Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Tesina di Ricerca Operativa 2

Travelling Salesman Problem

Autori

Raffaele Di Nardo Di Maio 1204879

Cristina Fabris 1205722

Anno Accademico 2019-2020

Indice

Introduzione

L'intera tesina verterà sul Travelling Salesman Problem. Quest'ultimo si pone l'obiettivo di trovare un tour ottimo,ovvero di costo minimo, all'intero di un grafo orientato.
In questa trattazione verranno analizzate soluzioni algoritmiche per una sua variante, detta simmetrica, che viene applicata a un grafo completo non orientato.
Di seguito viene riportata la formulazione matematica di tale versione:

$$\begin{cases} min \sum\limits_{e \in E} c_e \, x_e \\ \\ \sum\limits_{e \in \delta(v)} x_e = 2 & \forall \, v \in V \\ \\ \sum\limits_{e \in E(S)} x_e \leq |S| - 1 & \forall \, S \underset{\neq}{\subset} V : |S| \geq 3 \end{cases}$$

Un'istanza di tale problema viene definita normalmente da un grafo, per cui ad ogni nodo viene associata un numero intero (Ex. $\Pi = \{1, 2, 3, .., n\}$).
I risolutori che verranno applicati al problema sono di due tipologie:

- **Risolutori esatti**
  basati sul Branch & Bound. I più conosciuti sono:
  - **IBM ILOG CPLEX**
    gratuito se utilizzato solo a livello accademico.
  - **XPRESS**
  - **Gurobi**
  - **CBC**
    l'unico Open-Source tra questi

- **Risolutori euristici (meta-euristici)**
  algoritmi che forniscono una soluzione approssimata.

Esempio di risolutori: Concorde [2]
William Cook [1]

Istanze del problema e soluzioni

2.1 Istanze

Le istanze del problema solitamente sono punti dello spazio 2D, che sono quindi definiti da due coordinate, x e y. Per generare istanza enormi del problema, si utilizza un approccio particolare in cui viene definito un insieme di punti a partire da un'immagine già esistente.
La vicinanzae dei punti generati dipende dalla scala di grigi all'interno dell'immagine (Ex. generazione di punti a partire dal dipinto della Gioconda[3]).
Le istanze che verranno utilizzate dai programmi, creati durante il corso, utilizzano il template **TSPlib**. Di seguito viene riportato il contenuto di un file di questa tipologia.

```
NAME : esempio
COMMENT : Grafo costituito da 5 nodi
TYPE : TSP
DIMENSION : 5
EDGE_WEIGHT_TYPE : ATT
NODE_COORD_SECTION
1 6734 1453
2 2233 10
3 5530 1424
4 401 841
5 3082 1644
EOF
```

Listing 2.1: esempio.tsp

Le parole chiave solitamente contenute in questi file 2.1 sono:

- **NAME**
  seguito dal nome dell'istanza TSPlib

- **COMMENT**
  seguito dal commento associato all'istanza

- **TYPE**
  seguito dalla tipologia dell'istanza

- **DIMENSION**
  seguito dal numero di nodi nel grafo (*num_nodi*)

- **EDGE_WEIGHT_TYPE**
  seguito dalla specifica del tipo di calcolo che viene effettuato per ricavare il costo del tour

- **NODE_COORD_SECTION**
  inizio della sezione composta di *num_nodi* righe in cui vengono riportate le caratteriste di ciascun nodo, nella forma seguente:
  indice_nodo x y

- **EOF**
  decreta la fine del file

2.2 Soluzioni

Una soluzione del problema è data da una sequenza di nodi, definita come una permutazione dell'istanza (Ex. $S = \{x_1, x_2, ..., x_n\}$ tale che $x_i = x_j \;\; x_i \in \Pi \; \forall \; x_i \in S \; \wedge \; x_i! = x_j \; \forall \; i \neq j$). Poichè in questa variante non esiste alcuna origine, ogni tour può essere descritto da più permutazioni, due per ogni nodo del grafo. Una volta definita la soluzione S, infatti, questa può essere percorsa in entrambi i versi e l'origine può essere uno qualsiasi dei nodi del grafo.

2.2.1 Gnuplot

Una volta ottenuta la soluzione al problema di ottimizzazione ne viene disegnato il grafo per facilitarne la comprensione. Per fare ciò viene utilizzato un programma di tipo command-driven detto Gnuplot.
Per poter utilizzare Gnuplot all'interno del proprio programma esistono due metodi:

- Collegarne la libreria ed invocare direttamente le sue funzioni all'interno del programma;

- Collegare l'eseguibile interattivo al proprio programma. In questo caso i comandi deve essere passati all'eseguibile attraverso un file di testo.

In questa trattazione è stato scelto il secondo metodo. All'interno del file è possibile specificare a Gnuplot le caratteristiche grafiche che deve aver il grafo (stile e spessore della linea, dimensione dei punti, se presenti delle etichette dove devono essere posizionate ecc). Di seguito è riportato un esempio di tale file.

```
set style line 1 \
    linecolor rgb '#0000ff' \
    linetype 1 linewidth 1 \
    pointtype 7 pointsize 0.5

plot 'solution.dat' with linespoints linestyle 1 title "Solution",
    '' using 1:2:(sprintf("%d", $3)) notitle with labels center
       offset 1
```

Listing 2.2: esempio.tsp

In questo caso "solution.dat" è il file che contiene le informazioni relative al grafico. In particolare sono presenti le coordinate dei punti, uno per riga, seguite dall'etichetta a loro associata.

$$x_i \quad y_i \quad label_i$$

La terza colonna può anche essere lasciata vuota, in quanto la presenza delle etichette non è obbligatoria (in quel caso deve essere modificato il file relativo allo stile rispetto all'esempio, eliminando questa specifica). Nel caso in cui i nodi siano descritti su righe consecutive una all'altra (senza righe vuote tra loro) Gnuplot graficherà un ramo che colleghi i nodi su due righe adiacenti. Se invece si vogliono disegnare rami singolo è sufficiente inserire una riga vuota ogni due nodi (gli estremi del ramo che si vuole rappresentare).

CPLEX

Per poter utilizzare gli algoritmi di risoluzione forniti da CPLEX è necessario costruire il modello del problema legato all'istanza sopra descritta.
CPLEX possiede due meccanismi di acquisizione del modello:

- modalità interattiva: in cui il modello viene letto da un file precedentemente generato (*model.lp*)
- definendo il modello attraverso le API del linguaggio C (o del linguaggio utilizzato per la scrittura del programma)

Per memorizzare tale modello CPLEX utilizza due strutture dati:

- ENV (enviroment): contiene i parametri necessari all'esecuzione
- LP: contiene i dati degli elementi del modello

Ad ogni ENV è possibile associare più LP, ma nel nostro caso ne sarà sufficiente uno solo.

Come prima cosa, per poter costruire il modello da analizzare, è necessario creare un puntatore alle due strutture dati necessarie a CPLEX.

```
1  int error;
2  CPXENVptr env = CPXopenCPLEX(&error);
3  CPXLPptr lp = CPXcreateprob(env, &error, "TSP");
```

Listing 3.1: modelTSP.txt

La funzione alla riga 2 alloca la memoria necessaria e riempie la struttura con valori di default. Nel caso in cui non termini con successo memorizza un codice d'errore in *&errore*. La funziona invocata nella riga successiva, invece, associa la struttura LP all'enviroment che gli viene fornito. "TSP" sarà il nome del modello creato.

Al termine di queste operazioni il modello, riempirlo è stata costruita la seguente funzione:

```
build_model(&instanza_problema, env, lp);
```

Viene aggiunta al modello una colonna alla volta con i costi dei vari archi, sfruttando

```
CPXnewcols(env, lp, num_colonne, vettore_costi,
 vettore_lower_bound, vettore_upper_bound, dato_binario,
 stringhe_nomi);
```

Questa funzione aggiunge *num_colonne* colonne con una sola invocazione, per far si che ne aggiunga una sola è necessario passargli l'indirizzo di *vettore_costi, vettore_lower_bound, vettore_upper_bound, dato_binario, stringhe_nomi* affinché li veda come array da un elemento e non come variabili.

Per poter inserire il primo vincolo del problema

$$\sum_{e \in \delta(v)} x_e = 2 \qquad\qquad \forall\, v \in V$$

viene sfruttata la funzione

```
CPXnewrows(env, lp, numero_righe, vettore_termini_noti,
  vettore_tipo_vincoli, NULL, stringhe_nomi);
```

Anche in questo caso è necessario seguire le stesse accortezze dell'analoga sopra descritta poiché inserisce *numero_righe* alla volta.
In questo modo si viene a creare una matrice in cui è presente il valore 1 se il nodo in questione appartiene al ramo nella colonna corrispondente, 0 altrimenti.
Per convenzione è stato deciso di indicare tutti i rami (i, j), con $i \neq j$, rispettando la proprietà $i < j$. Per tener conto di questo particolarità è necessario fare particolare attenzione nell'inserimento delle righe.

i \ j	0	1	2	3	4	5
0		0	1	2	3	4
1			5	6	7	8
2				9	10	11
3					12	13
4						14
5						

Figura 3.1

Bibliografia

[1] *http://www.math.uwaterloo.ca/tsp/*

[2] *http://www.math.uwaterloo.ca/tsp/concorde/index.html*

[3] *http://www.math.uwaterloo.ca/tsp/data/ml/monalisa.html*