Traveling Salesman Problem

19a Coppa di Algoritmi

**Studente:** Luca Ambrosio

**Data:** 03/05/2019

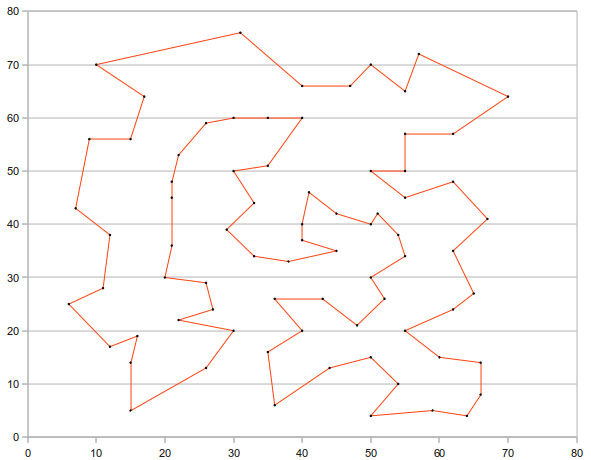
**Problema**

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 dei problemi NP.

Figura 1: Eil76, problema di TSP da 76 città

Il compito assegnato è di generare una soluzione ammissibile e quanto più possibile

vicina a quella ottimale (ovvero la migliore) in 3 minuti di esecuzione del programma. Il

programma deve poter lavorare su 10 problemi diversi:

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

Dove il numero che segue la sigla indica il numero di città presenti.

**Soluzioni implementate**

**Algoritmo costruttivo**

L’algoritmo costruttivo ha il compito di fornire una prima soluzione “grezza”, ma al contempo ammissibile in modo da poterla poi passare a un algoritmo di local search.

Di questa categoria di algoritmi ho deciso di implementare il Nearest Neighbor in quanto era molto semplice sia a livello logico che di implementazione.

La città di partenza viene scelta random, partendo da essa cerco la città più vicina in base alla matrice delle distanze appositamente precalcolata, e mi sposto.

Ogni città in cui mi muovo viene contrassegnata come visitata in modo da passarci una e una sola volta.

In base alla città di partenza la media dell’algoritmo può variare ma complessivamente si aggira attorno al 15% d’errore, risultati per nulla accettabili ma in questo modo abbiamo un tour valido per implementare il local search.

**Algoritmi di ottimizzazione locale**

L’algoritmo che ho scelto di implementare di questa categoria è il 2-opt.

Implementando la versione “classica”, in cui si eseguiva lo scambio con il *best gain* migliore di tutta l’esecuzione, la media migliorava notevolmente ma non scendeva comunque sotto il 5%.

Di conseguenza ho deciso di eseguire la mossa ogni volta che troviamo un nuovo *best gain* non solo il migliore di tutta l’esecuzione.

Dopo questo cambiamento ho ottenuto una media del 2%, risultato ottenuto scegliendo come città di partenza la migliore per questa combinazione di NN+2-opt (la città è stata scelta eseguendo il run con tutte le possibili combinazioni di città di partenza).

Nonostante questo, essendo che comunque l’algoritmo meta-euristico lavorava molto bene anche iniziando con una città random, non ho tenuto conto della migliore città di partenza, ma mi è stato utile per avere un buon punto di partenza per l’algoritmo SA.

**Algoritmi meta-euristici**

Infine ho deciso di implementare l’algoritmo meta euristico Simulated Annealing.

Il punto di partenza è stato comunque l’algoritmo visto in classe; partendo con una temperatura di 100 e un alpha di 0.97.

All’inizio facevo ciclare fino a che la temperatura era maggiore a un numero prossimo a zero e all’interno facevo iterare 100 volte l’algoritmo.

Avendo lo scheletro iniziale ho implementato dei miglioramenti inserendo una fase di diversificazione e una di intensificazione.

Per la diversificazione ho utilizzato un double bridge a cui passavo il mio tour corrente e sceglievo quattro nodi random che, seguendo l’algoritmo, andavo a scambiare creando dunque un nuovo tour valido ma con degli incroci, questa tecnica è indispensabile per non rimanere bloccati nei minimi locali.

Come intensificazione invece ho riutilizzato il local search precedentemente implementato e cioè il 2-opt andando dunque a togliermi gli incroci e di conseguenza migliorarmi il tour dopo aver applicato il double bridge.

Tour newTour = Algorithm2Opt.*apply2Opt*(*doubleBridge*(current.getTour()));

**Esecuzione del programma**

**Piattaforma**

La piattaforma usata è un computer portatile Asus ZenBook UX4. Nella Tabella 1 sono mostrati

alcuni parametri rilevanti.

Tabella 1: Piattaforma usata per i test

|  |  |
| --- | --- |
| Sistema Operativo | Windows 10 Home |
| Processore | Intel(R) Core(TM) i7-8550U 1.80GHz |
| RAM | 16 GB |

**Compilazione ed esecuzione**

Questo è tutto da riscrivere appena finisco come eseguire

Per compilare il software è presente un makefile, quindi è sufficiente digitare:

$ make

Per eseguire il programma è sufficiente digitare:

$ ./TSP −s seed −a algorithm −m map\_file −t tour\_file

dove *seed* è il seme per inizializzare il random, nel caso venga specifica l’opzione -s senza

indicare un seed esso sarà scelto autonomamente dall’applicazione, *algorithm* rappresenta

l’algoritmo da utilizzare fra quelli conosciuti dal software e infine *map\_file* e *tour\_file*

rispettivamente il file contenente la mappa del problema e il file contenente il tour della

soluzione trovata. Il programma produce anche un output minimale sul risultato ottenuto,

di seguito è presente un esempio:

$ ./TSP −m /home/mikol/TSP/map/ch130.tsp −t /home/mikol/TSP/sol/ch130.tour −a caso

ch130 37410

**Risultati**

In Tabella 2 sono mostrati i risultati migliori per ogni problema, la relativa percentuale d’errore e il seed dal quale partire per ottenere i migliori risultati.

Nella cartella test sono riportati i relativi file .opt.tour.

Tabella 2: Soluzioni migliori per ogni problema

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problema | Optimum | Risultato | Errore | Seed |
| ch130 | 6110 | 6110 | 0% | 1556029249552 |
| d198 | 15780 | 15781 | 0,006336734% | 1556029789563 |
| eil76 | 538 | 538 | 0% | 1556030369920 |
| fl1577 | 22249 | 1301989 | 5751.90% | 1330696248 |
| kroA100 | 21282 | 21282 | 0% | 1556031396763 |
| lin318 | 42029 | 42091 | 0,147299898% | 1556031781854 |
| pcb442 | 50788 | 50954 | 0,345409585% | 1556033037419 |
| pr439 | 107217 | 107217 | 0% | 1556033354706 |
| rat783 | 8806 | 9140 | 3,654266958% | 1556033765875 |
| u1060 | 224094 | 6326750 | 2723.26% | 1556034844654 |

**Conclusioni**

La percentuale finale è del …. Quindi sotto l’1% di errore ed è un risultato accettabile anche se si potrebbe migliorare.

Gli errori con percentuale più alta li trovo nei file con un elevato numero di città, ma soprattutto sul file rat783 in quanto ci sono le città divise in diversi cluster separati.

L’algoritmo si potrebbe migliorare utilizzando altre strutture come per esempio un minumun spanning tree e l’utilizzo delle candidate list.

Concludendo sono comunque soddisfatta dell’algoritmo, dei risultati e del livello del codice prodotto.

Data Firma

29 Aprile 2019 Nicol Allegra

L’algoritmo implementato è chiaramente inefficiente: la media finale dell'errore è 1635.11%.