipendenti in modo da minimizzare il tempo complessivo per completare tutti gli assemblaggi. Si scriva un programma (in un linguaggio di programmazione a scelta) basato su tecniche di ricerca per risolvere il problema, definendo un'euristica da utilizzare con A*.

2 Implementazione

Al fine di eseguire l'elaborato assegnato sono state usate le seguenti classi:

- **Compito**: classe che rappresenta i singoli task dei Prodotti da eseguire. A ogni compito è assegnato un tempo di esecuzione, un identificatore e una lista di riferimenti ad altri compiti dello stesso Prodotto che rappresentano i suoi requisiti. Un compito non può essere assegnato a un Impiegato fino a che tutti i suoi requisiti non sono stati completati.

- **Prodotto**: classe che racchiude un insieme di Compiti. Essi si dividono in: disponibili (avaibleCompiti), non disponibili (unavaibleCompiti), ultimati (doneCompiti) e in lavorazione (compitiWorkingOn). Sono disponibili inoltre i metodi di creazione di copia di un Prodotto e un metodo per la generazione di un Prodotto casuale, quest'ultimo metodo produce i requisiti tra compiti in base a una probabilità impostabile dall'utente.

- **State**: classe che rappresenta lo stato del problema a un certo istante di tempo. Essa è caratterizzata da un attributo (numberFreeEmployers) che indica il numero di impiegati liberi e da due liste (doneProductList, undoneProductList) di oggetti Prodotto che rappresentano i quali Prodotti sono stati ultimati e quali non.

- **Node**: classe che implementa a struttura dei nodi utilizzata dall'algoritmo preso in esame, a_star_search, per navigare nell'albero di ricerca. Tiene traccia dello stato (state), della profondità (depth), dell'azione utilizzata per giungere a se stesso (action), del costo necessario (path_cost) e del nodo "padre" da cui deriva (parent). Fornisce inoltre i metodi per generare da un Nodo tutti i possibili Nodi "figli".

- **Problem**: classe che definisce la struttura astratta per la classe RealProblem.

- **RealProblem**: classe che rappresenta il problema di assemblaggio di prodotti preso in esame. E' caratterizzata da un riferimento allo stato iniziale (initial) e un riferimento allo stato goal (goal) che crea a partire da quello iniziale fornito nel costruttore. Implementa i metodi:

  - action: che ritorna la lista delle possiili azioni effettuabili in un certo Stato.
  - result: che ritorna lo stato risultante dalla scelta di una particolare azione in un particolare stato.
  - goal_test: che analizza uno Stato e ritorna True se esso coincude con lo stato goal.
  - **h**: che, data uno Stato, analizza le tre euristiche h1, h2 e h3 per fare una stima della "distanza" che lo separa dallo Stato goal.
  - **h1**: ritorna il massimo tempo di esecuzione presente tra tutti i compiti (disponibili, non disponibili e in lavorazione) di tutti i prodotti non ultimati.
  - **h2**: ritorna il tempo massimo necessario a ultimare un compito attualmente non disponibile considerando la somma maggiore delle catene dei suoi requisiti.
  - **h3**: ritorna la stima del tempo necessario a ultimare tutti i compiti considerando la somma dei tempi di ogni compiti diviso il numero totale di impiegati.

M	K	N	N*M	Media tempi(s)
1	1	10	10	0.039
5	1	10	10	0.016
10	1	10	10	0.015

Table 1: **(A)**

M	K	N	N*M	Media tempi (s)
1	2	10	20	0.273
10	2	10	20	0.024
20	2	10	20	0.026

Table 2: **(B)**

3 Risultati

Dai dati empirici, ottenuti utilizzando la probablità di formazione di un requisito di default (10%), si ottengono le seguenti tabelle:

3.1 Algoritmo A_star_search

:

M	K	N	N*M	Media tempi(s)
1	3	10	30	0.436
15	3	10	30	0.080
30	3	10	30	0.069

Table 3: **(C)**

M	K	N	N*M	Media tempi(s)
1	4	10	40	0 1,031
20	4	10	40	0.202
40	4	10	40	0.177

Table 4: **(D)**

M	K	N	N*M	Media tempi(s)
1	5	10	50	2,101
25	5	10	50	0.326
50	5	10	50	0.315

Table 5: **(E)**

3.2 Algoritmo Best_first_graph_search

:

3.3 Conclusioni

Dai dati si evince la superiorità nel tempo di secuzione di *a_star_search* rispetto a *best_first_graph_search* a parità di input . Le euristiche hanno quindi permesso di ottenere migliori performance rispetto all'utilizzo del solo path_cost come criterio di esplorazione dell'albero.
Si nota inoltre che all'aumentare del numero di impiegati diminuisce il

M	K	N	N*M	Media tempi(s)
1	1	10	10	15,809
5	1	10	10	1.016
10	1	10	10	0.006

Table 6: **(F)**

M	K	N	N*M	Media tempi(s)
1	1	15	15	0 26,101
7	1	15	15	0.028
15	1	15	15	0.021

Table 7: **(G)**

tempo necessario al completamento del problema.
Gli esperimenti indicano che il tempo di esecuzione può notevolmente variare (anche di un fattore 100) in base all'istanza di input presente. Essendo i vincoli tra compiti generati randomicamente si è ottenuto che in caso di maggior numero di vincoli presenti la difficoltà del problema decresce, mentre aumenta con la maggiore presenza di compiti privi di requisiti. Ciò è dovuto al diminuire di possibili strade percorribili dall'algoritmo a ogni istante di tempo in caso di maggior numero di vincoli.