

**Università degli Studi di Cagliari**

**Facoltà di Scienze**

**Corso di Laurea Magistrale in Informatica**

Elaborato del corso di *Network Flows Optimization*

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

Studente Docente del Corso

**PISEDDU Enrico DI FRANCESCO Massimo**

matr. 60/73/65222

**INDICE**

**INTRODUZIONE pag. 3**

1. **VEHICLE ROUTING PROBLEM pag. 4**
   1. Il Vehicle Routing Problem (VRP) pag. 4
   2. Complessità del VRP pag. 5
   3. Il Tabu Search pag. 5
2. **PROGETTO pag. 7**
   1. Istanza da risolvere pag. 7
   2. Strategia TS utilizzata pag. 8
      1. Soluzione iniziale pag. 8
      2. Local Search pag. 8
      3. Uscita dall’ottimo locale pag. 8
   3. Schema dell’algoritmo sviluppato pag. 9
   4. Strumenti utilizzati pag. 10
3. **RISULTATI pag. 11**
   1. Risultati ottenuti pag. 11
   2. Confronto con CPLEX pag. 12

**CONCLUSIONI E SVILUPPI FUTURI pag. 13**

**INTRODUZIONE**

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 il metodo del Tabu Search (TS) per la risoluzione, in generale, dei problemi NP-hard.
* il secondo capitolo illustra 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. **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 *m* di veicoli con capacità *k,* 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*.

Siano:

* un grafo completo dotato di un insieme di vertici (rappresentante ognuno un cliente), e di un insieme di coppie di archi che collegano due vertici
* il costo associato all’arco
* la domanda di prodotto richiesta dal cliente i-esimo
* il numero di veicoli aventi ognuno la stessa capacità
* la quantità di prodotto che il veicolo spedisce al cliente
* la variabile decisionale booleana

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

soggetta ai seguenti vincoli tecnologici:



Il primo vincolo impone che ogni cliente debba essere servito esattamente una volta.

Il secondo vincolo impone che per ogni veicolo v e per ogni vertice p, ad ogni arco entrante corrisponda un arco uscente (vincoli di conservazione del flusso).

Il terzo vincolo impone, per ogni veicolo, l’assenza di *subtours* (i.e. cicli che non coinvolgono il depot).

Il quarto vincolo assicura che ogni veicolo possa trasportare al più una certa quantità.

Gli ultimi due vincoli impongono rispettivamente che le variabili decisionali possano assumere solo valori booleani e che la quantità spedita dal veicolo v al cliente i sia non negativa

* 1. **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 semplice 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! ≈ 2.4 1019, un numero proibitivo.

Esistono tuttavia degli algoritmi esatti che riescono a generare e certificare l’ottimo globale, ma anch’essi sono dotati di un limite superiore di 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. **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 inziale 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*).

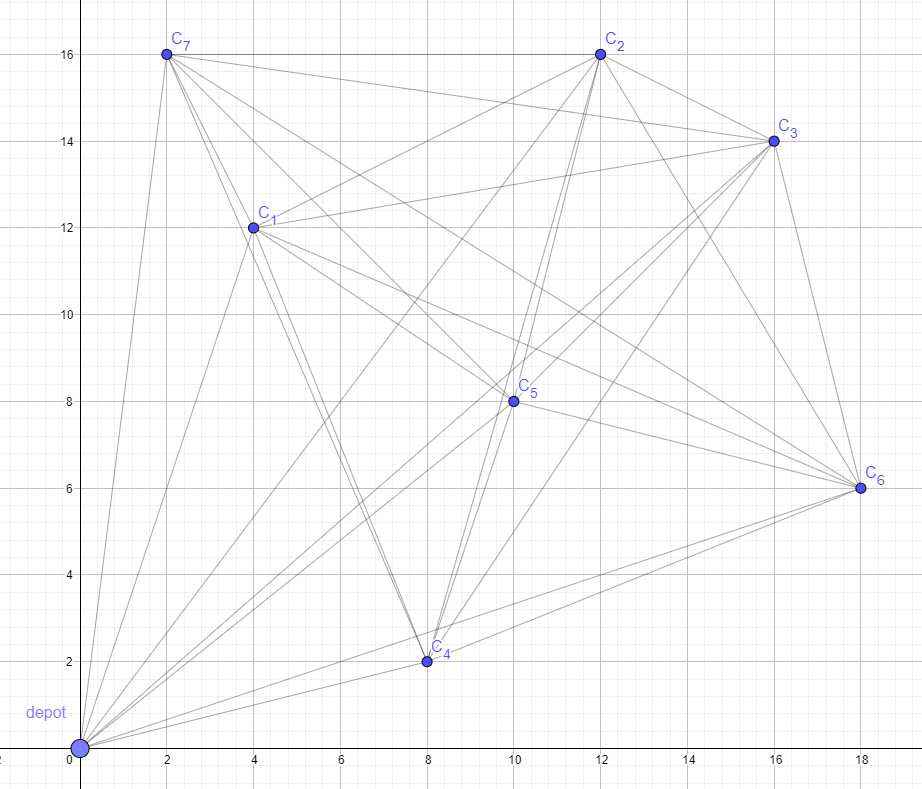
Il seguente pseudocodice riassume brevemente l’euristica del TS:

|  |
| --- |
| **A Basic Template Algorithm for TS** |
| 1. Define an initial feasible solution **x’** 2. Define Tabu List **T = {}** 3. Define the set of all visited solutions **S = {}** 4. **while** (stop criteria is false):   4.1 x = local\_search(x’, T)  4.2 record the solution x in S  4.3 z = not\_improving\_move(x)  4.4 update T by adding or deleting some move  4.5 x’ = z   1. **return** the minimum cost solution in S |

1. **PROGETTO**
   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



|  |  |
| --- | --- |
| **Cliente** | **Domanda** |
| C1 | 7 |
| C2 | 9 |
| C3 | 15 |
| C4 | 8 |
| C5 | 6 |
| C6 | 12 |
| C7 | 9 |

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **depot** | **C1** | **C2** | **C3** | **C4** | **C5** | **C6** | **C7** |
| **depot** | 0 | 12.65 | 20.00 | 21.26 | 8.25 | 12.81 | 18.97 | 16.12 |
| **C1** | - | 0 | 8.94 | 12.17 | 10.77 | 7.21 | 15.23 | 4.47 |
| **C2** | - | - | 0 | 4.47 | 14.56 | 8.24 | 11.66 | 10.00 |
| **C3** | - | - | - | 0 | 14.42 | 8.49 | 8.25 | 14.14 |
| **C4** | - | - | - | - | 0 | 6.32 | 10.77 | 15.23 |
| **C5** | - | - | - | - | - | 0 | 8.25 | 11.31 |
| **C6** | - | - | - | - | - | - | 0 | 18.87 |
| **C7** | - | - | - | - | - | - | - | 0 |

* 1. **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.

* + 1. Soluzione iniziale

Prima di applicare la strategia del TS, è stata calcolata 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 🡪 2🡪 4 🡪 7 🡪 6 🡪 0** |
| Rotta del Veicolo 2: **0 🡪 3 🡪 5 🡪 1 🡪 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
  + 1. Local Search (LS)

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

Date due rotte ed , 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: = {0🡪5🡪4🡪7🡪6🡪0} e = {0🡪3🡪2🡪1🡪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.

* + 1. 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 promo scambio di nodi fra le due rotte che peggiori la 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.

* 1. **Schema dell’algoritmo sviluppato**

|  |
| --- |
| **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 = {};  5. Define the set of visited solutions S = {};  6. Define an initial feasible solution **x’** for 2 vehicles with  capacity K and add this solution to the set S;  7. Define the constant *max\_iterations* and let i = 0;  8. **while** (i < *max\_iterations*):  9. Try to swap 2 nodes in x’ (if this move is not in T) and  stop when you obtain a better and feasible solution  called *local\_opt\_sol*;  10. Record *local\_opt\_sol* to the set S;  11. Try to swap 2 nodes and stop when you obtain a worst  and feasible solution called *worst\_sol*;  12. Add the reverse swap move done in step 11 to the set T and  delete the old entry in T    13. x’ = *worst\_sol*; i = i + 1;  14. **return** the minimum cost solution in S; |

* 1. **Strumenti utilizzati**

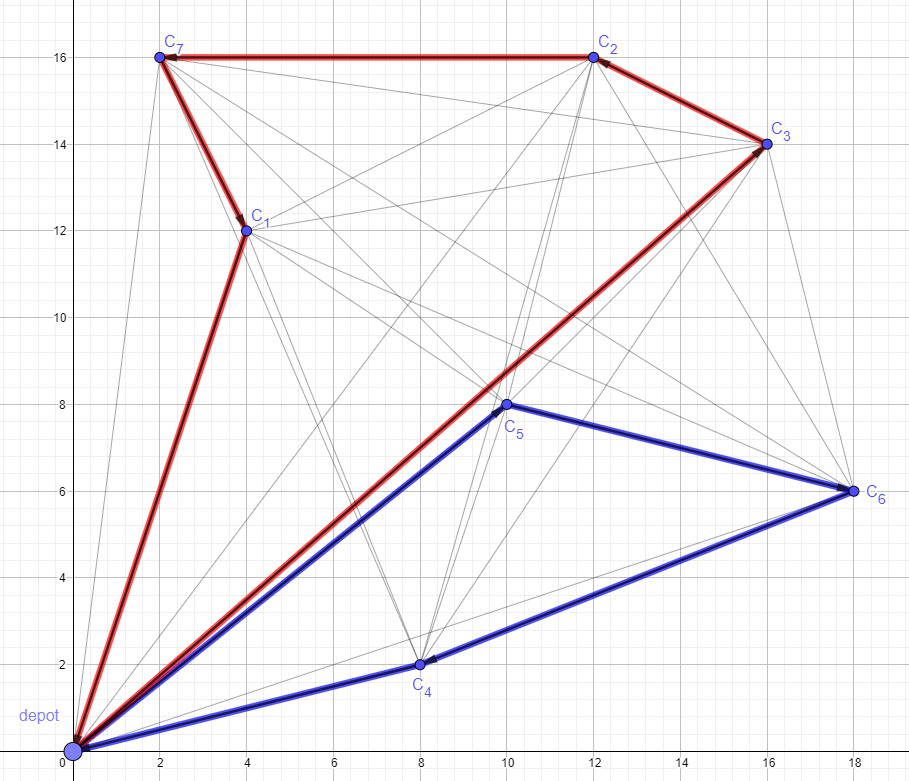
Per lo sviluppo dell’algoritmo TS precedentemente discusso, sono stati utilizzati:

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, 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.
5. **RISULTATI**
   1. **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:

|  |
| --- |
| Rotta del Veicolo 1: 0 🡪 3 🡪 2 🡪 7 🡪 1 🡪 0 |
| Rotta del Veicolo 2: 0 🡪 5 🡪 6 🡪 4 🡪 0 |

con costo 92.92. Si noti che la soluzione è ammissibile poiché non presenta *subtour*



* 1. **Confronto con CPLEX**

**CONCLUSIONI E SVILUPPI FUTURI**