



Università degli Studi di Cagliari
Facoltà di Scienze
Corso di Laurea Magistrale in Informatica
A.A. 2021/22

Elaborato del corso di *Network Flows Optimization*

**Design and Implementation of a Tabu Search Algorithm
for a Vehicle Routing Problem in Python**

Studente

PISEDDU Enrico

matr. 60/73/65222

Docente del Corso

Prof. DI FRANCESCO Massimo

INDICE

STRUTTURA DELL'ELABORATO	pag. 3
1. VEHICLE ROUTING PROBLEM	pag. 4
1.1 Il Vehicle Routing Problem (VRP)	pag. 4
1.2 Complessità del VRP	pag. 5
1.3 Il Tabu Search	pag. 5
2. PROGETTO	pag. 6
2.1 Istanza da risolvere	pag. 6
2.2 Strategia TS utilizzata	pag. 7
2.2.1 Soluzione iniziale	pag. 7
2.2.2 Local Search	pag. 7
2.2.3 Uscita dall'ottimo locale	pag. 7
2.3 Schema dell'algoritmo sviluppato	pag. 8
2.4 Strumenti utilizzati	pag. 9
3. RISULTATI	pag. 10
3.1 Soluzione e risultati ottenuti	pag. 10
3.2 Confronto con la soluzione di CPLEX	pag. 11
CONCLUSIONI E SVILUPPI FUTURI	pag. 12
APPENDICE	pag. 13

STRUTTURA DELL'ELABORATO

Il presente elaborato è composto da tre capitoli:

- il primo capitolo descrive il Vehicle Routing Problem (VRP), ne fornisce una formulazione, ne esamina la complessità e introduce l'euristica del Tabu Search (TS) per la risoluzione, in generale, dei problemi NP-hard;
- il secondo capitolo illustra il progetto svolto, in particolare i dati della specifica istanza del VRP che si è risolta, la strategia TS che si è scelto di perseguire per la risoluzione e gli strumenti utilizzati per lo sviluppo dell'algoritmo;
- il terzo capitolo presenta la soluzione ottenuta dall'algoritmo TS sviluppato e in seguito, confronta tale soluzione con quella proposta dal *tool* di ottimizzazione CPLEX.

1. VEHICLE ROUTING PROBLEM

1.1 Il Vehicle Routing Problem

Il Vehicle Routing Problem (VRP) è un classico problema in ambito di ottimizzazione su rete. In tale problema si ha a disposizione un numero K di veicoli con capacità C , i quali devono servire n clienti interconnessi fra loro che richiedono, ognuno, una certa quantità di prodotto. È richiesto che ciascun veicolo inizi e termini la propria rotta in un punto speciale detto *depot*.

Tale problema è formalizzato nel seguente modo. Siano:

- $G(V, E)$ un grafo completo dotato di un insieme $V = \{0 = \text{depot}, 1, \dots, n\}$ di vertici (rappresentante ognuno un cliente) e di un insieme $A = \{ij \mid i, j \in V\}$ di archi che collegano due vertici i, j
- c_{ij} il costo associato all'arco ij
- d_i la domanda di prodotto richiesta dal cliente i -esimo
- K il numero di veicoli aventi ognuno la stessa capacità C
- la variabile booleana $y_{ik} = \begin{cases} 1 & \text{se il cliente } i \text{ è servito dal veicolo } k \\ 0 & \text{altrimenti} \end{cases}$
- la variabile decisionale booleana $x_{ij}^k = \begin{cases} 1 & \text{se l'arco } ij \text{ è percorso dal veicolo } k \\ 0 & \text{altrimenti} \end{cases}$

L'obiettivo del problema è trovare, per ogni veicolo, le rotte ottimali tali da minimizzare il costo totale di trasporto, rappresentato dalla funzione obiettivo:

$$\sum_{i \in V} \sum_{j \in V} c_{ij} \sum_{k=1}^K x_{ij}^k$$

soggetta ai seguenti vincoli tecnologici:

- | | |
|---|---|
| 1. $\sum_{k=1}^K y_{ik} = 1$ | $\forall i \in V - \{0\}$ |
| 2. $\sum_{k=1}^K y_{0k} = K$ | |
| 3. $\sum_{j \in V} x_{ij}^k = \sum_{j \in V} x_{ji}^k = y_{ik}$ | $\forall i \in V, k = 1, \dots, K$ |
| 4. $\sum_{i \in V} d_i y_{ik} \leq C$ | $\forall k = 1, \dots, K$ |
| 5. $\sum_{i \in S} \sum_{j \notin S} x_{ij}^k \geq y_{hk}$ | $\forall S \subseteq V - \{0\}, h \in S, k = 1, \dots, K$ |
| 6. $y_{ik} \in \{0, 1\}$ | $\forall i \in V, k = 1, \dots, K$ |
| 7. $x_{ij}^k \in \{0, 1\}$ | $\forall i, j \in V, k = 1, \dots, K$ |

1.2 Complessità del VRP

Il VRP è un problema *NP-hard* ovvero non esiste un algoritmo esatto capace di determinarne la soluzione ottimale in tempo polinomiale rispetto all'input.

Il metodo più banale per la risoluzione del VRP consiste nell'enumerare tutte le possibili soluzioni (le rotte) e prendere quella con costo minore tale da rispettare tutti i vincoli. Tuttavia, questa strada non è praticabile poiché su un'istanza di un TSP (addirittura semplificata rispetto ad un VRP) di soli 20 nodi, tutte le possibili rotte da calcolare e da verificarne l'ammissibilità sarebbero $20! \approx 2.4 \cdot 10^{19}$, un numero proibitivo.

Esistono tuttavia degli algoritmi esatti che riescono a generare e certificare l'ottimo globale e in un tempo ragionevole ma solo per VRPs con poche centinaia di nodi, e di conseguenza, il loro uso non potrebbe essere impiegato per la risoluzione di problemi logistici reali in quanto al giorno d'oggi potrebbe essere necessario trovare la soluzione di un VRP con molte migliaia di nodi in un tempo breve.

1.3 Il Tabu Search

Il Tabu Search (TS) è un metodo euristico che si propone di superare i limiti – in termini temporali – degli algoritmi esatti cercando di fornire una soluzione molto vicina all'ottimo globale in un tempo molto breve ai problemi combinatori.

L'idea base del TS è quella di partire da una soluzione iniziale ammissibile e applicare una Local Search (LS) al fine di trovare mediante mosse migliorative, a partire da essa, l'ottimo locale. Una volta raggiunto l'ottimo locale, il TS prevede mosse non migliorative che peggiorino la soluzione corrente, in modo da poter far partire una nuova LS. Per evitare che il metodo del TS entri in stallo ovvero che la nuova LS riproponga come ottimo locale quello generato al passo precedente, il TS tiene traccia delle ultime mosse al fine di rendere proibite quelle inverse (i.e. renderle *tabu*) per un certo numero di volte.

Il seguente pseudocodice riassume brevemente l'euristica del TS:

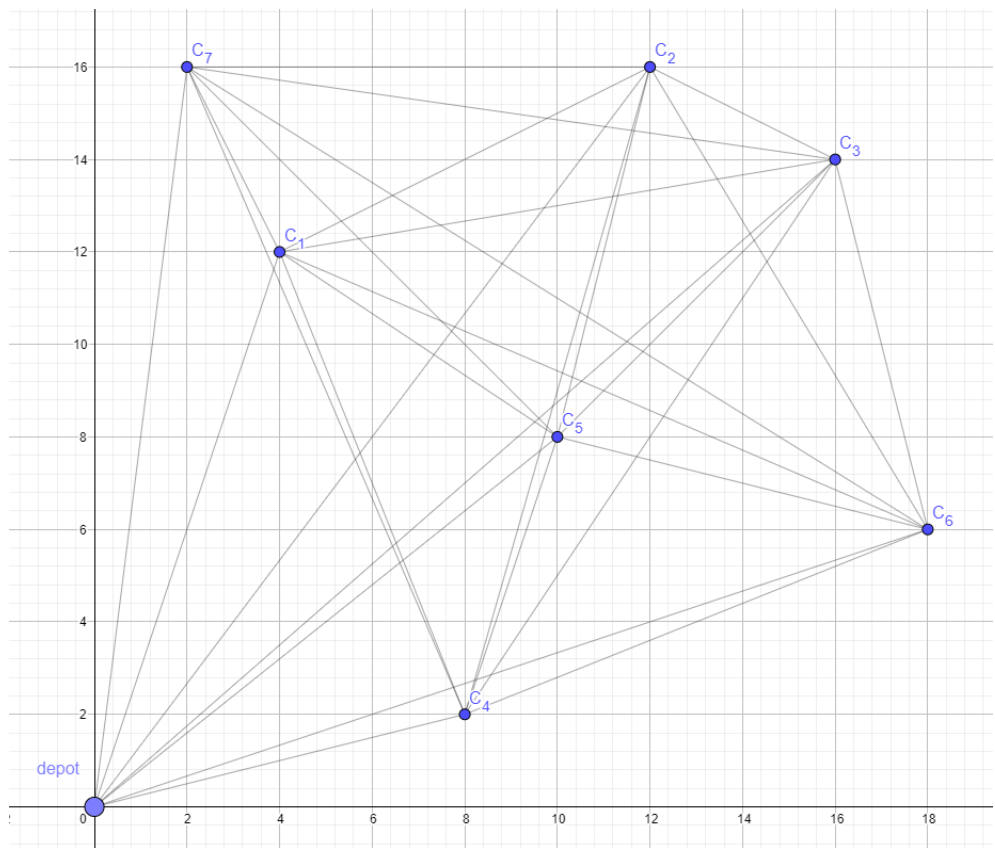
A Basic Template Algorithm for TS
1. Define an initial feasible solution \mathbf{x}'
2. Define the Tabu List $\mathbf{T} = \{\}$
3. Define the set of all visited solutions $\mathbf{S} = \{\}$
4. while (stop criteria is false):
4.1 $\mathbf{x} = \text{local_search}(\mathbf{x}', \mathbf{T})$
4.2 record the solution \mathbf{x} in \mathbf{S}
4.3 $\mathbf{z} = \text{not_improving_move}(\mathbf{x})$
4.4 update \mathbf{T} by adding or deleting some move
4.5 $\mathbf{x}' = \mathbf{z}$
5. return the minimum cost solution in \mathbf{S}

2. PROGETTO

2.1. Istanza da risolvere

È stato scelto di risolvere un'istanza di un VRP avente sette nodi, depot escluso, da servire con una flotta di due veicoli, ciascuno con capacità massima pari a 40. I nodi sono stati modellati come punti in uno spazio bidimensionale aventi ciascuno una coordinata (x, y) , col depot in posizione $(0,0)$, come illustrato in figura 1. In seguito, è stata calcolata la distanza euclidea per ogni coppia di punti.

Figura 1



Le domande di ciascun cliente e le distanze fra ogni coppia di clienti sono indicate nelle seguenti tabelle:

Cliente	Domanda
C ₁	7
C ₂	9
C ₃	15
C ₄	8
C ₅	6
C ₆	12
C ₇	9

	depot	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇
depot	0	12.65	20.00	21.26	8.25	12.81	18.97	16.12
C ₁	-	0	8.94	12.17	10.77	7.21	15.23	4.47
C ₂	-	-	0	4.47	14.56	8.24	11.66	10.00
C ₃	-	-	-	0	14.42	8.49	8.25	14.14
C ₄	-	-	-	-	0	6.32	10.77	15.23
C ₅	-	-	-	-	-	0	8.25	11.31
C ₆	-	-	-	-	-	-	0	18.87
C ₇	-	-	-	-	-	-	-	0

2.2. Strategia TS utilizzata

Al fine di comprendere al meglio la strategia TS utilizzata, nei successivi tre sottoparagrafi si farà ricorso alle prime iterazioni dall'algoritmo sviluppato a partire dalla soluzione iniziale ammissibile.

2.2.1. Soluzione iniziale

Prima di applicare la strategia del TS, è stata generata una soluzione ammissibile da cui far partire l'intero metodo. Tale soluzione iniziale è la seguente e ha costo 137.24:

Rotta del Veicolo 1: $0 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 6 \rightarrow 0$
Rotta del Veicolo 2: $0 \rightarrow 3 \rightarrow 5 \rightarrow 1 \rightarrow 0$

Si noti come le rotte siano ammissibili poiché:

- entrambe partono dal depot e terminano il viaggio nel depot (i.e. non esistono *subtours*)
- la capacità di carico di ciascun veicolo (40) è rispettata poiché il primo veicolo carica 38/40 (al cliente 2 consegna 9, al cliente 4 consegna 8, al cliente 7 consegna 9 e al cliente 6 consegna 12) e il secondo veicolo carica 28/40
- ogni cliente è visitato da un solo veicolo una sola volta

2.2.2. Local Search (LS)

Per la ricerca dell'ottimo locale è stata sviluppata una mossa di *first improvement*.

Date due rotte R_1 ed R_2 , rispettivamente percorse dal veicolo 1 e 2, descritte sotto forma di sequenze di nodi visitati, la strategia sviluppata del *first improvement* consiste nel trovare il primo scambio di nodi fra le due rotte tali che il costo totale sia minore, tale che i vincoli di capacità siano rispettati e tale che la mossa scelta non sia *tabu* (si veda il sottoparagrafo 2.2.3)

A titolo d'esempio, si considerino le rotte iniziali descritte nel paragrafo 2.2.1 con costo iniziale 137.24. L'algoritmo per la ricerca dell'ottimo locale cerca di effettuare un primo scambio fra il nodo 2 e il nodo 3: si ottiene una nuova soluzione con costo minore (136.86) ma non ammissibile poiché la capacità del primo veicolo eccede i 40. Al successivo passo si cerca di scambiare il nodo 2 col nodo 5: il nuovo costo diventa 119.53 e la soluzione è ammissibile poiché il primo veicolo trasporta 35/40 e il secondo veicolo 31/40. La LS ha quindi prodotto un ottimo locale con costo 119.53 definito dalle nuove rotte: $R_1 = \{0 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 6 \rightarrow 0\}$ e $R_2 = \{0 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 0\}$. Se questo scambio non avesse prodotto una soluzione ammissibile con costo minore, l'algoritmo avrebbe cercato di scambiare il nodo 2 con il nodo 1, il nodo 4 col 3, poi con il 5 e l'uno, e così via.

2.2.3. Uscita dall'ottimo locale

Una volta ottenuto un ottimo locale, l'algoritmo effettua una mossa non migliorativa al fine di ottenere una nuova soluzione (ammissibile e con costo maggiore) e cercare di esplorare, nella successiva iterazione, altre aree dello spazio delle soluzioni. Tale mossa "peggiorativa" è simmetrica

rispetto a quella del *first improvement*: si cerca infatti il primo scambio di nodi fra le due rotte che peggiori il costo della soluzione corrente e se lo scambio viene effettuato si memorizza nella *tabu list* la mossa inversa. L'inserimento della mossa inversa nella lista tabu consente alla successiva iterazione della LS di proibire che l'algoritmo entri in stallo riproponendo come soluzione ottima locale quella trovata al passo precedente. La *tabu list* è stata implementata come coda di lunghezza 1.

Si iterano i passi della LS e di uscita dall'ottimo locale – con relativa memorizzazione delle mosse *tabu* – per un numero fissato di volte, da calibrare in funzione della complessità del problema (numero di nodi e/o numero di veicoli disponibili).

2.3 Schema dell'algoritmo sviluppato

Di seguito è riportato lo pseudocodice dell'algoritmo implementato:

A Tabu Search Algorithm for the VRP (2 vehicles)	
1.	Generate a set of $n-1$ 2D points and the depot at $(0,0)$;
2.	Generate a demand for each point (except depot);
3.	Compute Euclidean distances for each pair of points;
4.	Define the tabu set $T = \{\}$; //it contains the tabu moves
5.	Define the set of visited solutions $S = \{\}$;
6.	Define an initial feasible solution \mathbf{x}' for 2 vehicles with capacity K and add this solution to the set S ;
7.	Define the constant <i>max_iterations</i> and let $i = 0$;
8.	while ($i < \text{max_iterations}$):
9.	Try to swap 2 nodes in \mathbf{x}' (if this move is not in T) and stop when you obtain a <u>better</u> and feasible solution called <i>local_opt_sol</i> ;
10.	Add <i>local_opt_sol</i> to the set S ;
11.	Try to swap 2 nodes and stop when you obtain a <u>worst</u> and feasible solution called <i>worst_sol</i> ;
12.	Add the reverse swap move done in step 11 to the set T and delete the old entry in T
13.	$\mathbf{x}' = \text{worst_sol}$; $i = i + 1$;
14.	return the minimum cost solution in S ;

2.4 Strumenti utilizzati

Per lo sviluppo dell'algoritmo TS precedentemente discusso, sono stati utilizzati:

1. Il linguaggio Python
2. L'IDE PyCharm
3. La libreria `numpy` per la modellazione e gestione di vettori e matrici
4. Le librerie `copy`, `matplotlib` e `time` rispettivamente per la gestione del passaggio di parametri per valore e riferimento, per la generazione di grafici e per la misurazione del tempo di esecuzione dell'algoritmo.

3. RISULTATI

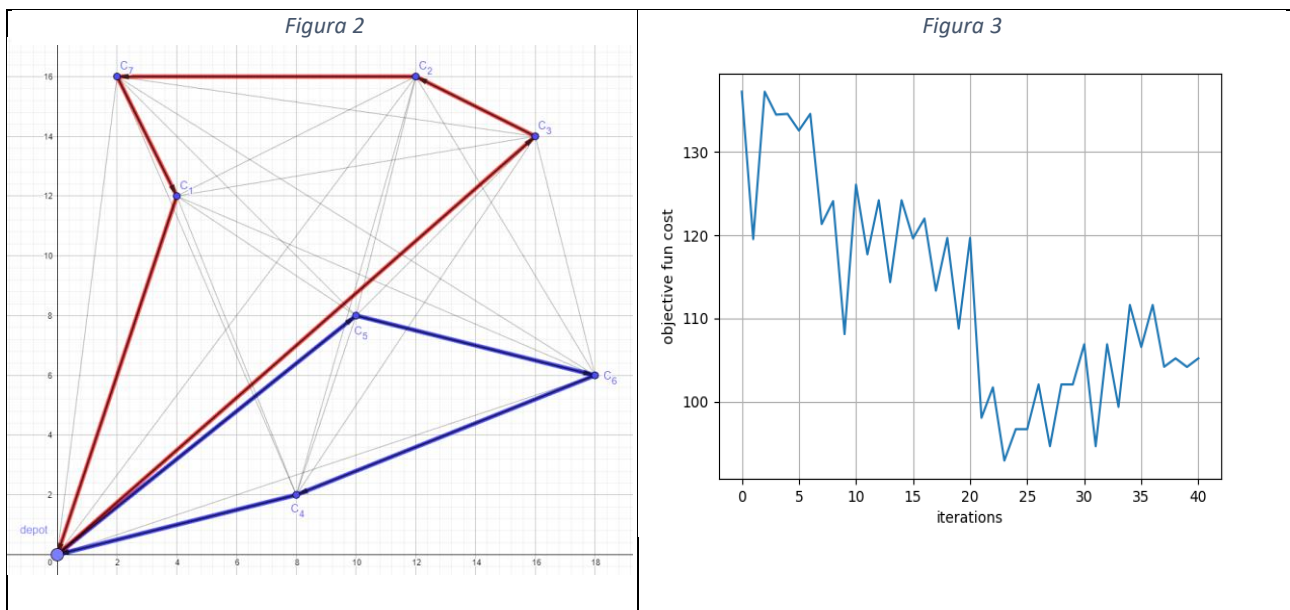
3.1. Soluzione e risultati ottenuti

L'algoritmo sviluppato nel capitolo precedente è stato utilizzato per la risoluzione dell'istanza del VRP descritta nel paragrafo 2.1 e ha prodotto la seguente soluzione in un tempo pari a 0.1 secondi:

Rotta del Veicolo 1:	$0 \rightarrow 3 \rightarrow 2 \rightarrow 7 \rightarrow 1 \rightarrow 0$
Rotta del Veicolo 2:	$0 \rightarrow 5 \rightarrow 6 \rightarrow 4 \rightarrow 0$

con costo 92.92. Si noti che tale soluzione è ammissibile poiché non presenta *subtour* e i vincoli di capacità sono rispettati se si confrontano i nodi visitati da ciascuna rotta con la domanda di ciascun nodo presente nella tabella del paragrafo 2.1: in figura 2 il veicolo indicato dalla tratta ROSSA soddisfa i clienti 3, 2, 7 e 1 consegnando loro rispettivamente $7+9+15+9=40/40$, mentre il veicolo 2 indicato dalla tratta BLU soddisfa i clienti 5, 6 e 4 consegnando rispettivamente $8+6+12=26/40$.

In figura 3 è mostrato un grafico che consente di visualizzare l'andamento del costo della funzione obiettivo per ogni iterazione dell'algoritmo TS sviluppato. All'iterazione 0 è presente il costo della soluzione iniziale, in seguito, dall'iterazione 1 in poi sono presenti i costi delle soluzioni prodotte dalla LS (iterazioni dispari) alternate alle soluzioni che consentono l'uscita dall'ottimo locale precedente (iterazioni pari). Da tale grafico si evince come la soluzione di costo minimo sia stata prodotta dalla LS all'iterazione 23.



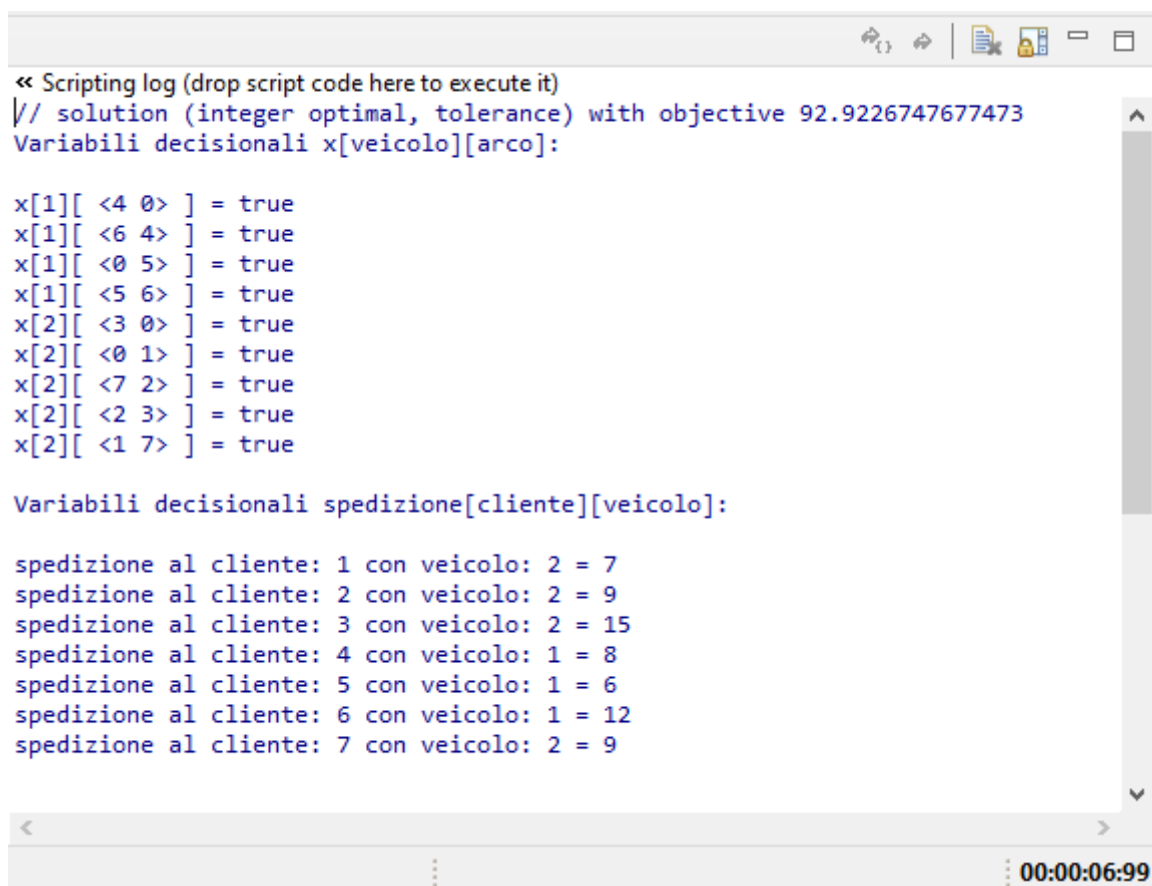
3.2. Confronto con la soluzione di CPLEX

Dopo aver sviluppato il progetto e visto la soluzione dell'algoritmo TS proposto, è stato scelto di risolvere la stessa istanza del VRP con il *tool* di ottimizzazione CPLEX al fine di confrontarne i risultati.

Per la risoluzione esatta dell'istanza in CPLEX è stata necessaria l'implementazione dei vincoli descritti nel paragrafo 1.1 e in particolare l'enumerazione di tutti i possibili vincoli di *subtour elimination*. Per la scrittura di questi ultimi vincoli è stato generato uno *script* in Python che ha prodotto in maniera automatizzata tutti i possibili vincoli di subtour che coinvolgono tutti i nodi escluso il depot.

È interessante notare che la soluzione proposta da CPLEX (in figura 4) è identica a quella proposta dall'algoritmo TS sviluppato, sia in termini di rotte che in termini di costo. La differenza sostanziale si è invece notata dal punto di vista computazionale: l'algoritmo TS sviluppato ha prodotto la soluzione ottima in 0.1 secondi, mentre CPLEX l'ha proposta in circa 7 secondi.

Figura 4



```
<< Scripting log (drop script code here to execute it)
// solution (integer optimal, tolerance) with objective 92.9226747677473
Variabili decisionali x[veicolo][arco]:

x[1][ <4 0> ] = true
x[1][ <6 4> ] = true
x[1][ <0 5> ] = true
x[1][ <5 6> ] = true
x[2][ <3 0> ] = true
x[2][ <0 1> ] = true
x[2][ <7 2> ] = true
x[2][ <2 3> ] = true
x[2][ <1 7> ] = true

Variabili decisionali spedizione[cliente][veicolo]:

spedizione al cliente: 1 con veicolo: 2 = 7
spedizione al cliente: 2 con veicolo: 2 = 9
spedizione al cliente: 3 con veicolo: 2 = 15
spedizione al cliente: 4 con veicolo: 1 = 8
spedizione al cliente: 5 con veicolo: 1 = 6
spedizione al cliente: 6 con veicolo: 1 = 12
spedizione al cliente: 7 con veicolo: 2 = 9
```

CONCLUSIONI E SVILUPPI FUTURI

In questo elaborato è stato visto dal punto di vista teorico il problema e la complessità del VRP, è stato poi descritto e progettato un algoritmo TS per la risoluzione di una semplice istanza di VRP che coinvolge otto nodi e due veicoli. In seguito, è stata confrontata la soluzione ottenuta con quella proposta dal tool CPLEX.

È emerso che nonostante le soluzioni ottenute (sia in termini di rotte che di costo) siano identiche, esse sono state prodotte in tempi completamente differenti (e per soli 8 nodi), circa 0.1 secondi per l'algoritmo TS e 7 secondi per CPLEX: ciò dimostra l'efficacia dell'euristica del TS, la quale riesce a proporre in tempi molto brevi una soluzione molto vicina e talvolta coincidente all'ottimo globale per problemi *NP-hard*. Questa strategia è quindi da preferire per la risoluzione di tali problemi decisionali che richiedono una soluzione in tempi brevi: utilizzare infatti un metodo esatto avrebbe un tempo di esecuzione proibitivo.

Un possibile sviluppo futuro a questo lavoro è quello di ampliare e generalizzare tale strategia del TS, proponendo nuove mosse e strategie per la risoluzione di alcune istanze di *benchmark* al fine di valutare al meglio la risposta dell'algoritmo euristico.

APPENDICE

Riferimenti al codice

Tutto il codice necessario allo sviluppo del progetto è disponibile al seguente link:

https://github.com/enricopiseddu/nfo_project

La struttura del repository è la seguente:

https://github.com/enricopiseddu/nfo_project

```
|
|----> python_ts
|      |
|      |----> VRP_TS_algorithm.py
|----> cplex
|      |
|      |----> vrp.mod
|      |----> vrp.dat
|----> PISEDdu_ENRICO_elaborato_finale.pdf|.doc
```