Scuola Universitaria Professionale
della Svizzera Italiana

Dipartimento
Tecnologie
Innovative

Traveling Salesman Problem
18ª Coppa di Algoritmi

Studente: Mellace Simone

Data: 5 maggio 2018

Problema

Il problema del commesso viaggiatore consiste nella ricerca del ciclo hamiltoniano più breve all'interno di un grafo pesato completo. Questo tipo di problema appartiene alla classe dei problemi *NP-Completi*. In Figura 1 è mostrato uno dei problemi TSP con soluzione, sottoinsieme compreso nei problemi NP.

Il compito assegnato è di generare una soluzione ammissibile e quanto più possibile vicina a quella ottimale, avendo a disposizione solamente 3 minuti di esecuzione del programma. Il programma deve poter lavorare su 10 problemi diversi, dati facoltativamente dei valori iniziali. I problemi da risolvere sono:

- *ch130*;
- *d198*;
- *eil76*;
- *fl1577*;
- *kroA100*;
- *lin318*;
- *pcb442*;
- *pr439*;
- *rat783*;
- *u1060*.

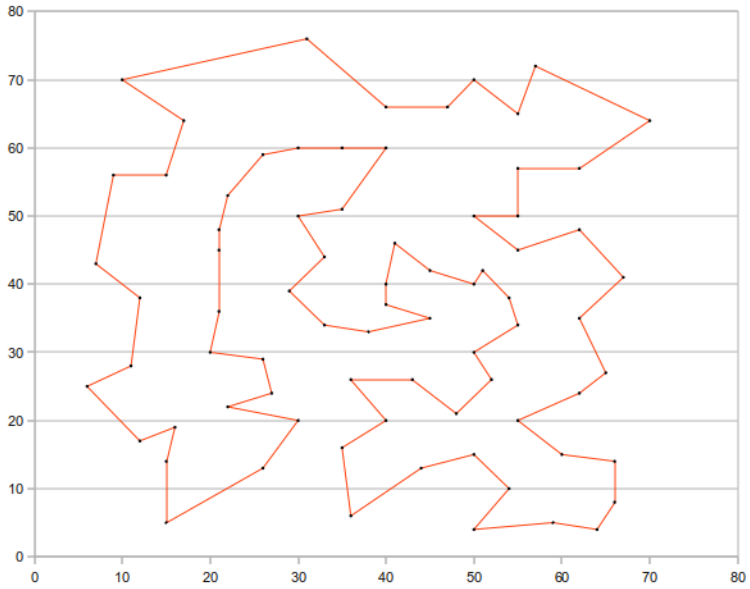

Figura 1: Eil76, problema di TSP da 76 città

Soluzioni implementate

Algoritmo costruttivo

L'algoritmo costruttivo ha il compito di generare una prima soluzione ammissibile.

L'algoritmo scelto è stato il Nearest Neighbour; in questo modo partendo da un nodo si ottiene la prima soluzione ammissibile. Questa scelta necessità più tempo del Random, ma la soluzione risultante è più adatta per gli algoritmi che verranno applicati in un secondo tempo. Dato un set di città iniziale con questo algoritmo la prima soluzione ottenuta sarà sempre la stessa; per questo motivo si poteva decidere di implementare un metodo random per la costruzione di una soluzione iniziale.

Nella versione implementata si parte sempre dal primo nodo del problema per creare una soluzione.

Algoritmo di ottimizzazione locale

Gli algoritmi di ottimizzazione locale sono utili a trovare i minimi locali.

Per questa fase è stato utilizzato il 2-Opt, in questo modo si riesce ad analizzare al meglio il risultato ottenuto dalla prima fase, ottenendo dei risultati con un margine di errore minore.

La versione adottata del 2-Opt è quella del best improvement, si migliora quindi il tour con lo scambio più profiquo. Ovviamente questo richiede più tempo d'esecuzione per la ricerca del miglior scambio, ma il tour risultante è migliore.

Algoritmo meta-euristico

La meta euristica risulta importante per riuscire a spostarsi da minimi locali a minimi globali.

Per questa fase si è deciso di adottare l'algoritmo Ant Colony System for TSP, algoritmo meta-euristico ispirato dal movimento naturale delle formiche nelle loro colonie.

La versione implementata prende spunto da "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", di Marco Dorigo e Luca Maria Gambardella. I dettagli che sono variati da questa implementazione verranno riportati in seguito.

In questa implementazione la variazione del feromone, nell'aggiornamento del feromone locale è sempre la temperatura iniziale. L'apporto di feromone globale avviene anche se il miglior tour di un set di formiche, non è meglio dell'attuale risultato migliore, in questo modo si cerca di dare alle formiche più possibilità per esplorare nuovi percorsi.

Durante alcune prove si è manifestata la difficolta di terminazione nei 3 minuti a disposizione, questo accadeva nei problemi con molte città, dato che inizialmente il numero di formiche per problema era fissato a 10. Per ovviare a questo problema e rispettare i vincoli temporali del problema, è stato fatto variare il numero di formiche utilizzate in modo inverso rispetto al numero di città presenti nel problema. In questo modo si è trovato un numero ottimale di formiche per ogni problema.

Il numero di formiche adottato per questi problemi, come anche il seed iniziale per impostare il Random e la soglia sopra la quale si può fare exploration, sono quindi variate a seconda del propblema, al fine di riuscire a diminuire il costo del miglior tour; questi valori possono essere passati quando si fa partire l'algoritmo.

La soglia che divide l'esecuzione delle formiche fra exploration ed exploitation è quindi variabile. Nella versione implementata, se si pratica l'exploration il miglior nodo viene sempre escluso finché possibile, così da aumentare le probabilità di variazione del percorso per ogni formica.

Un modo per velocizzare ulteriormente il processo potrebbe essere quello di utilizzare delle candidate list.

Esecuzione programma

Piattaforma

La piattaforma usata è un computer portatile Lenovo Y50. Nella Tabella 1 sono mostrati alcuni parametri rilevanti.

Tabella 1: Piattaforma usata per i test

Sistema Operativo	Windows 10 Pro
Processore	Intel(R) Core(TM) i7-4720HQ @ 2.6GHz
RAM	16GB

I risultati ottenuti sono comunque stati validati sulla macchina Windows che ci è stata messa a disposizione.

Compilazione ed esecuzione

La compilazione risulta molto semplice, è sufficiente digitare quanto segue:

```
javac *.java
```

Al fine di facilitare l'utilizzo del software implementato è stato sviluppato un file .JAR, il quale va utilizzato nel seguente modo:

```
java -jar nomeJar.jar problemName seed antsNumber threshold
```

Dove *problemName* è il nome esatto del file da analizzare (estensione compresa), il quale deve essere contenuto nella cartella *ALGO_cup_2018_problems*. *seed* è il seme voluto per inizializzare il random e *antsNumber* è il numero di fomiche da utilizzare nella ricerca della soluzione ottimale per il problema deciso. Il programma produce anche un output minimale sul risultato ottenuto nella console; quando il tempo sarà esaurito la soluzione migliore verrà scritta sul file *problemName.opt.tour* nella cartella *ALGO_cup_2018_solutions*

Per la riproduzione del JAR dai file java, una volta compilato il codice sorgente sarà possibile digitare quanto segue:

```
jar cmvf META-INF/MANIFEST.MF nomeJar.jar *.class
```

Il file MANIFEST.MF è fisso e va solo ad indicare quale classe è la principale.

Per automatizzare la procedura di esecuzione è fornito un file che fa partire automaticamente tutti i problemi date le condizioni iniziali ottimali trovate. Per far partire questo script su Windows basta fare un doppio click sul file oppure digitare in una cmd:

```
esecuzione_automatizzata.bat
```

Andando a cambiare i parametri del file si può far partire gli algoritmi adattando i parametri facilmente.

Risultati

Dopo diverse prove mirate a trovare i parametri migliori per ogni problema, i risultati ottimali sono mostrati in tabella 2.

Tabella 2: Soluzioni migliori con attributi ottimali per ogni problema

Problema	Optimum	Risultato	Errore	Seed	Ants	Soglia
ch130	6110	6110	0.0%	1155512583	3	0.95
d198	15780	15780	0.0%	-577850359	30	0.95
eil76	538	538	0.0%	1295242925	3	0.95
fl1577	22249	22737	2.19%	-1337044191	1	0.97
kroA100	21282	21282	0.0%	-1725146209	3	0.95
lin318	42029	42029	0.0%	543024183	15	0.97
pcb442	50788	51232	0.89%	2038991591	5	0.97
pr439	107217	107383	0.15%	98942990	5	0.97
rat783	8806	9026	2.50%	1480035758	1	0.97
u1060	224094	231482	3.30%	-451692501	2	0.97
media			0.90%			

Conclusioni

L'algoritmo implementato permette di ricavare delle soluzioni soddisfacenti: la media finale dell'errore è 0.90%. Purtoppo nei soli 3 minuti forniti l'algoritmo ACS utilizzato non riesce ad analizzare al meglio tutti i possibili tour e di conseguenza la soluzione ottenuta non è sempre ottimale. I problemi più rilevanti, nell'utilizzo di questo algoritmo si sono fatti notare specialmente sui problemi di grandi dimensioni, vedi:

fl1577
rat783
u1060

Per ovviare al problema è stato essenziale far variare il numero di formiche utilizzate in modo inverso alla cardinalità del problema, per contro il ristretto numero di formiche e il tempo limitante non permette la corretta interpretazione del feromone lasciato sul tracciato, come neanche la corretta terminazione da parte di tutte le formiche. Di conseguenza non è possibile trovare i tour ottimali a meno di non trovare i parametri - soprattutto i seed - ottimali.