Scuola Universitaria della Svizzera Italiana (SUPSI)

C08003 - Advanced Algorithms and Optimization

# $19^{th}$ Algorithms Cup

## Solving the Travelling Salesman Problem (TSP) with Java

Denys Vitali

April 2019

# Contents

# 1 Introduzione

Il presente documento contiene le informazioni riguardanti il mio metodo di risoluzione del problema del commesso viaggiatore (Travelling salesman problem, TSP). Questo progetto è un requisito del corso di *Algoritmi Avanzati ed Ottimizzazione (C08003)*, tenuto presso la *Scuola Universitaria Professionale della Svizzera Italiana (SUPSI)* durante il corso di laurea in ingegneria informatica.

## 1.1 Scopo

Lo scopo del progetto è quello di risolvere 10 problemi di TSP (originati dalla libreria TSPLIB) forniteci ad inizio corso sfruttando uno o più algoritmi sviluppati in Java.

## 1.2 Requisiti

Di seguito vengono riportati i requisiti per un corretto svolgimento della coppa:

- Limite di tempo: massimo 3 minuti per problema, il tempo è inteso da quando il software inizia a quando finisce.
- 10 problemi (forniti in un file .zip ad inizio corso)
- Linguaggio di programmazione: Java
- Obbligatorio l'utilizzo di Maven, vietato l'utilizzo di librerie esterne
- Esecuzione replicabile: è necessario salvare eventuali seed e parametri da usare
- Risoluzione dei problemi: effettuata mediante tests. Deve essere possibile eseguire un test per risolvere un problema.

## 1.3 Problema del commesso viaggiatore (TSP)

Il problema che andremo a risolvere attraverso il nostro algoritmo è quello del commesso viaggiatore. La descrizione può essere formalizzata come segue: un commesso viaggiatore necessita di visitare $n$ città, al massimo una volta, partendo da una città di partenza $A$ e tornando alla stessa dopo averle visitate tutte. Si chiede di trovare qual è il percorso ottimale (ossia il percorso più breve) che visiti tutte le città.

Il problema, di tipo NP-hard, viene espresso matematicamente nel modo seguente: dato un grafo pesato e completo $G = (V, E, w)$ con $n$ veritci, determinare il ciclo hamiltoniano con il costo più basso.
Data una matrice di incidenza $C$ (dove $c_{i,j} \in \{0, 1\}$), ed una matrice dei pesi $w$, la funzione obiettivo è la seguente:

$$\min \sum_{i=0}^{n} \sum_{j=0}^{n} c_{i,j} \cdot w_{i,j} \tag{1}$$

Dato che la computazione e verifica di tutte le $n!$ possibilità richiederebbe un tempo troppo elevato (ed il tempo per la risoluzione di un problema è limitato a 3 minuti), nel nostro progetto ci accontenteremo di una "buona" soluzione, utilizzando dei metodi euristici.

# 2 Svolgimento del progetto

## 2.1 I/O

Quale step iniziale e fondamentale è stato necessario sviluppare un parser dei problemi forniti, in quanto questi venivano forniti in un formato proprietario che sfrutta la struttura seguente:

```
NAME: ch130
TYPE: TSP
COMMENT: 130 city problem (Churritz)
DIMENSION: 130
EDGE_WEIGHT_TYPE: EUC_2D
BEST_KNOWN : 6110
NODE_COORD_SECTION
1 334.5909245845 161.7809319139
2 397.6446634067 262.8165330708
(...)
EOF
```

Al fine di sfruttare al meglio il tempo a disposizione per i runs ho deciso di spendere un po' più del mio tempo a sviluppare un parser che fosse efficiente e non facesse uso di Regular Expressions. Questo ha portato alla creazione della classe `TSPLoader`. Una volta caricato un problema (tramite il metodo `parseFile()`) viene restituita un'istanza della classe `TSPData`. Questa classe contiene i file di problema che verranno poi utilizzati in qualsiasi algoritmo di risoluzione.

# 3 Algoritmi Utilizzati

## 3.1 Genetic Algorithm

Per l'algoritmo genetico ho fatto uso di una funzione di crossover ampiamente discussa e rivisitata, ottimizzata per il TSP, chiamata EAX (Edge Assembly Crossover).

I cicli AB sono selezionati tramite una funzione `EAX.rand()` ed `EAX.heur()` creando un $E\text{-}Set$. Successivamente dall'$E\text{-}Set$ viene generato un'insieme $E_C$ nel modo seguente:

$$E_C := (E_A \cap \bar{D}) \cup (E_B \cap D) \tag{2}$$

dove $D$ è l'$E\text{-}Set$.

Alternativamente, ed in modo equivalente, si può definire $E_C$ nel modo seguente:

$$E_C := (E_A \setminus (E\text{-}set \cap E_A)) \cup (E\text{-}Set \cap E_B) \tag{3}$$