
UNIVERSITÀ
DEGLI STUDI
FIRENZE

# Relazione sull'elaborato di Intelligenza Arificiale del corso di laurea Ingegneria Informatica

Anno 2018-2019

Lorenzo Mandelli matricola 6155019

# ASSEMBLAGGIO PRODOTTI

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.

OSSERVAZIONI

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

Nle caso di uilizzo dell'algoritmo *a_star_search*:

| m | k | n | n*m | Media tempi ottenuti |
|---|---|---|---|---|
| 1 | 1 | 10 | 10 | 0,03909564 |
| 5 | 1 | 10 | 10 | 0,01636251 |
| 10 | 1 | 10 | 10 | 0,01512453 |

| m | k | n | n*m | Media tempi ottenuti |
|---|---|---|---|---|
| 1 | 2 | 10 | 20 | 0,27306628 |
| 10 | 2 | 10 | 20 | 0,02434176 |
| 20 | 2 | 10 | 20 | 0,02612103 |

| m | k | n | n*m | Media tempi ottenuti |
|---|---|---|---|---|
| 1 | 3 | 10 | 30 | 0,43669564 |
| 15 | 3 | 10 | 30 | 0,08078381 |
| 30 | 3 | 10 | 30 | 0.06982484 |

| m | k | n | n*m | Media tempi ottenuti |
|---|---|---|---|---|
| 1 | 4 | 10 | 40 | 1,03151347 |
| 20 | 4 | 10 | 40 | 0,20170954 |
| 40 | 4 | 10 | 40 | 0,17756774 |

| m | k | n | n*m | Media tempi ottenuti |
|---|---|---|---|---|
| 1 | 5 | 10 | 50 | 2,10588167 |
| 25 | 5 | 10 | 50 | 0,32629078 |

| | | | | |
|---|---|---|---|---|
| 50 | 5 | 10 | 50 | 0,31515741 |

Nel caso di utilizzo dell'algoritmo di *best_first_graph_search*:

| m | k | n | n*m | Media tempi ottenuti |
|---|---|---|---|---|
| 1 | 1 | 10 | 10 | 15.8092975 |
| 5 | 1 | 10 | 10 | 1,01629798 |
| 10 | 1 | 10 | 10 | 0,00602293 |

| m | k | n | n*m | Media tempi ottenuti |
|---|---|---|---|---|
| 1 | 1 | 15 | 15 | 26.4820055 |
| 7 | 1 | 15 | 15 | 0.02812423 |
| 15 | 1 | 15 | 15 | 0.02187440 |

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 tempo necessario al completamento del problema.
Gli esperimenti indicano che il tempo di esecuzione può notevolmente variare (amche 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.