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:

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

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 e soluzioni del problema

2.1 Istanze

Le istanze del problema, analizzate durante il corso, sono punti dello spazio 2D, identificati quindi da due coordinate (x,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 vicinanza dei punti generati dipende dalla scala di grigi all'interno dell'immagine (es. generazione di punti a partire dal dipinto della Gioconda[3]).
Le istanze che vengono elaborate 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 più importanti, contenute in questi file 2.1, sono:

- **NAME**
  seguito dal nome dell'istanza TSPlib
- **COMMENT**
  seguito da un 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   coordinata_x   coordinata_y
```

- **EOF**
  decreta la fine del file

2.2 Soluzioni

Una soluzione del problema è una sequenza di nodi che corrisponde ad una permutazione dell'istanza (es. $S = \{x_1, x_2, ..., x_n\}$ tale che $x_i = x_j \quad x_i \in \Pi \, \forall \, x_i \in S \, \wedge \, x_i! = x_j \, \forall \, i \neq j$). Poichè in questa variante non esiste alcuna origine, ogni tour può essere descritto da due versi di percorrenza e l'origine può essere un nodo qualsiasi del grafo.

2.2.1 Gnuplot

Una volta ottenuta la soluzione del problema di ottimizzazione, viene disegnato il grafo per facilitare all'utente la comprensione della sua correttezza. Per fare ciò viene utilizzato Gnuplot, un programma di tipo command-driven. Per poterlo utilizzare all'interno del proprio programma esistono due metodi:

- Collegare la libreria ed invocare le sue funzioni all'interno del nostro programma
- Collegare l'eseguibile interattivo al proprio programma. In questo caso i comandi deve essere passati all'eseguibile attraverso un file di testo e l'utilizzo di un pipe.

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. Di seguito viene riportato un esempio di tale file.

```
1  set style line 1 \
2      linecolor rgb '#0000ff' \
3      linetype 1 linewidth 1 \
4      pointtype 7 pointsize 0.5
5
6  plot 'solution.dat' with linespoints linestyle 1 title "Solution"
7       ,'' using 1:2:(sprintf("%d", $3)) notitle with labels center
8       offset 1
9
10 set term png
11 set output "solution.png"
12 replot
```

Listing 2.2: style.txt

Nell'esempio sopra riportato, nella prima parte viene definito lo stile, il colore delle linee e la tipologia di punti, che verrano in seguito visualizzati all'interno del grafico prodotto.
In seguito viene effettuato il plot del grafo in una finestra, utilizzando il primo e secondo valore di ciascuna riga del file **solution.dat** come coordinate mentre il terzo valore viene utilizzato come etichetta.

Il file **solution.dat** contiene le informazioni relative alla soluzione del grafico in cui ciascuna riga ha la seguente forma:

```
coordinata_x   coordinata_y   posizione_in_tour
```

coordinata_x rappresenta la coordinata x del nodo;
posizione_in_tour rappresenta la coordinata y del nodo;
posizione_in_tour rappresenta l'ordine del nodo all'interno del tour, assunto come nodo di origine il nodo 1.

Il grafico viene generato dal comando **plot**, leggendo tutte le righe non vuote e disegnando un punto nella posizione **(coordinata_x,coordinata_y)** del grafico 2D. In seguito viene tracciata una linea solo tra coppie di punti, legati a righe consecutive non vuote all'interno di **solution.dat**.

Attraverso le istruzioni riportate nelle righe 10-12 di **style.txt**, viene invece salvato il grafico appena generato nell'immagine **solution.png**.

Di seguito vengono riportate le varie fasi necessarie alla definizione di un pipe e al passaggio di questo al programma GNUplot:

- **Definizione del pipe**

```
1 FILE* pipe = _popen(GNUPLOT_EXE, "w");
```

  dove **GNUPLOT_EXE** è una stringa composta dal percorso completo dell'eseguibile di GNUplot, seguita dall'argomento **-persistent** (es. *"D:/Programs/GNUplot/bin/gnuplot -persistent"*).

- **Passaggio delle istruzioni a GNUplot**

```
2  f = fopen("style.txt", "r");
3
4  char line[180];
5  while (fgets(line, 180, f) != NULL)
6  {
7    fprintf(pipe, "%s ", line);
8  }
9
10 fclose(f);
```

  viene passata una riga alla volta, del file **style.txt**, a GNUplot mediante il pipe precedentemente creato.

- **Chiusura del pipe**

```
11 _pclose(pipe);
```

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, vedi Figura 3.1:

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

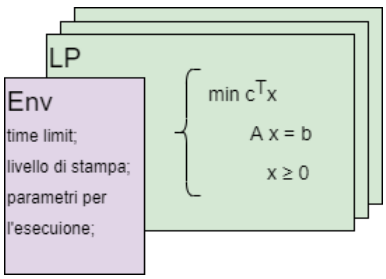

Figura 3.1: Strutture CPLEX

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'ENV che gli viene fornito. Il terzo parametro passato, nell'esempio "TSP", sarà il nome del modello creato.
Al termine di queste operazioni verrà quindi creato un modello vuoto. All'interno del nostro programma per riempirlo è stata costruita la seguente funzione:

```
void cplex_build_model(puntatore_istanza_problema, env , lv);
```

in cui:

puntatore_istanza_problema	è un puntatore alla struttura che contiene l'istanza del problema
env	di tipo CPXENVptr, è un puntatore alla struttura ENV precedentemente creata
lp	di tipo CPXLPptr, è un puntatore alla struttura LP precedentemente creata

Al suo interno viene aggiunta al modello una colonna alla volta con i costi dei diversi archi, sfruttando la seguente invocazione:

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

in cui:

env	di tipo CPXENVptr, è un puntatore alla struttura ENV precedentemente creata
lp	di tipo CPXLPptr, è un puntatore alla struttura LP precedentemente creata
num_colonne	numero di colonne che si vogliono inserire

vettore_costi	vettore con i costi degli archi da inserire
vettore_lower_bound	vettore contenente i lower bound delle variabili da inserire
vettore_upper_bound	vettore contenente gli upper bound delle variabili da inserire
dato_binario	vettore contenete la tipologia delle variabili da inserire, nel nostro caso binarie
stringhe_nomi	vettore di stringhe contenenti i nomi delle variabili da inserire

Questa funzione aggiunge *num_colonne* colonne con una sola invocazione, quindi tutti i parametri da lei richiesti devono essere puntatori ad array. Ogni elemento dell'array in posizione generica i deve contenere le informazioni richieste dalla funzione, relative alla colonna i-esima. Per far si che $CPXnewcols()$ aggiunga una sola riga, è necessario passargli come parametri array formati da un solo elemento, cioè il puntatore alle loro variabili.

Per poter inserire il primo vincolo del problema

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

viene invece sfruttata quest'altra funzione

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

in cui:

env	di tipo CPXENVptr, è un puntatore alla struttura ENV precedentemente creata
lp	di tipo CPXLPptr, è un puntatore alla struttura LP precedentemente creata
numero_righe	numero di righe che si vogliono inserire
vettore_termini_noti	vettore con i termini noti del vincoli

vettore_tipo_vincolo	vettore che specifica la tipologia dei vincoli da inserire
NULL	valore predefinito per il nostro utilizzo
stringhe_nomi	vettore di stringhe contenenti i nomi delle variabili da inserire

Anche in questo caso è necessario seguire le stesse accortezze dell'analoga sopra descritta poiché inserisce *numero_righe* alla volta.In questo modo è possibile inserire, una per volta, una riga per ogni variabile. Al termine di tutti gli inserimenti sarà stata creata una matrice in cui sarà 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. In Figura 3.2 è riportato lo schema degli indici associati all'inserimento.

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 3.2: Indici della matrice

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*