UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Ricerca Operativa

Documentazione Progetto Ricerca Operativa

Anno Accademico 2024/2025

Professore: **Prof. Maurizio Boccia**
Luigi Guerrera M63001386

Contents

Chapter 1

Vehicle Routing Problem

Il Vehicle Routing Problem (VRP) consiste nel determinare una serie di percorsi per una flotta di veicoli, che partendo da uno o più depositi, devono visitare un insieme di clienti distribuiti geograficamente. L'obiettivo è minimizzare una funzione di costo totale, che può considerare la distanza percorsa, il tempo totale di viaggio o i costi operativi dei veicoli. La soluzione ottimale del VRP è vincolata dal rispetto dei limiti di capacità dei veicoli e delle tempistiche di servizio per ogni cliente.

Il Vehicle Routing Problem with Time Windows (VRPTW) è una variante del VRP che introduce un ulteriore livello di complessità: le finestre temporali per la consegna. Il VRPTW è un problema **NP-hard**. Ogni cliente deve essere servito entro un intervallo di tempo specifico, noto come finestra temporale, definito da un tempo di inizio e un tempo di fine. Se un veicolo arriva in anticipo, è costretto a aspettare fino al momento di apertura della finestra temporale; se arriva in ritardo, la consegna non può essere effettuata, rendendo la soluzione non valida.

Il VRPTW può essere formalmente definito come segue:

- **Insieme dei clienti** C={C1,C2,...,Cn}: ogni cliente Ci è caratterizzato da una domanda d_i, una finestra temporale $[e_i, l_i]$(dove e_i ed l_i rappresentano, rispettivamente, l'inizio e la fine della finestra temporale), e un tempo di servizio s_i.
- **Deposito iniziale**: rappresenta il punto di partenza e di arrivo dei veicoli. Ogni veicolo parte dal deposito e, una volta completato il proprio percorso, vi fa ritorno.
- **Veicoli** V={v1,v2,...,vm}: una flotta di veicoli, ciascuno con una capacità massima Q. Ogni veicolo deve seguire un percorso che rispetti la capacità e le finestre temporali dei clienti visitati.
- **Obiettivo**: minimizzare la distanza percorsa totale della flotta, che può essere espressa come:

$$\min_{v \in V} \sum_{(i,j) \in C} \sum c_{ij} x_{ij}^{v}$$

dove:

- V rappresenta l'insieme dei veicoli disponibili,
- C rappresenta l'insieme dei clienti,
- c_{ij} è il costo di viaggio tra i nodi i e j,
- x_{ij}^v è una variabile binaria che vale 1 se il veicolo v percorre l'arco tra i e j, altrimenti vale 0.

Vincoli del Problema Il VRPTW include numerosi vincoli, che garantiscono la validità delle soluzioni:

1. **Vincolo di Capacità**: ogni veicolo può trasportare al massimo Q unità, quindi per ogni percorso vvv la somma delle domande dei clienti serviti deve essere inferiore o uguale alla capacità del veicolo.
2. **Vincolo di Finestra Temporale**: ogni cliente deve essere visitato entro la propria finestra temporale. Se un veicolo arriva in anticipo, può attendere fino all'apertura della finestra temporale; tuttavia, non può arrivare in ritardo.
3. **Vincolo di Unicità della Visita**: ciascun cliente deve essere visitato esattamente una volta.
4. **Vincolo di Continuità del Percorso**: ogni veicolo deve partire e tornare al deposito, senza interrompere il percorso.

Funzione Obiettivo e Penalizzazioni La funzione obiettivo del VRPTW è solitamente una combinazione della minimizzazione della distanza percorsa e del numero di veicoli utilizzati.

La funzione obiettivo ha un ruolo fondamentale nell'algoritmo e in base a come questa è implementata e a come sono configurati i suoi parametri i valori del risultato cambiano favorendo ad esempio soluzioni con più o meno percorsi, che danno più o meno peso alla violazione dei vincoli o che danno più o meno peso alla distanza complessiva percorsa (ossia il valore della f1)

In questa particolare implementazione si considera un grafo completamente connesso come in figura:

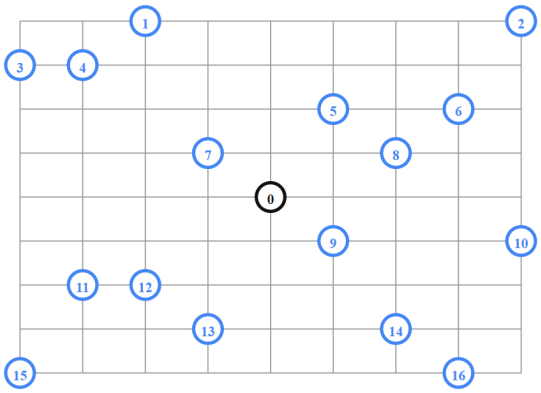

Figure 1.1: Grafo

In patricolare si usa la distanza euclidea in quanto il grafo è completamente connesso e si percorrono i segmenti passanti tra i due nodi come presentato nella seguente figura che rappresenta una soluzione al problema della immagine precedente. Invece la distanza tra ogni coppia di nodi vale: $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

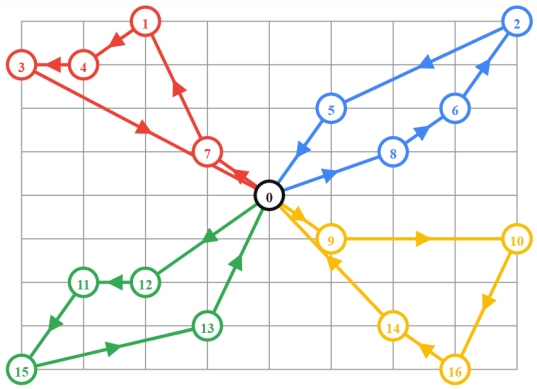

Figure 1.2: Soluzione Grafo

Chapter 2

Tabu Search

La Tabu Search è una tecnica di ottimizzazione che può trovare soluzioni ottimali migliori rispetto a molte delle procedure classiche di ottimizzazione, infatti questo approccio viene utilizzato per evitare di rimanere bloccati in minimi locali . Il primo passo consiste nel trovare una soluzione iniziale ammissibile, poi si applica una procedura iterativa che ha lo scopo di ridurre i costi di percorrenza dei percorsi. Il numero di iterazioni è controllato implicitamente impostando un limite di tempo per l'esecuzione dell'algoritmo o direttamente fissando un numero massimo di iterazioni.

Nella Tabu Search, due elementi fondamentali sono la tabu list e l'aspiration criterion:

• La **tabu list** è un elenco che contiene le mosse tabu, ovvero le variazioni della soluzione corrente che portano ad una soluzione già visitata. Ogni volta che generiamo dei nuovi vicini possiamo facilmente controllare se i nuovi percorsi corrispondono ad una soluzione già visitata. La tabu list non è infinita ma ogni volta che raggiunge la sua lunghezza massima (tabu_size) la lista viene aggiorna in modalità FIFO, salvando quindi solo le mosse più recenti e perdendo la memoria delle mosse tabu più vecchie .

• L'**aspiration criterion** viene usato per determinare l'accettazione di una soluzione. Infatti data una soluzione vicina viene accettata se non viola la tabu list e la differenza nei costi di viaggio tra la soluzione corrente e quella vicina è maggiore di 0 oppure se è associata al minimo costo di viaggio trovato finora anche se viola la tabu list.

Un altro aspetto fondamentale della Tabu Search è la ricerca dei vicini. Questo viene effettuato tramite la funzione **generate_neighborhood** che, sulla base della soluzione corrente (costituita dall'insieme di percorsi dell'ultima iterazione), genera una lista di mosse ammissibili.

2.1 Soluzione Iniziale

La soluzione iniziale è data da un algoritmo dei risparmi modificato per aggiungere un controllo sulle collisioni temporali:

- Inizializza i cluster (ogni cliente è inizialmente un singolo cluster)
- Calcola i risparmi (cioè la distanza percorsa da un veicolo che si risparmia unendo delle route separate)
- Ordina i risparmi in ordine decrescente
- Esegue i merge dei cluster basati sui risparmi
- Verifica che i clienti non siano già nella stessa rotta, che il merge rispetti la capacità, e che non ci siano sovrapposizioni temporali nella nuova route e in quel caso unisce i cluster.

2.2 Funzioni obiettivo e Penalizzazioni

La funzione obiettivo del VRPTW è solitamente una combinazione della minimizzazione della distanza percorsa e del numero di veicoli utilizzati. Esistono inoltre varianti della funzione obiettivo che introducono penalizzazioni per:

- **Attesa in caso di arrivo anticipato**: poiché i veicoli non possono servire il cliente fino all'apertura della finestra temporale, è possibile prevedere costi aggiuntivi per i tempi di attesa.
- **Violazione delle finestre temporali**: in alcuni casi, le penalizzazioni possono essere applicate qualora un veicolo arrivi leggermente in ritardo, anche se ciò comporta soluzioni di qualità inferiore.

Nell' implementazione iniziale sono state implementate una funzione f1, che calcola la somma di ogni distanza percorsa da un veicolo, ed una f2 che somma alla distanza totale della soluzione anche un fattore legato alle penalità. In particolare sono state valutate 2 f2 diverse e relative prestazioni, in particolare una prima f2 che considera sia le attese sia le violazioni delle finsetre temporali, ed un'altra implementazione che considera solo il numero di mancate consegne per via di vincoli temporali o di capacità che come vediamo restituisce risultati migliori.

2.3 Mosse

La funzione generate_neighborhood scorre tutte le mosse possibili e per ogni possibile mossa, dopo aver verificato se ammissibile, la aggiunge alla lista del vicinato. Le mosse considerate in questa implementazione sono le seguenti:

- Scambi di Clienti

- Inserimenti
- Rimozioni
- Inversioni

2.3.1 scambio

Data la mossa ("scambio", i, j, Cliente1.cust_no, k, l, Cliente2.cust_no) il programma esegue uno scambio tra il Cliente1, contenuto inizialmente all'interno di soluzione[i][j] (percorso i-esimo della soluzione corrente in posizione j), e Cliente2, contenuto inizialmente all'interno di soluzione[k][l] (percorso k-esimo della soluzione corrente in posizione l). A termine di tale mossa avremo Cliente1 in soluzione[k][l] e Cliente2 in soluzione[i][j].

2.3.2 inversione

L'inversione può essere vista come un caso particolare di swap in cui si scambiano 2 elementi consecutivi di una stessa route, necessario in quanto non sono previsti scambi all'interno di una stessa rotta. In particolare la mossa ("inversione", i, j, Cliente1.cust_no, 0, 0, Cliente2.cust_no) sostituisce l'elemento soluzione[i][j] con l'elemento soluzione[i][j-1].

2.3.3 rimozione

Questa mossa è fondamentale in quanto permette di avere un numero di routes diverso da quello di partenza. E' particolarmente importante se si usa come soluzione iniziale una soluzione che non bada ai vincoli temporali quale l'algoritmo dei risparmi classico che bada solo al vincolo della capacità di un veicolo. A seguito di una mossa ("rimozione", i, j, Cliente1, 0, 0, 0) il cliente in soluzione[i][j] viene rimosso dal percorso i-esimo e viene inserito in una nuova route creata ad-hoc contenente il solo Cliente1. Questa permette quindi di considerare soluzioni che prevedono più veicoli di quelli considerati in partenza.

2.3.4 inserimento

L'inserimento è la mossa opposta alla rimozione che nel caso specifico di inserimento da percorsi con un solo cliente equivale al merge effettuato durante la soluzione iniziale. Questa mossa permette quindi di ridurre il numero di veicoli sfruttati nella soluzione. ("inserimento", i, j, Cliente1.cust_no, k, l, Cliente2.cust_no) inserisco il nodo in posizione j all'interno del percorso i, nel percorso k dopo l'elemento in posizione l.

2.4 Tabu list

E' stata scelta una memoria di 10 elementi che sono nello stesso formato di una mossa ossia (tipo_mossa, percorso1_index, cliente1_index, cliente1_id, percorso2_index, cliente2_index,

cliente2_id). Ad ogni iterazione se questa coda circolare FIFO è piena viene rimosso l'ultimo elemento ed inserito la nuova mossa in testa. Il controllo is_tabu controlla che non vi siano all'interno della coda altre mosse che agiscono sui clienti già toccati nelle ultime 10 mosse (per ogni mossa verifica che non ci siano cliene1 or cliente2 come clienti interessati).

2.5 Criterio di arresto

L'implementazione di TabuSearch in questione termina l'esecuzione una volta raggiunte le 50*n iterazioni, dove n è il numero di clienti del problema, oppure all'occorrenza di 50 iterazioni consecutive senza update della soluzione.

Chapter 3

Risultati

Qui sono presentati i risultati ottenuti con l'implementazione di TabuSearch adattato per il VRPTW sono i seguenti, ogni soluzione è valutata sulla base delle funzioni obiettivo f1 e f2.

La $f2(sol) = f1(sol) + alpha * penality$, dove l' alpha considerato per il calcolo di f2 nell'output è 1 e la penalità considera tutti i tempi di attesa se il veicolo arriva in anticipo, i tempi di spostamento se si è in ritardo e viene mancata la consegna e anche lo sforamento della capacità di un veicolo.

Si valuta la penality pari al numero di consegne mancate moltiplicate per il massimo della matrice distance_matrix (max_distance), considerando anche il numero di veicoli come fattore di penalità moltiplicandolo per max_distance/2 e moltiplicando il suddetto fattore moltiplicativo del numero di veicoli per alpha, in modo da avere un valore maggiormente adattivo che permette di dare per alpha molto grande o molto piccolo un peso quanto più equivalente possibile rispetto alla violazione dei vincoli di capacità e temporali, si ottengono i seguenti risultati finali (la soluzione iniziale è quella di clarke e whrite modificato):

```
R101 25
Numero di veicoli stimato: 6
NV: 7
Funzione obiettivo f2 (distanza + penalita'): 1826.61
Funzione obiettivo f1 (distanza): 611.06
0 violazioni di capacita' e 14 violazioni temporali
NV: 14
Funzione obiettivo f2 (distanza + penalita'): 1724.69
Funzione obiettivo f1 (distanza): 821.71
0 violazioni di capacita' e 6 violazioni temporali

R101 50
Numero di veicoli stimato: 8
NV: 14
```

```
Funzione obiettivo f2 (distanza + penalita'): 3938.2
Funzione obiettivo f1 (distanza): 1142.63
0 violazioni di capacita' e 30 violazioni temporali
NV: 32
Funzione obiettivo f2 (distanza + penalita'): 3552.14
Funzione obiettivo f1 (distanza): 1841.41
0 violazioni di capacita' e 8 violazioni temporali

R101 100
Numero di veicoli stimato: 16
NV: 27
Funzione obiettivo f2 (distanza + penalita'): 8587.01
Funzione obiettivo f1 (distanza): 1929.34
0 violazioni di capacita' e 59 violazioni temporali
NV: 65
Funzione obiettivo f2 (distanza + penalita'): 7876.33
Funzione obiettivo f1 (distanza): 3422.58
0 violazioni di capacita' e 16 violazioni temporali

R102 25
Numero di veicoli stimato: 12
NV: 6
Funzione obiettivo f2 (distanza + penalita'): 1443.01
Funzione obiettivo f1 (distanza): 609.49
0 violazioni di capacita' e 9 violazioni temporali
NV: 11
Funzione obiettivo f2 (distanza + penalita'): 1430.79
Funzione obiettivo f1 (distanza): 701.46
0 violazioni di capacita' e 5 violazioni temporali

R102 50
Numero di veicoli stimato: 20
NV: 11
Funzione obiettivo f2 (distanza + penalita'): 3885.54
Funzione obiettivo f1 (distanza): 1089.97
0 violazioni di capacita' e 28 violazioni temporali
NV: 26
Funzione obiettivo f2 (distanza + penalita'): 3286.79
Funzione obiettivo f1 (distanza): 1617.79
0 violazioni di capacita' e 7 violazioni temporali

R102 100
Numero di veicoli stimato: 40
NV: 22
Funzione obiettivo f2 (distanza + penalita'): 7026.51
Funzione obiettivo f1 (distanza): 1792.2
0 violazioni di capacita' e 46 violazioni temporali
NV: 45
Funzione obiettivo f2 (distanza + penalita'): 6567.91
```

```
Funzione obiettivo f1 (distanza): 2573.3
0 violazioni di capacita' e 21 violazioni temporali
```

I risultati migliori ottenuti per suddetti benchmarks Solomon attraverso algoritmi per soluzioni ottimali sono i seguenti:

```
R101.25
NV:8
distanza: 617.1

R101.50
NV:12
distanza: 1044.0

R101.100
NV:20
distanza: 1637.7

R102.25
NV:7
distanza: 547.1

R102.50
NV:11
distanza: 909

R102.100
NV:18
distanza: 1466.6
```

Come possiamo vedere dai risultati con la prima soluzione non siamo molto lontani dai suddetti valori sebbene con lievi variazioni sul numero di veicoli e sulla distanza complessiva ma tale risultato rivela delle differenze sostanziali nel numero di vincoli violati rispetto alla soluzione ottimale. Con alcune modifiche abbiamo visto che si possono ottenere un numero di consegne mancate molto inferiore rendendo di fatto la soluzione accettabile sebbene con valori di f1 solitamente maggiori rispetto a quelli ottimali.

Supponendo che la soluzione ottimale non violi nessun vincolo e considerando le violazioni dei risultati ottenuti accetabili, ha senso valutare la bontà della soluzione in particolar modo nel caso di soluzioni con un numero di violazioni temporali inferiore quindi prendendo gli ultimi risultati presentati:

$$\text{R101.25}: \ gap = \frac{|OPT(I) - EUR(I)|}{|OPT(I)|} * 100 = \frac{|617 - 821|}{617} * 100 = 33$$

$$\text{R101.50}: \ gap = \frac{|OPT(I) - EUR(I)|}{|OPT(I)|} * 100 = \frac{|1044 - 1841.41|}{1044} * 100 = 76$$

R101.100 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|1637.7-3422.58|}{1637.7} * 100 = 108.9$

R102.25 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|547.1-701.46|}{547.1} * 100 = 28.1$

R102.50 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|909-1617.79|}{909} * 100 = 77.9$

R102.100 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|1466.6-2573.3|}{1466.6} * 100 = 75.4$

3.1 Effetto della soluzione iniziale

Si potrebbe anche pensare di cambiare la soluzione iniziale e valutarne gli effetti.

Con la soluzione iniziale banale, ossia con un numero di veicoli pari al numero di clienti, e dando un peso maggiore alla penalità legata al numero di veicoli otteniamo soluzioni migliori ma con ancora delle violazioni di vincoli. Da ciò si capisce quanto sia importante tarare questi fattori in base al problema reale ed in base a quali siano le circostanze preferibili in un caso reale.

Per cui si decide di tentare con la nuova soluzione iniziale e una nuova f2 che, solo in caso di violazioni (ossia mancate consegne) nella soluzione, aggiunge una penalità per la "perdita di reputazione" posta pari a 10*max_distance*alpha come presentato di seguito:

```
def f2(routes, distance_matrix, alpha, beta, vehicle_capacity):

...

if violazioni_capacita+violazioni_temporali==0:
   return total_distance+(alpha*max_distance*NV)
elif alpha>=1:
   return total_distance+alpha*max_distance*(violazioni_capacita+
      violazioni_temporali)+(alpha*max_distance*NV) + alpha*
      max_distance*50
else:
   return total_distance+alpha*max_distance*(violazioni_capacita+
      violazioni_temporali)+(alpha*max_distance*NV) + max_distance*50
```

Con queste modifiche a soluzione iniziale e f2 abbiamo i seguenti risultati:

```
R101.25 :
NV: 25
Funzione obiettivo f2 (distanza + penalita'): 2982.68
Funzione obiettivo f1 (distanza): 1246.18
0 violazioni di capacita' e 0 violazioni temporali
NV: 13
Funzione obiettivo f2 (distanza + penalita'): 1683.53
Funzione obiettivo f1 (distanza): 780.55
0 violazioni di capacita' e 0 violazioni temporali

R101.50 :
NV: 50
Funzione obiettivo f2 (distanza + penalita'): 6797.28
Funzione obiettivo f1 (distanza): 2624.78
0 violazioni di capacita' e 0 violazioni temporali
NV: 18
```

```
Funzione obiettivo f2 (distanza + penalita'): 2787.61
Funzione obiettivo f1 (distanza): 1285.51
0 violazioni di capacita' e 0 violazioni temporali

R101.100 :
NV: 100
Funzione obiettivo f2 (distanza + penalita'): 14172.44
Funzione obiettivo f1 (distanza): 4989.44
0 violazioni di capacita' e 0 violazioni temporali
NV: 31
Funzione obiettivo f2 (distanza + penalita'): 4899.86
Funzione obiettivo f1 (distanza): 2053.13
0 violazioni di capacita' e 0 violazioni temporali

R102.25 :
NV: 25
Funzione obiettivo f2 (distanza + penalita'): 2982.68
Funzione obiettivo f1 (distanza): 1246.18
0 violazioni di capacita' e 0 violazioni temporali
NV: 9
Funzione obiettivo f2 (distanza + penalita'): 1280.3
Funzione obiettivo f1 (distanza): 655.16
0 violazioni di capacita' e 0 violazioni temporali

R102.50 :
NV: 50
Funzione obiettivo f2 (distanza + penalita'): 6797.28
Funzione obiettivo f1 (distanza): 2624.78
0 violazioni di capacita' e 0 violazioni temporali
NV: 16
Funzione obiettivo f2 (distanza + penalita'): 2461.23
Funzione obiettivo f1 (distanza): 1126.03
0 violazioni di capacita' e 0 violazioni temporali

R102.100 :
NV: 100
Funzione obiettivo f2 (distanza + penalita'): 14172.44
Funzione obiettivo f1 (distanza): 4989.44
0 violazioni di capacita' e 0 violazioni temporali
NV: 28
Funzione obiettivo f2 (distanza + penalita'): 4501.16
Funzione obiettivo f1 (distanza): 1929.92
0 violazioni di capacita' e 0 violazioni temporali
```

Da notare che rispetto alle soluzioni ottime si osserva un utilizzo di un numero di veicoli pari ad 1,5 volte quello ottimale.

Stime dell'errore

Si può stimare l'errore dei risultati come segue:

R101.25 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|617-780.55|}{617} * 100 = 26.5$

R101.50 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|1044-1285.51|}{1044} * 100 = 23.13$

R101.100 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|1637.7-2053.13|}{1637.7} * 100 = 25.36$

R102.25 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|547.1-655.16|}{547.1} * 100 = 19.75$

R102.50 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|909-1126.03|}{909} * 100 = 23.87$

R102.100 : $gap = \frac{|OPT(I)-EUR(I)|}{|OPT(I)|} * 100 = \frac{|1466.6-1929.92|}{1466.6} * 100 = 31.59$