UNIVERSITÀ DEGLI STUDI
DI GENOVA

Corso di laurea in Ingegneria Informatica

Sviluppo di un algoritmo per un problema di vehicle routing multi-trip con pickup e delivery, finestre temporali e soste opzionali applicato al trasporto urbano di passeggeri

Relatori

Prof. Massimo Paolucci

Dott. Davide Anghinolfi

Dott. Roberto Ronco

Candidato

Mirko GUALDUCCI

14 Giugno 2022

Sommario

Questa tesi affronta una variante del Vehicle Routing Problem (VRP) che coinvolge un servizio di pickup and delivery in un contesto urbano, dove si utilizzano delle navette che possono essere prenotate in un arco temporale della giornata per viaggi specificando il punto di partenza e quello di arrivo desiderati. Questi veicoli, con una capacità massima che esprime il numero di persone che sono in grado di trasportare, possono percorrere più rotte nell'arco della stessa giornata da un deposito ad un altro, con possibili soste temporanee durante il tragitto in un insieme di stazioni di parcheggio.

Il contesto principale del problema affrontato riguarda il trasporto di passeggeri in una grande città metropolitana, al fine di poter servire le richieste di trasporto dei clienti, soddisfacendone il più possibile i requisiti temporali e minimizzando la distanza totale percorsa dai mezzi. Questo servizio potrà essere offerto da un sistema che raccoglierà una lista di prenotazioni effettuate il giorno precedente e che assegnerà gli utenti in differenti mezzi. I percorsi non saranno fissi, ma dipenderanno dalle richieste effettuate dai clienti, raccolte in precedenza. L'obiettivo che si persegue nell'affrontare questo problema è quello di soddisfare le richieste di tutti i clienti tramite la riduzione di percorrenza totale e il rispetto delle finestre temporali di prenotazione.

Il problema considerato include caratteristiche condivise con diverse classi di problemi VRP trattati in letteratura. Tuttavia, tale problema risulta originale considerando in modo congiunto tali aspetti, quali tra gli altri, la presenta di finestre temporali, il pickup e delivery, la presenza di più depositi, la possibilità di effettuare viaggi multipli, la presenza di stazioni di sosta in cui i viaggi possono essere conclusi e iniziati.

In questa tesi, viene implementata una versione specifica della metaeuristica Adaptive Large Neighborhood Search (ALNS). Il primo passo dell'algoritmo proposto in questa tesi consiste nel determinare una soluzione iniziale corrispondente a una lista di rotte da effettuare per soddisfare le richieste dei clienti. Successivamente la ALNS utilizza delle tecniche di distruzione e riparazione della soluzione nel tentativo di migliorare la soluzione corrente attraverso un'esplorazione dello spazio delle soluzioni. Tale esplorazione viene terminata dopo un numero massimo

di iterazioni senza miglioramento, restituendo la miglior soluzione trovata.

Infine si può osservare come l'approccio sviluppato per questo problema risulti di tipo trasversale, poiché può essere applicato a diversi contesti in cui è necessario determinare il *routing* di veicoli, come nel caso di problemi derivanti da altre realtà logistiche.

Ringraziamenti

Il primo ringraziamento va al professore Massimo Paolucci, relatore della tesi, grandissima fonte di ispirazione. Un altro ringraziamento nei suoi confronti è per aver creduto in me sin dal primo momento che ci siamo conosciuti e per la sua grande disponibilità nei miei confronti. Porterò per sempre nel mio bagaglio personale tutta la conoscenza appresa in questo periodo di preparazione.

Ringrazio Davide, correlatore della tesi, per avermi dato la possibilità di lavorare al suo fianco. In lui ho trovato grande supporto per la conoscenza dell'ambito di studio affrontato e anche nella metodologia di lavoro.

Ringrazio Roberto, correlatore della tesi, che ha avuto molta pazienza con me nella revisione totale dell'elaborato e per il suo impegno che mi ha portato al raggiugimento di questo lavoro in tempi rapidi, così da per poter concludere in anticipo il mio percorso universitario.

Un ringraziamento speciale va alla mia compagna di vita Elisa, che nei momenti migliori ed in quelli peggiori della mia vita mi è sempre stata affianco, spronandomi a dare il meglio di me stesso sotto ogni ambito. A lei va il merito del raggiungimento di questo obiettivo personale. In ogni momento ha saputo come aiutarmi, sacrificando molti momenti della nostra vita al fine di raggiungere questo obiettivo importante della mia vita.

Colgo l'occasione per ringraziare mia madre Antonella, che ha sempre saputo supportarmi negli studi, interessandosi al mio percorso in ogni momento ed essendo sempre disponibile nella vita di tutti i giorni.

Ringrazio anche mio fratello Mathias per avermi sempre supportato.

Un ringraziamento speciale va anche alla famiglia della mia fidanzata, sua madre Mariagrazia, suo padre Luciano, suo fratello Francesco e sua sorella Chiara per avermi sempre fatto sentire come un membro della loro famiglia.

Un ringraziamento importante va anche a Lady, che ha sempre portato tanto affetto e gioia nella mia vita anche con piccolissimi gesti.

Un altro ringraziamento va anche ai miei amici Marco, Erika, Loris, Francesco, Mattia, Chiara e Hailab con i loro esempi di vita, il loro supporto e la loro amicizia ho avuto una spalla in più che mi ha fatto arrivare a questo obiettivo.

Dedico questa tesi anche a me stesso, per i sacrifici fatti in questi anni che non mai avrei pensato di riuscire a sostenere e per essermi rimesso in gioco, riprendendo a studiare.

Vorrei dedicare principalmente questo lavoro a mia nonna Franca, che ha sempre tenuto al raggiungimento di questo obiettivo e oggi sarebbe fiera di quel che sono riuscito a portare a termine.

"Mirko Gualducci"

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

ALNS

Adaptive Large Neighborhood Search

ACVRP

Asymmetric Capacitated Vehicle Routing Problem

BPP

Bin Packing Problem

CCC

Capacity-Cut Constraints

CVRP

Capacitated Vehicle Routing Problem

DCVRP

Distance Capacitated Vehicle Routing Problem

DVRP

Distance-Constrained Vehicle Routing Problem

HGA

Hybrid Genetic Algorithm

HPSO

Hybrid Particel Swarm Optimization

LNS

Large Neighborhood Search

LP

Linear Programming

MIP

Mixed-Integer Programming

Multi-D

Multi-Depot

Multi-D&T

Multi-Depot and Multi-Trip

Multi-T

Multi-Trip

SA

Simulated Annealing

SCVRP

Symmetric Capacitated Vehicle Routing Problem

TSP

Traveling Salesman Problem

TSPPD

Traveling Salesman Problem with Pickup and Delivery

TSPTW

Traveling Salesman Problem with Time Windows

VLSN

Very Large Scale Neighborhood Search

VRP

Vehicle Routing Problem

VRPB

Vehicle Routing Problem with Backhauls

VRPPD

Vehicle Routing Problem with Pick-Up and Delivery

VRPPDTW

Vehicle Routing Problem with Pick-Up and Delivery and Time Window

VRPSPD

Vehicle Routing Problem with Simultaneous Pickup and Delivery

VRPTW

Vehicle Routing Problem with Time Window

VRPTW-R

Vehicle Routing Problem with Time Window and Release Dates

Capitolo 1

Introduzione

A causa della pandemia causata dalla malattia infettiva COVID-19 e del lockdown che ne è conseguito, l'anno 2020 ha costituito uno spartiacque nell'approccio al commercio elettronico (e-commerce) da parte di molte realtà aziendali italiane. Sebbene il trend di compra/vendita online fosse già osservabile negli anni precedenti, il 2022 ne ha contribuito significativamente alla crescita, sconvolgendo la vita di molte persone, cambiandone le abitudini di consumo, e portando così molte attività a ripensare il proprio modello di commercio.

Questo effetto ha avuto un grande impatto sulla logistica, trasformandola in un asset importante per il successo e la crescita delle aziende. I centri di distribuzione ed i magazzini sono stati portati ad avere un notevole incremento di pressione. La priorità è stata quella di soddisfare tutte le richieste del mercato, ottimizzando le risorse possedute e la minimizzandone i costi.

Come si può evincere da dati di ricerche effettuate [1], si è registrato sul mercato un calo di fatturato del -9,3%, ovvero di 77,8 mld di euro. A partire da giugno 2020 c'è stata una ripresa del +6% dei volumi rispetto allo stesso periodo del 2019. L'emergenza non ha fermato la possibilità di innovarsi, con la creazione di 501 startup logistiche (+57% rispetto al 2018) e raccogliendo 9,56 mld di dollari in finanziamenti (+92%).

In grandi e in piccoli centri urbani, il cliente ha sempre il desiderio di poter raggiungere la propria destinazione nel minor tempo possibile e con la minima spesa. Per questo, oltre alla possibilità di accedere a mezzi pubblici come autobus e treni, i clienti si affidano ai propri mezzi privati come auto e moto. Inoltre una buona parte di loro preferisce percorrere tratti di viaggio cittadini tramite l'utilizzo di mezzi a spesa temporale come taxi o veicoli smart.

Per questo nasce l'idea di poter trasportare i passeggeri in modalità flessibile, con un modello che si adatta alla domanda. Questo può essere effettuato tramite un sistema a chiamata, il quale selezionerà il numero di autobus da utilizzare in una giornata a seconda del numero di richiesta dei clienti. Utilizzando questa

modalità, i percorsi non saranno mai fissi, come avviene per gli autobus cittadini, ma dipenderanno dalle prenotazioni effettuate dai clienti. Con questo utilizzo si può affrontare un discorso di sostenibilità dove i mezzi vengono utilizzati solamente nei percorsi desiderati.

Questo servizio viene integrato nelle aree urbane suddividendo le linee di forza che tagliano la città in orizzontale e nelle valli. La possibilità di avere sistemi capillari e flessibili, consentono ai passeggeri di fare viaggi più brevi o di poter raggiungere le linee di forza per fare spostamenti più lunghi.

Viene affrontata la realizzazione di questa idea tramite l'uso di una app o di un portale web dal quale un utente può prenotare per il giorno successivo una corsa tramite una navetta. L'utente quindi determina il luogo di partenza e quello di arrivo con le determinate finestre temporali in cui preferisce essere prelevato e depositato. Ciascuna navetta potrà portare più di una persona insieme, non superando la propria capienza. Ogni prenotazione verrà aggiunta alla lista di richieste da soddisfare, le quali verranno elaborate dal sistema. A fine giornata il modello costruito produrrà una serie di percorsi da effettuare per il minor numero di mezzi possibili tra quelli disponibili. I veicoli partiranno da un deposito e ne raggiungeranno un altro a fine giornata, includendo delle soste in zone apposite per un periodo massimo di sosta di riposo obbligata. Si può vedere un esempio di schema del problema nella figura 1.1.

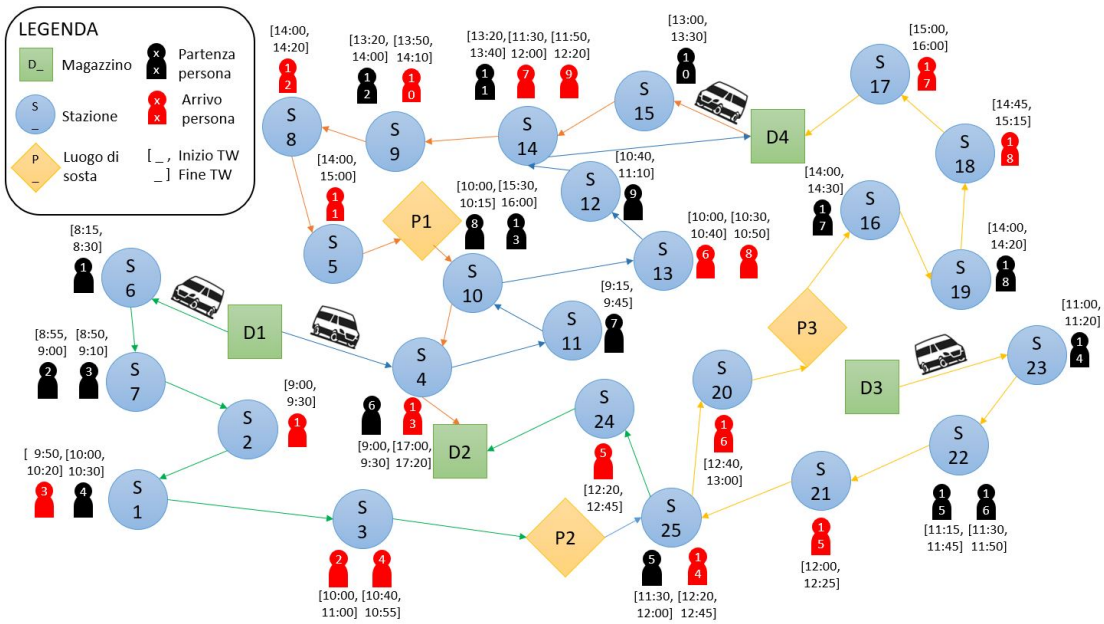

Figura 1.1: Schema del problema affrontato

In questa tesi, il quesito definito precedentemente viene approcciato come un problema di Vehicle Routing Problem con Pickup and Delivery di persone (VRPPD), capacità massima dei veicoli (CVRP), finestre temporali (VRPTW), multi-trip e multi-depot (Multi-D&T) con soste obbligate. Per quanto ne sappiamo, questo problema non è stato affrontato da nessuno e quindi non esiste lettertura indicata.

Inizialmente formuliamo il problema come un modello di programmazione intera mista (MIP), utilizzabile da un risolutore come CPLEX con modelli legati ad istanze su piccola scala. Viene inoltre sviluppata un algoritmo di ottimizzazione tramite Adaptive Large Neighborhood Search (ALNS) con l'utilizzo del criterio di accettazione mutuato dalla Simulated Annealing (SA). I risultati dimostrano che l'algoritmo sviluppato può risolvere con efficienza anche istanze a larga scala.

Il documento viene suddiviso come segue: il capitolo 2 esamina i problemi di VRP già esistenti ed utilizzati nel modello del nostro problema. Il capitolo 3 definisce il problema e lo formula con un modello matematico. Il capitolo 4 descrive le principali euristiche ed i principali metodi risolutivi utilizzati per risolvere i problemi a larga scala. Il capitolo 5 definisce tutte le specifiche del problema e il modo in cui sono state gestite. Il capitolo 6 propone i risultati quasi ottimali ottenuti su problemi di piccola e larga scala. Il capitolo 7 contiene le conclusioni derivanti dal problema trattato in questa tesi.

Capitolo 2

I problemi di vehicle routing: caratteristiche generali e stato dell'arte

Il primo problema di VRP è stato affrontato da Dantzig e Ramser nel 1959 [2], con lo scopo di trovare i percorsi ottimali per veicoli omogenei che servono un gruppo di stazioni di servizio soddisfacendo la domanda di petrolio richiesta da ciascuna stazione. Dopo cinque anni, Clarke e Wright [3] hanno generalizzato questo problema a un problema di ottimizzazione lineare che si presenta spesso nell'ambito della logistica e dei trasporti, ovvero come servire un insieme di clienti, geograficamente dispersi attorno a un deposito centrale, utilizzando un insieme di camion di differente portata.

Il classico VRP presuppone che ci sia un singolo deposito da cui partono e ritornano i veicoli e i clienti devono essere serviti una sola volta da ognuno di essi. In seguito sono state elaborate delle specializzazioni e delle varianti per il VRP [4]:

- *Capacitated VRP* ($CVRP$): i veicoli hanno una capacità di carico limitata per le merci che devono essere consegnate (vedi Sezione 2.1).

- *VRP with Pickup and Delivery* ($VRPPD$): gli ordini di trasporto corrispondono a prelievi da località di origine e consegne a località di destinazione che devono essere visitate in ordine in una unica rotta di ciascun veicolo (vedi Sezione 2.2).

- *VRP with Time Windows* ($VRPTW$): devono essere effettuate le consegne a dei clienti che hanno delle finestre temporali entro le quali i veicoli devono arrivare (vedi Sezione 2.3).

- *VRP with Backhauls* ($VRPB$): un veicolo effettua nella stessa rotta consegne che hanno origine nel deposito e ritiri che hanno destinazione nel deposito, servendo prima i clienti che richiedono consegne (linehauls) e quindi quelli che richiedono ritiri (backhauls).

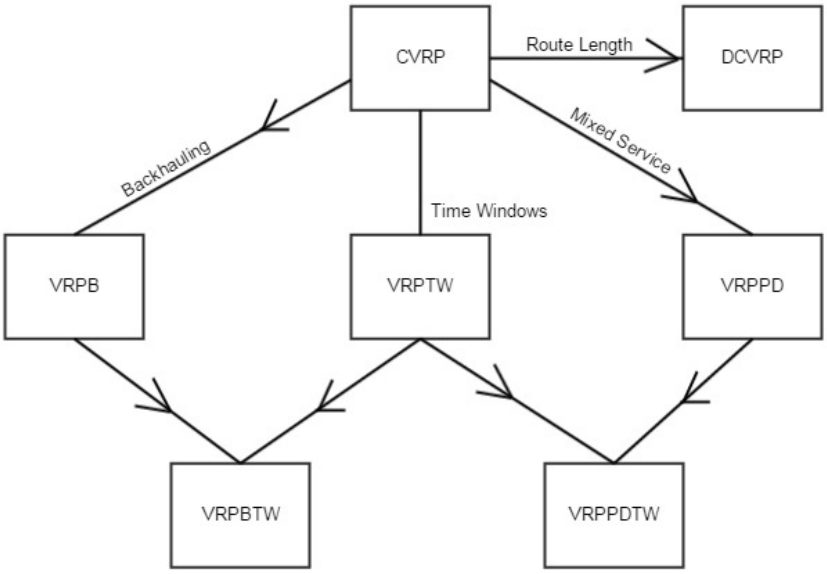

Figura 2.1: Mappa delle relazioni tra i più comuni problemi di VRP

Nei capitoli successivi vengono analizzati i vari sottoproblemi del VRP, i quali vengono descritti in "The Vehicle Routing Problem" da P. Toth e D. Vigo [5].

Inoltre sono stati affrontati altri due tipi di problemi per questa tesi, ovvero il Multi-D e Multi-T VRP con finestre temporali e pickup e delivery, racchiusi in parte nella pubblicazione "Multi-depot multi-trip vehicle routing problem with time windows and release dates" da L. Zhen, C. Ma, K. Wang, L. Xiao e W. Zhang [6] (vedi Sezione 2.4).

2.1 Capacited Vehicle Routing Problem

Questa sezione si concentra sulla versione base del VRP, il Capacited VRP (CVRP). Il CVRP corrisponde ad una estensione del problema di TSP (uno dei più noti problemi NP-hard), dove lo scopo è quello di trovare il tragitto più breve per collegare un numero dato di nodi considerando l'utilizzo di veicoli multipli con una capacità fissata.

Nel CVRP, tutti i clienti corrispondono a dei punti di consegna (o alternativamente a dei punti di ritiro), le richieste possono essere deterministiche, ovvero già

note in anticipo, e non frazionabili. Tutti i veicoli sono identici, hanno sede in un unico deposito centrale e vengono imposti solo limiti alla loro capacità. L'obiettivo è quello di minimizzare il costo totale (ad esempio tramite una funzione pesata dal numero di rotte e dalla loro lunghezza o dal tempo di percorrenza) per servire tutti i clienti.

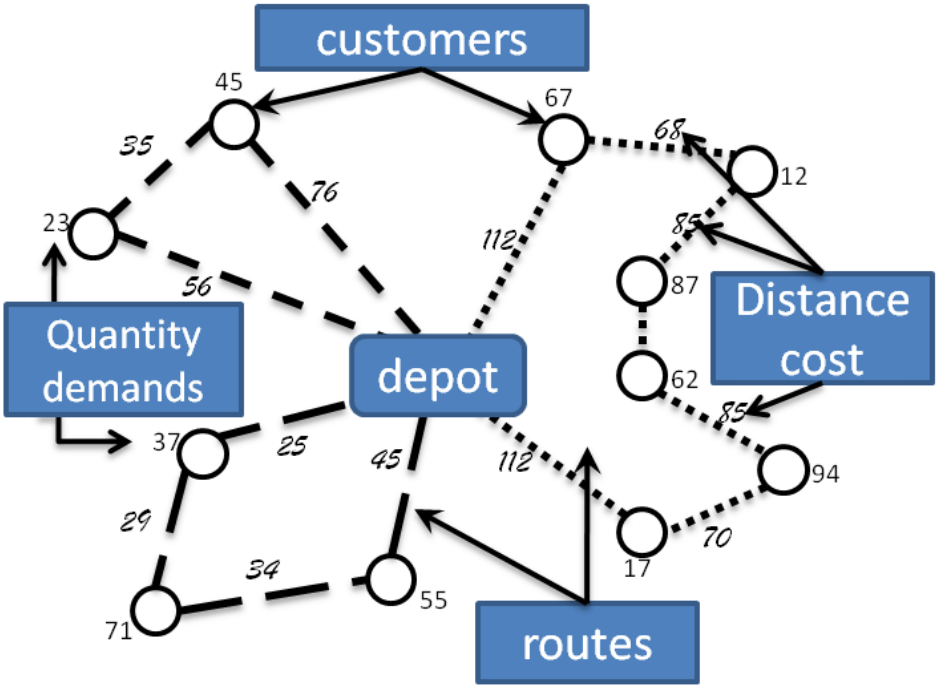

Figura 2.2: Esempio di problema di CVRP dove su ogni arco viene rappresentato il carico attuale sul veicolo e a ogni nodo il quantitativo di prodotto lasciato

Il problema di CVRP può essere descritto formalmente come segue. Sia $G = (V, A)$ un grafo non orientato completo, dove $V = \{0, \ldots, n\}$ è l'insieme dei vertici e A è l'insieme degli archi. I vertici $i = 1, \ldots, n$ corrispondono ai clienti, mentre il vertice 0 corrisponde al deposito. Ad ogni arco $(i, j) \in A$ è associato un costo non negativo c_{ij}, che rappresenta il costo di viaggio speso per andare dal vertice i al vertice j. Generalmente, si assume che archi del tipo (i, i) non possano appartenere ad alcuna rotta. Questo vincolo viene imposto definendo $c_{ii} = +\infty$ per tutti $i \in V$. Se G è un grafo orientato, la matrice dei costi c è asimmetrica e il problema corrispondente è chiamato CVRP asimmetrico (ACVRP). Altrimenti, abbiamo $c_{ij} = c_{ji}$ per tutti $(i, j) \in A$, il problema è chiamato CVRP simmetrico (SCVRP), e l'insieme di archi A è costituito da un insieme di archi non orientati, E. Dato un arco $e \in E$, siano $\alpha(e)$ e $\beta(e)$ i suoi vertici estremi. In seguito indicheremo l'insieme gli archi del grafo non orientato G con A quando gli archi saranno indicati

mediante i loro estremi (i, j), $i, j \in V$, e con E quando gli archi saranno indicati con un unico indice e.

Quando si considera un solo vertice $i \in V$, scriveremo $\delta(i)$ anziché $\delta(\{i\})$.

In diversi casi pratici, la matrice dei costi soddisfa la disuguaglianza triangolare definita come

$$c_{i,k} + c_{kj} \geq c_{ij} \qquad \forall\ i, j, k \in V \tag{2.1}$$

Quindi non conviene deviare dal collegamento diretto tra due vertici i e j. La presenza della disuguaglianza triangolare è talvolta richiesta dagli algoritmi di CVRP, e questa si può ottenere in modo semplice aggiungendo al costo di ogni arco una quantità positiva M opportunamente grande. Tuttavia, la drastica distorsione della metrica indotta da questa operazione può produrre limiti inferiori e superiori molto negativi rispetto a quelli corrispondenti ai costi originari. Si noti che quando il costo di ciascun arco del grafico è uguale al costo del percorso più breve tra i suoi estremi, la matrice dei costi corrispondente soddisfa la disuguaglianza triangolare.

In alcuni casi i vertici sono associati a punti del piano aventi coordinate date, e il costo c_{ij}, per ogni arco $(i, j) \in A$, è definito come la distanza euclidea tra i due punti corrispondenti ai vertici i e j. In questo caso, la matrice dei costi è simmetrica per definizione e soddisfa la disuguaglianza triangolare.
Il problema che ne risulta è chiamato SCVRP euclideo.

Si può osservare che se arrotondassimo all'intero più vicino dei costi dell'arco euclideo a valori reali, si causarebbe una violazione della disuguaglianza triangolare, mentre ciò non accadrebbe se i costi venissero arrotondati per eccesso.

Ogni cliente i $(i = 1, .., n)$ è associato a una domanda nota non negativa d_i, da consegnare, e il deposito ha una domanda fittizia $d_o = 0$. Dato un insieme di vertici $S \subseteq V$, sia $d(S) = \sum_{i \in s} d_i$ la domanda totale dell'insieme.

Un insieme di K veicoli identici, ciascuno con capacità C, è disponibile presso il deposito. Condizione necessaria per l'ammissibilità del problema è che $d_i \leq C$ per ogni $i = 1, \ldots, n$. Ogni veicolo può effettuare al massimo una rotta, e assumiamo che K non sia inferiore a K_{min}, dove K_{min} è il numero minimo di veicoli necessari per servire tutti i clienti. Il valore di K_{min} può essere determinato risolvendo il Bin Packing Problem (BPP) associato al CVRP, che richiede la determinazione del numero minimo di contenitori (*bin*), ciascuno con capacità C, necessari per caricare tutti gli n articoli, ciascuno con peso non negativo d_i, $i = 1, \ldots, n$. Sebbene il BPP sia NP-hard in senso forte, le istanze con centinaia di elementi possono essere risolte ottimalmente in modo molto efficace.

Dato un insieme $S \subseteq V \backslash \{0\}$, indichiamo con $r(S)$ il numero minimo di veicoli necessari per servire tutti i clienti in S, ovvero il valore di soluzione ottimale del

BPP associato all'insieme di elementi S. Si noti inoltre che $r(V\backslash\{0\}) = K_{min}$. Spesso, $r(S)$ è sostituito dal limite inferiore di BPP

$$\lceil d(S)/C \rceil \tag{2.2}$$

Il CVRP consiste nel trovare un insieme di esattamente K circuiti semplici (ciascuno dei quali corrisponde al percorso di un veicolo) con costo minimo, il quale viene definito come la somma dei costi degli archi appartenenti ai circuiti, e tali che:

1. ogni circuito visiti il vertice del deposito;
2. ogni vertice del cliente viene visitato da esattamente un circuito;
3. la somma delle richieste dei vertici visitati da un circuito non supera la capacità del veicolo C.

In letteratura sono state considerate diverse varianti delle versioni base del CVRP. In primo luogo, quando il numero di veicoli disponibili K è maggiore di K_{min}, può essere possibile lasciare alcuni veicoli inutilizzati, e quindi devono essere determinati al massimo K circuiti. In questo caso, i costi fissi sono spesso associati all'uso dei veicoli e l'obiettivo aggiuntivo che richiede la minimizzazione del numero di circuiti (cioè dei veicoli utilizzati) si aggiunge a quella che richiede la minimizzazione del costo totale. Un'altra variante frequentemente considerata si presenta quando i veicoli disponibili sono diversi, ovvero che hanno diverse capacità C_k, $k = 1, \ldots, K$. Infine, non possono essere consentite rotte contenenti un solo cliente. Il CVRP è noto per essere un problema NP-hard (in senso forte) e generalizza il noto Traveling Salesman Problem (TSP), richiedendo la determinazione di un circuito semplice di costo minimo che visiti tutti i vertici di G (circuito hamiltoniano) e che si presenta quando $C > d(V)$ e $K = 1$. Pertanto, tutti i rilassamenti proposti per il TSP sono validi anche per il CVRP.

La prima variante di CVRP che consideriamo è il cosiddetto Distance-Constrained VRP (DVRP), dove per ogni rotta il vincolo di capacità è sostituito da un vincolo di lunghezza (o di tempo) massima. In particolare, una lunghezza non negativa t_{ij} (oppure t_e) è associata ad ogni arco $(i, j) \in A$ (o ad ogni connessione $e \in E$), e la lunghezza totale degli archi di ogni percorso non può superare la lunghezza massima del percorso T. Se i veicoli sono diversi, le lunghezze massime del percorso sono T_k, $k = 1, \ldots, K$. Inoltre, quando le lunghezze d'arco rappresentano i tempi di percorrenza, a ciascun cliente i può essere associato un tempo di servizio s_i, che denota il periodo di tempo per il quale il veicolo deve sostare nel luogo in cui si trova. In alternativa, i tempi di servizio possono essere sommati ai tempi di percorrenza degli archi, cioè definendo, per ogni arco (i, j), $t_{ij} = t'_{ij} + s_i/2 + s_j/2$, dove t'_{ij}. è il tempo di percorrenza originale dell'arco (i, j). Generalmente le matrici

di costo e lunghezza coincidono, cioè $c_{ij} = t_{ij}$ per tutti $(i, j) \in A$ (o $c_e = t_e$ per tutti $e \in E$). L'obiettivo del problema è quindi quello di minimizzare la lunghezza totale dei percorsi o della loro durata, quando il tempo di servizio è compreso nel tempo di percorrenza degli archi. Il caso in cui sono presenti sia la capacità del veicolo che i vincoli di distanza massima è chiamato CVRP vincolato alla distanza (DCVRP).

Continuiamo sviluppando una formulazione di programmazione lineare intera per ACVRP, che viene successivamente adattata a SCVRP. Il modello è una formulazione di flusso veicolare a due indici che utilizza $O(n^2)$ variabili binarie x per indicare se un veicolo attraversa un arco nella soluzione ottima. In altre parole, la variabile x_{ij} assume valore 1 se l'arco $(i, j) \in A$ appartiene alla soluzione ottima, mentre corrisponde al valore 0 altrimenti.

$$min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{2.3}$$

$$\sum_{i \in V} x_{ij} = 1 \quad \forall\, j \in V \backslash \{0\} \tag{2.4}$$

$$\sum_{j \in V} x_{ij} = 1 \quad \forall\, i \in V \backslash \{0\} \tag{2.5}$$

$$\sum_{i \in V} x_{i0} = K \tag{2.6}$$

$$\sum_{j \in V} x_{0j} = K \tag{2.7}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S) \quad \forall\, S \subset V \backslash \{0\}, S \neq \emptyset \tag{2.8}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} = \sum_{i \in S} \sum_{j \notin S} x_{ij} \quad \forall\, S \subseteq V \backslash \{0\}, S \neq \emptyset \tag{2.9}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} = |S| - r(S) \quad \forall\, S \subseteq V \backslash \{0\}, S \neq \emptyset \tag{2.10}$$

$$x_{ij} \in \{0{,}1\} \quad \forall\, i, j \in V \tag{2.11}$$

I vincoli indegree e outdegree (2.4) e (2.5) impongono che esattamente un solo arco associato a un cliente entra ed esca da ogni vertice. Analogamente i vincoli (2.6) e (2.7) impongono i requisiti di grado per il vertice deposito. Si noti che un vincolo arbitrario tra i $2|V|$ (2.4)-(2.7) è in realtà implicato dai restanti $2|V| - 1$, quindi può essere rimosso. I cosiddetti capacity-cut constraints (CCCs) di (2.8) impongono sia la connettività della soluzione che i requisiti di capacità del veicolo.

Essi prevedono infatti che ogni taglio $(V\backslash S,\ S)$ definito da un insieme di clienti S sia attraversato da un numero di archi non inferiore a $r(S)$ (numero minimo di veicoli necessari per servire il gruppo S). I CCCs rimangono validi anche se $r(S)$ è sostituito dal banale limite inferiore BPP (2.2).

Si osservi che quando si verifica che $|S| = 1$ o $S = V\backslash\{0\}$, i CCCs (2.8) diventano forme indebolite dei corrispondenti vincoli di grado (2.4)-(2.7). Nel vincolo (2.9) ogni taglio $(V\backslash S,\ S)$ viene attraversato in entrambe le direzioni dallo stesso numero di volte. Nel vincolo (2.10) vengono eliminati tutti i subtour generalizzati imponenedo che almeno $r(S)$ archi lascino ogni insieme di clienti S.

Entrambe le famiglie di vincoli (2.8) e (2.10) hanno una cardinalità che cresce esponenzialmente con n. Ciò significa che è praticamente impossibile risolvere direttamente il rilassamento della programmazione lineare del problema. Un possibile modo per superare parzialmente questo inconveniente è considerare solo un limitato sottoinsieme di questi vincoli e aggiungere i restanti solo se necessario, utilizzando opportune procedure di separazione. I vincoli considerati possono essere rilassati in maniera Lagrangiana, come fatto da Fisher [7] e Miller [8], oppure possono essere esplicitamente inclusi nel rilassamento della programmazione lineare, come fatto negli approcci branch-and-cut. In alternativa, una famiglia di vincoli equivalente a (2.8) e (2.10) e avente una cardinalità polinomiale può essere ottenuta considerando i vincoli di eliminazione dei subtour proposti per il TSP da Miller, Tucker e Zemlin [9] ed estendendoli a CVRP (Christofides, Mingozzi e Toth [10] e Desrochers e Laporte [11]):

$$u_i - u_j + Cx_{ij} \leq C - d_j \quad \forall\, i, j \in V\backslash\{0\}, i \neq j,\ \textit{tale che}\ d_i + d_j \leq C \tag{2.12}$$

$$d_i \leq u_i \leq C \quad \forall\, i \in V\backslash\{0\} \tag{2.13}$$

dove u_i, $i \in V\backslash\{0\}$, è una variabile continua aggiuntiva che rappresenta il carico del veicolo dopo aver visitato il cliente i. È facile vedere che i vincoli (2.12)-(2.13) impongono i requisiti sia di capacità che di connettività di CVRP. Infatti, quando si ha $x_{ij} = 0$, il vincolo (2.12) non è vincolante poiché $u_i \leq C$ e $u_j \geq d_j$, mentre quando $x_{ij} = 1$, impongono che $u_j \geq u_i + d_j$ (da notare che i subtour isolati vengono eliminati).

Vale la pena notare che il rilassamento della programmazione lineare della formulazione (2.3)-(2.7), (2.11), (2.12) e (2.13) è generalmente molto più debole di quello della formulazione (2.3)-(2.8), (2.11). Vincoli più restrittivi sono stati proposti da Desrochers e Laporte [11].

2.2 Vehicle Routing Problem with Time Windows

Il VRP with Time Windows (VRPTW) è l'estensione del CVRP in cui vengono imposti vincoli di capacità e ad ogni cliente i è associato un intervallo di tempo $[a_i, b_i]$, chiamato finestra temporale (time window). Vengono inoltre indicati l'istante temporale in cui i veicoli lasciano il deposito, il tempo di percorrenza t_{ij}, per ogni arco $(i, j) \in A$ (o t_e per ogni $e \in E$) e un tempo di servizio aggiuntivo s_i per ogni cliente i. Il servizio di ciascun cliente deve iniziare entro la finestra temporale associata e il veicolo deve fermarsi presso la sede del cliente per i primi istanti di tempo. Inoltre, in caso di arrivo anticipato presso la sede del cliente i, il veicolo è generalmente autorizzato ad attendere fino all'istante a_i, ovvero fino all'inizio del servizio.

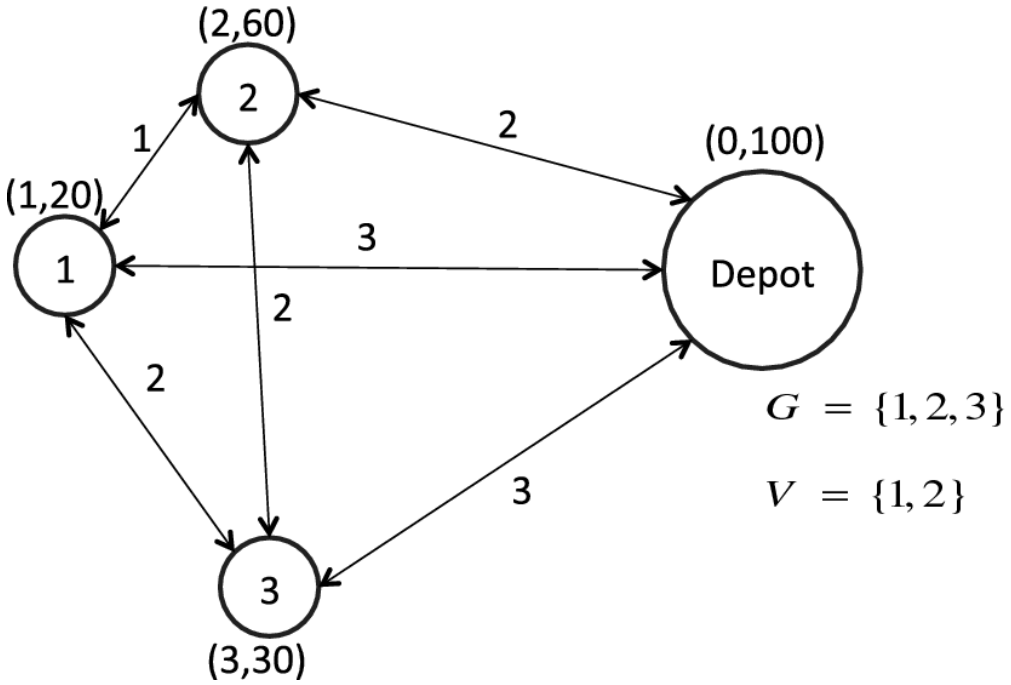

Figura 2.3: Esempio di problema di VRPTW dove ogni cliente ha una finestra temporale in cui è disponibile

Normalmente le matrici dei costi e dei tempi di percorrenza coincidono, e le finestre temporali sono definite assumendo che tutti i veicoli lascino il deposito all'istante 0. Si osserva inoltre che i requisiti della finestra temporale inducono un orientamento implicito di ogni percorso anche se le matrici originarie sono simmetriche. Pertanto, questo problema viene normalmente modellato come un problema asimmetrico.

Il problema di VRPTW consiste nel trovare un insieme di esattamente K circuiti semplici con costo minimo e tale che:

1. ogni circuito visiti il vertice del deposito;

2. ogni vertice del cliente venga visitato da esattamente un circuito;

3. la somma delle richieste dei vertici visitati da un circuito non superi la capacità del veicolo C;

4. per ogni cliente i, il servizio inizi entro la finestra temporale $[a_i, b_i]$, e il veicolo si fermi per s_i istanti temporali.

Inoltre questo è un problema NP-hard in senso forte, poiché generalizza il CVRP, nel caso in cui $a_i = 0$, $b_i = +\infty$, per ogni $i \in V\backslash\{0\}$. Inoltre, il cosiddetto TSP with Time Windows (TSPTW) è il caso speciale di VRPTW in cui $C \geq d(V)$ e $K = 1$.

Utilizzando la notazione della sezione precendete, il VRPTW è definito tramite il grafo $G = (V, A)$, dove il deposito è rappresentato dai due nodi 0 e $n + 1$. Tutti le tratte veicolari ammissibili corrispondono a percorsi in G che iniziano dal nodo 0 e terminano al nodo $n + 1$. Una finestra temporale è anche associata ai nodi 0 e $n + 1$, cioè, $[a_0, b_0] = [a_{n+1}, b_{n+1}] = [E, L]$, dove E e L rappresentano la prima possibile partenza dal deposito e rispettivamente l'ultimo arrivo possibile al deposito. Inoltre, per questi due nodi non sono definite richieste e tempi di servizio, cioè $d_0 = d_{n+1} = s_0 = s_{n+1} = 0$. Le soluzioni possibili esistono solo se $a_0 = E \leq min_{i\in V\backslash\{0\}}(b_i - t_{0i})$ e $b_{n+1} = L \geq min_{i\in V\backslash\{0\}}(a_i + s_i + t_{i0})$. Si noti inoltre che un arco $(i, j) \in A$ può essere eliminato per considerazioni temporali, se $a_i + s_i + t_{ij} > b_j$, oppure per limiti di capacità, se $d_i + d_j > C$, o ad altri fattori. Ricordiamo, infine, che quando i veicoli possono rimanere nel deposito, soprattutto nel caso in cui l'obiettivo primario consiste nel minimizzare il numero di veicoli utilizzati, l'arco $(0, n + 1)$ con $c_{0,n+1} = t_{0,n+1} = 0$ deve essere aggiunto all'insieme degli archi A.

Viene presentata in seguito una formulazione di programmazione matematica per il VRPTW che coinvolge due tipi di variabili: variabili di flusso $x_{ijk}, (i, j) \in A, k \in K$, uguale a 1 se l'arco (i, j) è utilizzato dal veicolo k e 0 in caso contrario, e variabili temporali $w_{ik}, i \in V, k \in K$, che specificano l'inizio del servizio al nodo i quando viene servito dal veicolo k.

Il VRPTW può quindi essere formalmente descritto con il seguente modello di flusso di rete multi-commodity con finestre temporali e vincoli di capacità:

$$min \sum_{k\in K} \sum_{(i,j)\in A} c_{ij} x_{ijk} \tag{2.14}$$

$$\sum_{k\in K}\sum_{j\in\Delta^+(i)} x_{ijk} = 1 \quad \forall\, i \in N \tag{2.15}$$

$$\sum_{j\in\Delta^+(0)} x_{0jk} = 1 \quad \forall\, k \in K \tag{2.16}$$

$$\sum_{i\in\Delta^-(j)} x_{ijk} - \sum_{i\in\Delta^+(j)} x_{jik} = 0 \quad \forall\, k \in K, j \in N \tag{2.17}$$

$$\sum_{i\in\Delta^-(n+1)} x_{i,n+1,k} = 1 \quad \forall\, k \in K \tag{2.18}$$

$$x_{ijk}(w_{ik} + s_i + t_{ij} - w_{jk}) \leq 0 \quad \forall\, k \in K, (i,j) \in A \tag{2.19}$$

$$a_i \sum_{j\in\Delta^+(i)} x_{ijk} \leq w_{ik} \leq b_i \sum_{j\in\Delta^+(i)} x_{ijk} \quad \forall\, k \in K, i \in N \tag{2.20}$$

$$E \leq w_{ik} \leq L \quad \forall\, k \in K, i \in \{0, n+1\} \tag{2.21}$$

$$\sum_{i\in N} d_i \sum_{j\in\Delta^+(i)} x_{ijk} \leq C \quad \forall\, k \in K \tag{2.22}$$

$$x_{ijk} \geq 0 \quad \forall\, k \in K, (i,j) \in A \tag{2.23}$$

$$x_{ijk} \in \{0{,}1\} \quad \forall\, k \in K, (i,j) \in A \tag{2.24}$$

La funzione obiettivo (2.14) di questa formulazione non lineare esprime il costo totale. Dato che $N = V\backslash\{0, n+1\}$ rappresenta l'insieme dei clienti, i vincoli in (2.15) limitano l'assegnazione di ciascun cliente ad esattamente un percorso di un solo veicolo. Successivamente, i vincoli (2.16)-(2.18) caratterizzano il flusso sul percorso che deve essere eseguito dal veicolo k. Successivamente, i vincoli (2.19)-(2.22) garantiscono la fattibilità della schedulazione rispetto alle considerazioni temporali e agli aspetti di capacità. Si noti che per un veicolo k dato, i vincoli in (2.20) impongono $w_{ik} = 0$ ogni volta che il cliente i non è visitato dal veicolo k. Infine, le condizioni (2.24) impongono condizioni binarie alle variabili di flusso.

Le condizioni binarie (2.24) consentono di linearizzare i vincoli (2.19) come:

$$w_{ik} + s_i + t_{ij} - w_{jk} \leq (1 - x_{ijk})M_{ij} \quad \forall\, k \in K, (i,j) \in A \tag{2.25}$$

dove M_{ij} sono valori costanti molto grandi. Inoltre, M_{ij} può essere sostituito da $max\{b_i + s_i + t_{ij} - a_j, 0\}, (i,j) \in A$, e i vincoli (2.19) o (2.25) devono essere applicati solo per gli archi $(i,j) \in A$ tale che $M_{ij} > 0$; altrimenti, quando $max\{b_i + s_i + t_{ij} - a_j, 0\} = 0$, questi vincoli vengono soddisfatti per tutti i valori di w_{ik}, w_{jk} e x_{ijk}.

2.3 Vehicle Routing Problem with Pickup and Delivery

Nella versione base del VRP with Pickup and Delivery (VRPPD), ad ogni cliente i sono associate due quantità d_i e p_i, che rappresentano rispettivamente la domanda di merce omogenea da consegnare e quella da ritirare presso il cliente i. A volte, per ogni cliente i viene utilizzata una sola quantità di domanda $d_i = d_i - p_i$ che indica la differenza netta tra le richieste di consegna e ritiro (questa possibilmente sarà negativa). Per ogni cliente i, O_i indica il vertice origine della domanda di consegna e D_i indica il vertice destinazione della domanda di ritiro.

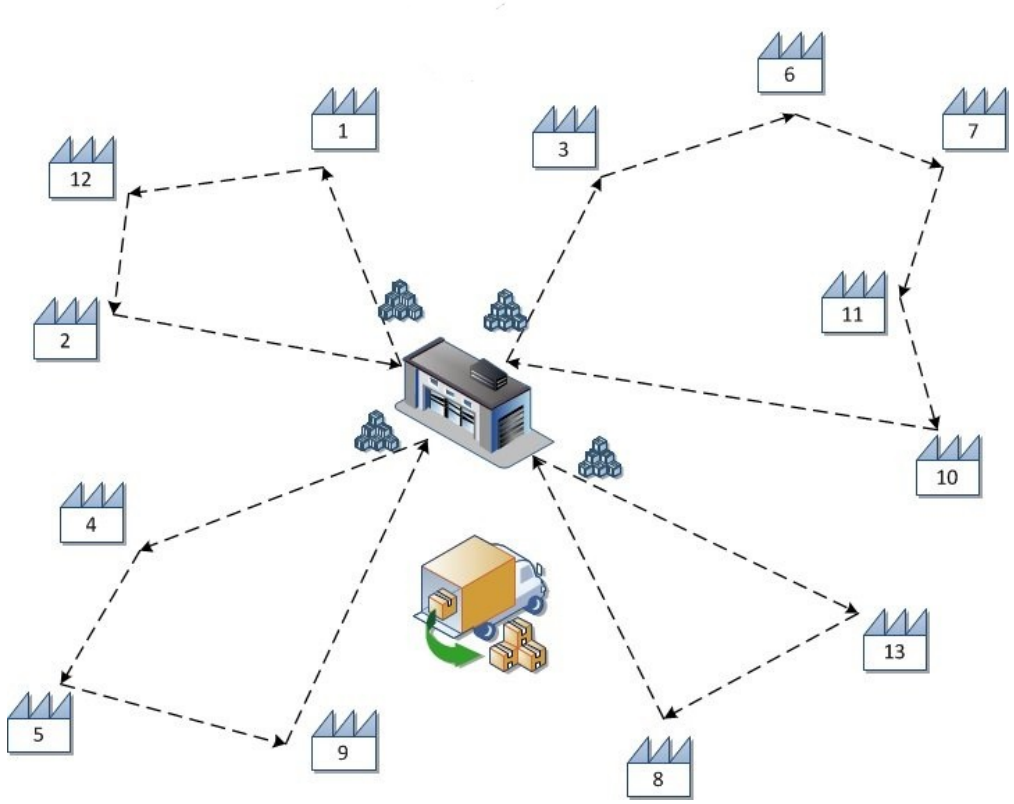

Figura 2.4: Esempio di problema di VRPPD dove un veicolo carica e/o scarica merce da portare e/o prelevare da ogni cliente

Si presume che, presso ogni sede del cliente, la consegna venga effettuata prima del ritiro; pertanto, il carico corrente di un veicolo prima di arrivare in una data località è definito dal carico iniziale meno tutte le richieste già consegnate più tutte le richieste già raccolte.

Il VRPPD consiste nel trovare un insieme di esattamente K circuiti semplici con costo minimo, e tale che:

1. ogni circuito visiti il vertice del deposito;
2. ogni vertice del cliente viene visitato da esattamente un circuito;
3. il carico corrente del veicolo lungo il circuito deve essere non negativo e non può mai superare la portata del veicolo C;
4. per ciascun cliente i il cliente O_i, se diverso dal deposito, deve essere servito nello stesso circuito e prima del cliente i
5. per ogni cliente i, il cliente D_i, se diverso dal deposito, deve essere servito nello stesso circuito e dopo il cliente i.

Spesso l'origine o la destinazione delle richieste sono comuni (ad esempio sono associate al deposito, come in CVRP), e quindi non è necessario indicarle esplicitamente. Questo problema è noto come VRP with Simultaneous Pickup and Delivery (VRPSPD). VRPPD e VRPSPD sono problemi NP-hard in senso forte, poiché generalizzano il CVRP nel caso in cui $O_i = D_i = 0$ e $p_i = 0$ per ogni $i \in V$. Inoltre, il cosiddetto TSP with Pickup and Delivery (TSPPD) è il caso speciale di VRPSPD in cui $K = 1$. Il caso di VRPPD in cui sono presenti finestre temporali è stato studiato in letteratura ed è chiamato VRP with Pickup and Deliveriy with Time Windows (VRPPDTW).

Identifichiamo la richiesta i tra due nodi come i e $n + i$, corrispondenti, rispettivamente, alla fermata di ritiro e alla fermata di consegna di una richiesta. È possibile che nodi diversi rappresentino la stessa posizione geografica. Quindi, denotiamo l'insieme dei nodi di prelievo con $P = \{1, \ldots, n\}$ e l'insieme dei nodi di consegna con $D = \{n + 1, \ldots, 2n\}$. Inoltre, definiamo $N = P \cup D$. Se la richiesta i consiste nel trasportare d_i unità da i a $n + i$, sia $l_i = d_i$ e $l_{n+i} = -d_i$.

Sia K l'insieme dei veicoli. Poiché non tutti i veicoli possono soddisfare tutte le richieste, ogni veicolo k ha un insieme specifico $N_k = P_k \cup D_k$ ad esso associato, dove N_k, P_k e D_k sono sottoinsiemi appropriati di N, P e D, rispettivamente. Per ogni veicolo k definiamo ora il grafo $G_k = (V_k, A_k)$. Poniamo $V_k = N_k \cup \{o(k), d(k)\}$ come insieme dei nodi comprensivi rispettivamente dell'origine $o(k)$ e della destinazione $d(k)$, depositi per il veicolo k. Il sottoinsieme A_k di V_k x V_k comprende tutti gli archi ammissibili. La capacità del veicolo k è data da C_k e il suo tempo e costo di viaggio tra i nodi distinti $i, j \in V_k$ equivalgono rispettivamente a t_k e Q_k.

Si presume che il veicolo k sia lasciato scarico nel suo deposito di origine all'istante $a_{o(k)} = b_{o(k)}$. Ogni percorso di ritiro e consegna ammissibile per questo veicolo corrisponde ad un percorso fattibile da $o(k)$ a $d(k)$ nel grafo G_k, visitando ogni nodo al massimo una volta. Se il veicolo visita il nodo $i \in N$, deve farlo

entro la finestra temporale $[a_i, b_i]$ quando deve iniziare il tempo di servizio s_i. Se arrivasse troppo presto, il veicolo potrebbe aspettare.

La formulazione prevede tre tipi di variabili:

- variabili di flusso binarie x_{ijk}, uguali a 1 se l'arco $(i, j) \in A_k$ è utilizzato dal veicolo k, 0 altrimenti;
- variabili temporali T_{ik} che specificano quando il veicolo k avvia il servizio dal nodo $i \in V_k$;
- variabili L_{ik} che restituiscono il carico del veicolo k dopo che il servizio a nodo $i \in V_k$ è stato completato.

La formulazione è la seguente:

$$min \sum_{k \in K} \sum_{(i,j) \in A_k} c_{ijk} x_{ijk} \tag{2.26}$$

$$\sum_{k \in K} \sum_{j \in N_k \cup \{d(k)\}} x_{ijk} = 1 \quad \forall\, i \in P \tag{2.27}$$

$$\sum_{j \in N_k} x_{ijk} - \sum_{j \in N_k} x_{j,n+i,k} = 0 \quad \forall\, k \in K, i \in P_k \tag{2.28}$$

$$\sum_{j \in P_k \cup \{d(k)\}} x_{o(k),j,k} = 1 \quad \forall\, k \in K \tag{2.29}$$

$$\sum_{i \in N_k \cup \{o(k)\}} x_{ijk} - \sum_{i \in N_k \cup \{d(k)\}} x_{jik} = 0 \quad \forall\, k \in K, j \in N_k \tag{2.30}$$

$$\sum_{i \in D_k \cup \{o(k)\}} x_{i,d(k),k} = 1 \quad \forall\, k \in K \tag{2.31}$$

$$x_{ijk}(T_{ik} + s_i + t_{ijk} - T_{jk}) \leq 0 \quad \forall\, k \in K, (i, j) \in A_k \tag{2.32}$$

$$a_i \leq T_{ik} \leq b_i \quad \forall\, k \in K, i \in V_k \tag{2.33}$$

$$T_{ik} + t_{i,n+i,k} \leq T_{n+i,k} \quad \forall\, k \in K, i \in P_k \tag{2.34}$$

$$x_{ijk}(L_{ik} + l_j - L_{jk}) = 0 \quad \forall\, k \in K, (i, j) \in A_k \tag{2.35}$$

$$l_i \leq L_{ik} \leq C_k \quad \forall\, k \in K, i \in P_k \tag{2.36}$$

$$0 \leq L_{n+i,k} \leq C_k - l_i \quad \forall\, k \in K, n+i \in D_k \tag{2.37}$$

$$L_{o(k),k} = 0 \quad \forall\, k \in K \tag{2.38}$$

$$x_{ijk} \geq 0 \quad \forall\, k \in K, (i,j) \in A_k \tag{2.39}$$

$$x_{ijk} \in \{0,1\} \quad \forall\, k \in K, (i,j) \in A_k \tag{2.40}$$

La funzione obiettivo (2.26) minimizza il costo totale del viaggio. I vincoli (2.27) e (2.28) impongono che ogni richiesta (ovvero i nodi di ritiro e consegna) sia effettuata esattamente una volta e dallo stesso veicolo. I vincoli (2.29)-(2.31) caratterizzano una struttura di flusso multi-commodity e assicurano che ogni veicolo k parta dal suo deposito di origine $o(k)$ e termini il suo percorso al suo deposito di destinazione $d(k)$. I requisiti di compatibilità tra i percorsi e gli orari sono gestiti dai vincoli delle finestre temporali (2.32) e (2.33). Per ogni richiesta, i vincoli (2.34) obbligano il veicolo a visitare il nodo di ritiro prima del nodo di consegna. Successivamente, i vincoli (2.35) esprimono i requisiti di compatibilità tra i percorsi e i carichi dei veicoli, mentre i vincoli (2.36)-(2.37) formano gli intervalli di capacità dipendenti dal veicolo rispetto ai nodi di ritiro e consegna. Infine, il carico iniziale del veicolo è imposto da (2.38) e la non negatività dei requisiti binari sono dati rispettivamente da (2.39) e (2.40). La serie di vincoli da (2.28) a (2.40), così come la funzione obiettivo, sono separabili per ogni veicolo $k \in K$. Questa formulazione limita la durata del percorso al massimo a $b_{d(k)} - a_{o(k)}$. Si noti inoltre che i vincoli (2.32), insieme ai vincoli della finestra temporale, consentono a un veicolo di attendere prima di visitare un nodo. Non c'è penalità sul tempo di attesa, e il tempo di arrivo al nodo j può essere calcolato come segue:

$$x_{ijk} = 1 \implies T_{jk} = max\{a_j, T_{ik} + s_i + t_{ijk}\} \quad (i,j) \in A_k \tag{2.41}$$

Anche la minimizzazione della dimensione della flotta può essere considerata in questa formulazione, incorporando un alto valore per i costi $c_{o(k),jk}$ per $j \in A_k$. In questo caso, si dovrebbe includere l'arco $(o(k), d(k))$ in A_k a costo zero per consentire l'inutilizzo di un veicolo. Quando la flotta è eterogenea, ai coefficienti di costo possono essere assegnati pesi disuguali per incoraggiare l'uso di alcune classi di veicoli rispetto ad altre.

Come proposto da Dumas, Desrosiers e Soumis [12] (vedi anche Desaulniers [13] per una discussione generale), la suddetta funzione obiettivo (lineare) può essere facilmente sostituita da una funzione non lineare più generale. Ad esempio, sia $c_k(L_{ik}) > 0$, $i \in V_k$, definita come una funzione non decrescente del carico totale trasportato sul veicolo k, subito dopo che il servizio è stato completato al nodo

i. Questa funzione agisce come fattore di penalizzazione sul costo del viaggio e la funzione obiettivo (2.26) può essere sostituita da:

$$min \sum_{k \in K} \sum_{(i,j) \in A_k} c_k(L_{ik}) c_{ijk} x_{ijk} \tag{2.42}$$

2.4 Multi-Depot Multi-Trip Vehicle Routing Problem with Time Windows and Release Dates

In questa sezione viene analizzato il Multi-Depot Multi-Trip Vehicle Routing Problem with Time Windows and Release Dates (Multi-D&T VRPTW-R), come problema composto da una parte di VRP a più depositi (Multi-D) e una parte di VRP a più viaggi (Multi-T) insieme ad un problema di VRPTW con date di rilascio (VRPTW-R).

Inanzitutto definiamo il problema di VRP a più depositi (Multi-D VRP) come una estensione del classico VRP. Inizialmente il problema era stato considerato per avere più depositi ma i veicoli partivano e arrivavano nello stesso deposito. Salhi, Arif e Wassan, 2013 [14], hanno considerato il problema del percorso dei veicoli con più depositi utilizzando veicoli eterogenei e hanno sviluppato un approccio di Variable Neighborhood Search, riscontrando che è altamente competitivo. Desaulniers, Lavigne, Soumis, 1998 [15], hanno proposto invece un problema a più depositi con finestre temporali (Multi-D VRPTW) che differentemente dal precedente, considerava in più gli intervalli limitati di servizio dei veicoli. Questo problema viene formulato come un modello di flusso di rete multi-commodity, intero e non lineare e viene definito utilizzando un framework di branch-and-bound che costituisce uno schema di branch-and-price assimilando un metodo di generazione di colonne. Dondo e Cerdà, 2009 [16], hanno ideato un algoritmo migliorativo di ricerca locale nella risoluzione del Multi-D VRPTW, il cui scopo è ispezionare grandi quartieri nelle soluzioni attuali per ottenere una serie di percorsi al minimo costo. Bettinelli, Ceselli e Righini, 2011 [17], hanno usato un algoritmo di branch-and-cut-and-price per trovare la corretta soluzione ad una variante di VRPTW, dove i veicoli vengono limitati da diverse capacità e partono da diversi depositi. Bae e Moon, 2016 [18], hanno applicato il Multi-D VRPTW a un problema di logistica e gestione della catena di approvvigionamento, tramite l'utilizzo di veicoli per la consegna e l'installazione di apparecchiature elettroniche.

Successivamente, definiamo il problema di VRP a più viaggi (Multi-T VRP) come una variante del VRP classico, nel quale viene considerato che ogni veicolo può effettuare più viaggi. Wassan Naved, Wassan Niaz, Gábor e Saïd, 2017 [19], hanno affrontato il problema di multi-trip VRP con backhauls implementando

un algoritmo di Variable Neighborhood Search a due livelli. Nguyen, Crainic e Toulouse, 2013 [20], hanno proposto una metaeuristica di Tabù Search per time-dependent multi-zone multi-trip VRPTW, la quale viene utilizzata per migliorare la selezione e l'allocazione del percorso del veicolo. I risultati sperimentali hanno indicato che il loro metodo può fornire soluzioni di qualità superiore sia in termini di numero di veicoli richiesti che di costo di viaggio. Yan, Chu, Hsaio e Huang, 2015 [21], hanno affrontato un problema di multi-trip split delivery VRPTW tramite l'utilizzo multiplo dei veicoli. Hanno proposto un algoritmo risolutivo a due step per trovare la soluzione a questo problema NP-hard. I risultati ottenuti hanno determinato quanto questi problemi di VRP possono essere risolti in maniera effettiva ed efficiente. Hernandez, Feillet, Giroudeau e Naud, 2016 [20], hanno affrontato un problema Multi-T VRPTW esaminando il caso nel quale i veicoli debbano visitare tutti i clienti avendo una durata illimitata. Per risolverlo hanno utilizzato due branch-and-price frameworks basati su copertura di formulazioni che coprono due set, in seguito hanno comparato queste due metodologie conducendo delle prove di prestazione su istanze di piccola scala, ovvero su 25 clienti. Tang, Yu e Li, 2015 Naud, 2016 [22], hanno proposto un algoritmo basato su modelli trip-chain-oriented set-partitioning per stabilire il problema di multi-trip VRP e scheduling problem.

Infine riguardo alla combinazione dei problemi a più viaggi (Multi-T VRP) e a più depositi (Multi-D VRP), L. Zhen, C. Ma, K. Wang, L. Xiao, W. Zhang, 2020 [6], hanno affrontato prima il problema Multi-D&T VRPTW-R basandosi sulle operazioni pratiche di consegna del pacchetto di acquisti online, in seguito hanno formulato questo problema come un modello MIP con l'obiettivo di ridurre al minimo il tempo totale di viaggio e di servizio di tutti i veicoli, infine è stato sviluppato un HPSO e un HGA per risolvere il problema.

Il Multi-D&T VRPTW-R può essere definito (utilizzando la notazione utilizzata nei capitoli precedenti) tramite un grafo orientato $G = (V, A)$, dove V è l'insieme dei vertici che è partizionato in un insieme di depositi D e un insieme di clienti N, e $A = (i, j)$, $i, j \in V$ insieme degli archi. Definiamo un insieme di veicoli K di uguale capacità C e un insieme di viaggi W. Ogni deposito avrà la propria flotta di veicoli. Ogni cliente $i \in N$ ha una domanda d_i che deve essere consegnata durante la finestra temporale $[a_i, b_i]$. Ogni veicolo che serve il cliente $i \in N$ e viaggia da $i \in N$ a $j \in V$ incorre in un tempo trascorso $t_i^S + t_{ij}^N$ (se $j \in N$, ovvero j è un cliente) oppure $t_i^S + t_{k,i}$, (se $j \in D$, ovvero se j è un deposito), dove t_i^S è il tempo per servire il cliente i, t_{ij}^N, è il tempo di viaggio tra il cliente $i \in N$ e il cliente $j \in N$, e $t_{k,i}$ è il tempo di viaggio tra il deposito del veicolo k e il cliente $i \in N$. Sia D_k definito come il deposito in cui si trova inizialmente il veicolo k. L'orario in cui un veicolo parte da un deposito per la consegna dei pacchi deve essere successivo alle date di rilascio t_i^R richieste da tutti i clienti inclusi nel viaggio del veicolo. Prima di quest'ultime, un mezzo deve essere caricato in un deposito per un certo tempo t_k^D,

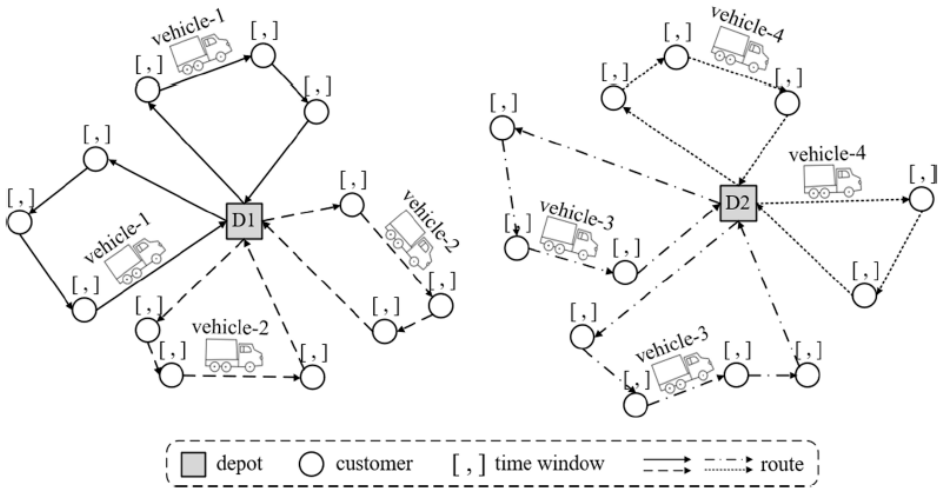

Figura 2.5: Esempio di problema di VRPTW-R dove un veicolo può fare più viaggi e andare in depositi differenti

il quale indica il tempo di servizio presso il deposito del veicolo k. Introduciamo il parametro T^H che rappresenta un orizzonte temporale e impostiamo così una finestra temporale $[0, T^H]$ per ogni deposito. Questo significa che ogni viaggio deve iniziare non prima di 0 e deve terminare non oltre T^H.

Avendo definito le variabili del problema, si suppone che:

- Tutta la flotta di veicoli sia omogenea e disponibile presso i depositi;
- Ogni viaggio deve iniziare e terminare nello stesso deposito;
- Non può esserci alcuna sovrapposizione di tempo tra ogni viaggio dello stesso veicolo;
- Ogni cliente viene servito esattamente una volta da un veicolo;
- La domanda totale servita durante il viaggio di un veicolo non può superare la capacità del veicolo.

Di seguito viene riportata la notazione utilizzata nella formulazione del modello.

Indici e insiemi:

- i,j,l: indice di un cliente.
- N: insieme di clienti.
- k: indice di un veicolo.

- K: insieme dei veicoli.
- w: indice di un viaggio.
- W: insieme di viaggi per tutti i veicoli.

Parametri:

- a_i: orario di inizio servizio presso il cliente i.
- b_i: orario di fine servizio presso il cliente i.
- d_i: domanda del cliente i.
- t_i^R: data di rilascio dei prodotti richiesti dal cliente i.
- t_i^S: tempo di servizio del cliente i.
- t_{ij}^N: tempo di viaggio tra il cliente i e j.
- t_k^D: tempo di servizio del deposito del veicolo k.
- $t_{k,i}$: tempo di viaggio tra il deposito del veicolo k e il cliente i.
- T^H: orizzonte temporale.
- C: capacità di ogni veicolo.
- M: numero positivo sufficientemente grande.

Variabili decisionali:

- τ_{kw}^L: momento temporale in cui il veicolo k parte dal deposito nel viaggio w.
- τ_{kw}^B: momento temporale in cui il veicolo k ritorna al deposito nel viaggio w.
- τ_{kiw}: momento temporale in cui il veicolo k arriva al cliente i nel viaggio w.
- δ_{kw}^L: carico del veicolo k nel viaggio w in partenza dal deposito.
- δ_{kiw}: carico del veicolo k nel viaggio w dopo che il veicolo ha visitato il cliente i.
- γ_{kw}: valore binario uguale a 1 se il veicolo k opera il viaggio w; 0 altrimenti.
- ξ_{ijkw}: valore binario uguale a 1 se il veicolo k visita il cliente j immediatamente dopo aver visitato il cliente i nel viaggio w; 0 altrimenti.
- λ_{kiw}: valore binario uguale a 1 se il veicolo k serve il cliente i nel viaggio w; 0 altrimenti.

- μ_{kiw}: valore binario uguale a 1 se il cliente i è il primo cliente servito dal veicolo k nel viaggio w; 0 altrimenti.
- $\eta : kiw$: valore binario uguale a 1 se il cliente i è l'ultimo cliente servito dal veicolo k nel viaggio w; 0 altrimenti.

Viene formulato di seguito il modello MIP di Multi-D&T VRPTW-R:

$$min \sum_{k\in K}\sum_{w\in W}\sum_{(i,j)\in N} t^N_{ij}\xi_{ijkw} + \sum_{k\in K}\sum_{w\in W}\sum_{i\in N} t_{ki}\mu_{kiw} + \sum_{k\in K}\sum_{w\in W}\sum_{i\in N} t_{ki}\eta_{kiw} \tag{2.43}$$

$$\sum_{k\in K}\sum_{w\in W}\lambda_{kiw} = 1 \quad \forall\, i \in N \tag{2.44}$$

$$\mu_{klw} + \sum_{i\in N}\xi_{ilkw} = \eta_{klw} + \sum_{j\in N}\xi_{ljkw} = \lambda_{klw} \quad \forall\, l \in N, k \in K, w \in W \tag{2.45}$$

$$\sum_{i\in N}\mu_{kiw} = \sum_{i\in N}\eta_{kiw} = \gamma_{kw} \quad \forall\, k \in K, w \in W \tag{2.46}$$

$$\gamma_{k,w+1} \leq \gamma_{k,w} \quad \forall\, k \in K, w \in W/|W| \tag{2.47}$$

$$\sum_{i\in N}\lambda_{kiw} \leq \gamma_{kw}|N| \quad \forall\, k \in K \tag{2.48}$$

$$\delta^L_{kw} = \sum_{i\in N}\lambda_{kiw}d_i \quad \forall\, k \in K, w \in W \tag{2.49}$$

$$\delta^L_{kw} \leq \gamma_{kw}C \quad \forall\, k \in K, w \in W \tag{2.50}$$

$$\tau_{kiw} \leq \lambda_{kiw}b_i \quad \forall\, i \in N, k \in K, w \in W \tag{2.51}$$

$$\tau^L_{kw} \geq \lambda_{kiw}(t^R_i + t^D_k) - M(1-\gamma_{kw}) \quad \forall\, i \in N, k \in K, w \in W \tag{2.52}$$

$$\tau^L_{k,w+1} \geq \tau^B_{kw} + t^D_k - M(1-\gamma_{k,w+1}) \quad \forall\, i \in N, k \in K, w \in W \tag{2.53}$$

$$\tau_{kiw} \geq \tau^L_{kw} + t_{ki} - M(1-\mu_{kiw}) \quad \forall\, i \in N, k \in K, w \in W \tag{2.54}$$

$$\tau^B_{kw} \geq max\{\tau_k iw, a_i\} + t^s_i + t_{ki} - M(1-\eta_{kiw}) \quad \forall\, i \in N, k \in K, w \in W \tag{2.55}$$

$$\tau_{kjw} \geq max\{\tau_k iw, a_i\} + t_i^s + t_{ij}^N - M(1 - \xi_{ijkw}) \quad \forall\, i, j \in N, k \in K, w \in W \tag{2.56}$$

$$\tau_{kw}^B \leq T^H \quad \forall\, k \in K, w \in W \tag{2.57}$$

$$\gamma_{kw} \in \{0,1\} \quad \forall\, k \in K, w \in W \tag{2.58}$$

$$\xi_{ijkw} \in \{0,1\} \quad \forall\, i \in N, k \in K, w \in W \tag{2.59}$$

$$\lambda_{kiw} \in \{0,1\} \quad \forall\, i \in N, k \in K, w \in W \tag{2.60}$$

$$\mu_{kiw} \in \{0,1\} \quad \forall\, i \in N, k \in K, w \in W \tag{2.61}$$

$$\eta_{kiw} \in \{0,1\} \quad \forall\, i \in N, k \in K, w \in W \tag{2.62}$$

La funzione obiettivo (2.43) minimizza il tempo di percorrenza totale di tutti i veicoli. Il vincolo (2.44) impongono che ogni cliente venga servito esattamente una volta. 2.45 mostra che, per un cliente servito durante il viaggio di un veicolo, se è il primo cliente nella sequenza dei clienti serviti di questo viaggio, nessun cliente può essere servito prima di lui nella stessa rotta, in caso contrario, deve esistere un cliente che viene servito esattamente prima di lui. Allo stesso modo, se è l'ultimo cliente nella sequenza di quelli serviti nello stesso viaggio, nessun altro può essere servito dopo di lui, altrimenti deve esserci un cliente che viene fornito dopo di lui. (2.46) assicura che in un viaggio effettuato debba esistere solo un primo cliente e un ultimo cliente da servire, al contrario, se il viaggio non venisse eseguito, non esisterebbe un primo cliente in questo viaggio e nemmeno l'ultimo. (2.47) garantisce che il viaggio $w + 1$ del veicolo k possa essere effettuato solo se ha effettuato il viaggio w. (2.48) definisce che un veicolo può servire un cliente solo se quest'ultimo è incluso nel viaggio. (2.49) assicura che i carichi totali di un veicolo corrispondano alla somma delle richieste del cliente. (2.50) conferma che i carichi totali di un veicolo non possono superare la sua capacità totale. (2.51) definisce la finestra temporale del servizio. (2.52) dichiara la relazione temporale tra l'orario di partenza di un veicolo e le sue date di rilascio, nonché l'orario di servizio di deposito in un viaggio. (2.53) stabilisce la relazione temporale tra viaggi consecutivi dello stesso veicolo. (2.54)–(2.55) sono vincoli di conservazione del tempo di viaggio. (2.56) è il vincolo di conservazione del tempo di servizio. (2.57) assicura che il tempo di riconsegna del veicolo al deposito non sia successivo all'orizzonte temporale T^H. I vincoli (2.58)–(2.62) definiscono i domini delle variabili decisionali.

Va notato che i vincoli (2.55) e (2.56) sono presentati in forma non lineare con funzioni di massimo, quindi non è necessario linearizzarli perché l'attuale versione degli strumenti di implementazione del modello applicati in questo studio supporta già l'utilizzo dell funzioni max e min.

Capitolo 3

Metodi euristici e algoritmi di riferimento

Il modello che viene prodotto nel capitolo 4 può essere risolto con differenti tool solvers (come ad esempio CPLEX) per instanze a piccola scala. Per risolvere i problemi a larga scala invece ci affidiamo all'utilizzo delle metauristiche.

In particolare, nel caso di questo elaborato, è stata utilizzata la metauristica di Adaptive Large Neighborhood Search (ALNS), ideata da S. Ropke e D. Pissinger, 2006 [23], per poi comparare la sua prestazione facendo diversi esperimenti su di essi. In particolare la ALNS è una estensione della Large Nighborhood Search (LNS) presentata da Shaw [24].

Nelle sezioni successive vengono descritti tutti i metodi utilizzati per risolvere il nostro problema. Inizialmente viene trattato il significato di euristica e metaeuristica nella Sezione 3.1, in seguito vengono esposte le euristiche di LNS nella Sezione 3.2 e di ALNS nella Sezione 3.3, per poi concludere con il criterio di stop di ricerca della soluzione migliore, entrando nel caso concreto della SA nella Sezione 3.4.

3.1 Definizione di euristica e di metaeuristica

In generale, per risolvere un problema di ottimizzazione combinatoria nel miglior modo possibile, utilizziamo dei metodi (definiti "esatti") che ci restituiscono una soluzione ottima della funzione obiettivo tra tutte le soluzioni che possono essere trovate. Questa strada non è sempre accettabile perchè principalmente il tempo impiegato per trovare la soluzione e la complessità del problema sono molto elevate, quindi è impossibile trovare una soluzione abbastanza precisa e pratica basandoci sulla formulazione di un modello di programmazione matematica del problema di ottimizzione. In altri casi, nonostante si abbia una buona formulazione matematica, non si ha una disponibilità temporale adeguata ad ottenere una soluzione ottimale

(che potrebbe essere una settimana) avendo un tempo di esecuzione molto lungo per poter gestire un problema di cui si vuole ottenere una soluzione vicina all'immediato (come nel nostro problema definito nella sezione 5.1, gestendo delle prenotazioni di viaggi per il giorno successivo).

Un'euristica deve necessariamente restituire una soluzione sub-ottima in tempi molto rapidi. Generalmente, la ricerca di soluzioni approssimate è quello che ci interessa nelle applicazioni reali (con problemi di grande dimensione), ed è dovuto principalmente dalle seguenti implicazioni:

- i parametri utilizzati nei reali applicativi sono delle approssimazioni sottoposte ad errore, per cui non si cerca di andare a lungo nella ricerca di una soluzione di valutazione dubbia.
- generalmente si eseguono questi algoritmi in tempo reale, quindi bisogna ottenere una soluzione ammissibile in tempi brevi.
- in certi momenti si vuole avere una soluzione realizzabile per il problema richiesto per poter realizzare in modo rapido degli scenari di lavoro.

Questi sono alcuni dei punti che ci fanno comprendere perchè siano molto utilizzati i metodi che ci possano dare delle buone soluzioni restituite in tempi di calcolo molto rapidi. Stiamo parliando proprio dei metodi euristici.

Ultimamente hanno preso molto campo gli approcci euristici generali, chiamate metauristiche. Sono delle evoluzioni delle euristiche, dove le idee e le strutture di fondo sono dichiarate, ma lo sviluppo delle varie parti dell'algoritmo varia a seconda di ciascun problema. Molte prendono esempio da fenomeni di vita reale reali (come la Simulated Annealing, SA, definita nella sezione 3.4). Le metauristiche possono essere considerate come generalizzazioni di un principale approccio, quello di ricerca locale, il quale si fonda su un criterio di andamento a tentativi. L'applicazione di qualsiasi metauristica a un problema di ottimizzazione richiede un lavoro molto accurato e complicato.

3.2 Large Neighborhood Search

La metauristica di Large Neighborhood Search (LNS) è stata introdotta da Shaw nel 1997 [24] per risolvere il problema di VRPTW. Il meccanismo principale è quello di distruggere (destroy) e riparare (repair) in modo iterativo una soluzione per avere un risultato migliore del precedente. Per una descrizione più approfondita, ho preso spunto dal documento di Ropke e Pisinger del 2018 [25].

L'euristica LNS appartiene alla classe di euristiche note come algoritmi di Very Large Scale Neighborhood search (VLSN) [26]. Tutti gli algoritmi VLSN sono basati sull'osservazione che la ricerca in un grande vicinato porta a trovare ottimi

locali di alta qualità, e quindi un algoritmo VLSN può restituire soluzioni migliori. Tuttavia, la ricerca in un grande quartiere richiede molto tempo, quindi vengono utilizzate varie tecniche di filtraggio per limitare la ricerca. Negli algoritmi VLSN, l'intorno è tipicamente limitato a un sottoinsieme di soluzioni che possono essere ricercate in modo efficiente.

Nella metaeuristica LNS, l'intorno è definito da un metodo di distruzione e da uno di riparazione. Un metodo di distruzione rimuove una parte della soluzione corrente, mentre un metodo di riparazione ricostruisce la soluzione distrutta utilizzando gli elementi rimossi dall'operazione precedente. Il metodo di distruzione contiene tipicamente un elemento di stocasticità, in modo che parti differenti della soluzione vengano distrutte ogni volta che il metodo viene invocato.

L'intorno $N(x)$ di una soluzione x è definito come l'insieme delle soluzioni che possono essere raggiunte applicando prima il metodo di distruzione e poi il metodo di riparazione.

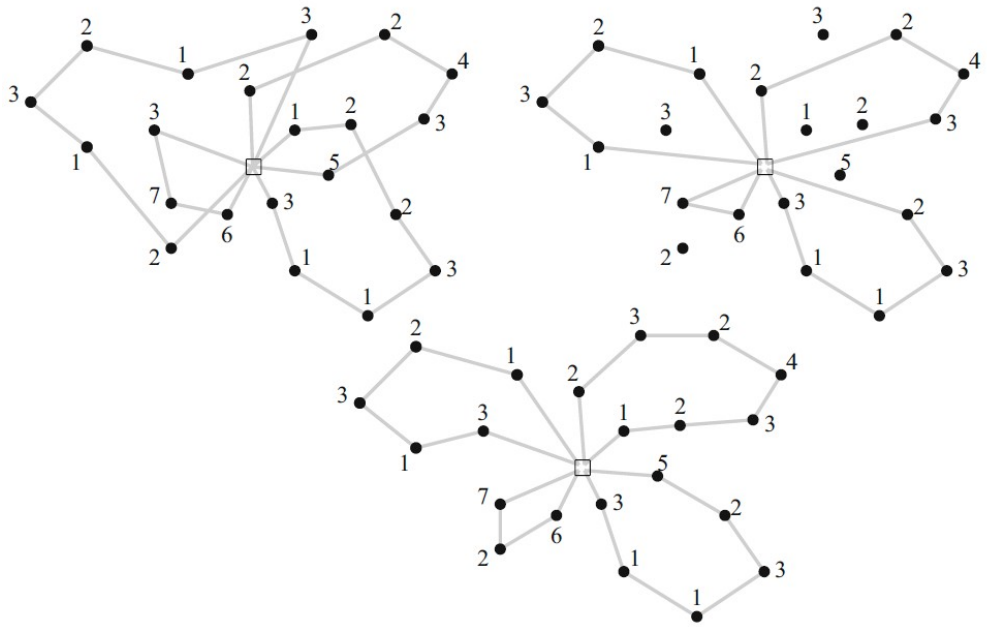

Figura 3.1: Esempio di applicazione dei metodi di distruzione e di riparazione. Nella figura in alto a sinistra viene mostrata una soluzione CVRP prima dell'operazione di distruzione. Nella figura in alto a destra viene mostrata la soluzione dopo un'operazione di distruzione che ha rimosso sei clienti (ora disconnessi dalle rotte). Nella figura in basso viene mostrata la soluzione ottenuta dopo il reinserimento dei clienti dall'operazione di riparazione.

Ad esempio, considerando il problema di CVRP, un metodo di distruzione potrebbe rimuovere il 15% dei clienti dalla soluzione attuale, tagliando i percorsi in

cui sono stati rimossi. Un metodo di distruzione molto semplice potrebbe rimuovere i clienti in modo casuale. Un metodo di riparazione, invece, potrebbe ricostruire la soluzione inserendo i nodi rimossi precedentemente, utilizzando un'euristica greedy, ovvero semplicemente scansionando tutti i clienti liberi, inserendo quello con il costo di inserzione più basso e ripetendo l'inserimento fino a quando non sono stati inseriti tutti. Le fasi di distruzione e riparazione sono illustrate in Figura 3.1.

Dato che il metodo di distruzione può rimuovere gran parte della soluzione, l'intorno conterrà tipicamente un gran numero di soluzioni. Consideriamo ad esempio un'istanza di CVRP con 100 nodi. Esistono quindi $C(100,10) = 100!/(15! * 85!) = 2.5 * 10^{17}$ diversi modi per selezionare i nodi da rimuovere se la percentuale o il grado di distruzione della soluzione è del 15%.

Esistono differenti modi per riparare la soluzione per ciascuna rimozione effettuata, ma scelte di rimozione diverse possono portare ovviamente alla stessa soluzione dopo la riparazione.

In seguito viene presentata l'euristica di LNS in modo più dettagliato, lo pseudocodice per questa euristica è mostrata nell'algoritmo 1.

Algorithm 1 Large Neighborhood Search

```
1:  procedure LNS(x)
2:      ▷ x è un possibile input
3:      x^b = x;                                  ▷ Soluzione migliore nella ricerca
4:      while il criterio di stop è soddisfatto do
5:          ▷ d(_) metodo di distruzione
6:          ▷ r(_) metodo di riparazione
7:          x^t = r(d(x));                        ▷ Soluzione temporanea
8:          if accept(x^t, x) then
9:              x = x^t;
10:         end if
11:         if c(x^t) < c(x^b) then
12:             x^b = x^t;
13:         end if
14:     end while
15:     return x^b                                ▷ Ritorna la miglior soluzione
16: end procedure
```

Nell'algoritmo troviamo tre variabili principali: x denota la soluzione corrente, x^b la migliore soluzione trovata durante la ricerca e x^t una soluzione temporanea che può essere scartata o promossa allo stato di soluzione corrente. La funzione $d(\_)$ rappresenta il metodo di distruzione mentre $r(\_)$ il metodo di riparazione. In particolare, $d(x)$ restituisce una copia di x parzialmente distrutta. L'applicazione

di $r(\_)$ ad una soluzione incompleta la ripara, restituendo una soluzione fattibile costruita da quella distrutta.

Nella riga (3) viene inizializzata la migliore soluzione globale. Nella riga (7), l'euristica applica prima il metodo di distruzione e poi il metodo di riparazione per ottenere una nuova soluzione x^t. Nella riga (8), viene valutata la nuova soluzione e l'euristica determina se questa soluzione deve diventare la nuova soluzione corrente (riga 9) o se deve essere rifiutata. La funzione di accettazione può essere implementata in diversi modi. La scelta più semplice è accettare solo soluzioni migliorative. Nella riga (11) viene controllato se la nuova soluzione è migliore della soluzione migliore, dove denotiamo con $c(x)$ il valore oggettivo della soluzione x. La soluzione globale viene aggiornata alla riga (12), se necessario. Si conclude il ciclo, quindi si torna alla riga (4) dove viene verificata la condizione di terminazione. Spetta all'implementatore scegliere il criterio di terminazione, impostando ad esempio un limite al numero di iterazioni o un limite di tempo. Alla riga (15) viene restituita la migliore soluzione trovata. Si può osservare dallo pseudocodice che la metaeuristica LNS non cerca l'intero intorno di una soluzione, ma si limita a campionare questo intorno.

L'idea principale alla base dell'euristica LNS è che i grandi quartieri consentono all'euristica di navigare facilmente nello spazio della soluzione, anche se l'istanza è strettamente vincolata. Questo è da opporsi alle euristiche di ricerca di piccoli quartieri (come quelle di Local Search) che possono rendere più difficile l'esplorazione di parti distanti nello spazio della soluzione.

Nel documento originale di LNS [24], il metodo di accettazione consentiva solo soluzioni migliorative (indichiamo tale metodo di accettazione come hill-climber). Negli articoli successivi "An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows" di S. Ropke e D. Pisinger [27] e "Record Breaking Optimization Results Using the Ruin and Recreate Principle" di G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt e G. Dueck [28] sono stati utilizzati criteri di accettazione presi in prestito dalla Simulated Annealing (SA).

Il metodo di distruzione è una parte importante dell'euristica di LNS. La scelta più importante quando si implementa questo tipo di algoritmo è il grado di distruzione: se venisse distrutta solo una piccola parte della soluzione, l'euristica potrebbe avere problemi ad esplorare lo spazio di ricerca poiché si perderebbe l'effetto del grande vicinato. Se una parte molto grande della soluzione venisse distrutta, l'euristica LNS quasi degraderebbe in ripetute riottimizzazioni. Questo potrebbe richiedere molto tempo o potrebbe produrre soluzioni di scarsa qualità a seconda di come viene riparata la soluzione parziale. Shaw [24] ha proposto di aumentare gradualmente il grado di distruzione, mentre Ropke e Pisinger [27] hanno scelto casualmente il grado di distruzione ad ogni iterazione, impostando il grado da un intervallo specifico dipendente dalla dimensione dell'istanza. Il metodo di distruzione, inoltre, deve consentire di raggiungere l'intero spazio di ricerca, o

almeno la parte interessante dello spazio di ricerca in cui ci si aspetta di trovare l'ottimo globale. Pertanto, non può concentrarsi sempre sulla rimozione di un particolare componente della soluzione, ma deve rendere possibile la distruzione di ogni parte della soluzione.

C'è molta libertà nella scelta del metodo di riparazione di una euristica di LNS. Una prima decisione da porsi è quella di verificare se il metodo di riparazione possa essere ottimale, ovvero se la migliore soluzione completa possibile possa essere costruita dalla soluzione parziale, oppure se debba essere un'euristica, assumendo che si sia soddisfatta una buona soluzione costruita dalla soluzione parziale. Un'operazione di riparazione ottimale sarà più lenta di un'operazione di euristica, ma potrebbe potenzialmente portare a soluzioni di alta qualità in poche iterazioni. Tuttavia, da un punto di vista della diversificazione, un'operazione di riparazione ottimale potrebbe non essere invitante: verrebbero prodotte solo soluzioni migliorative o di costo identico e potrebbe essere difficile lasciare valli nello spazio di ricerca a meno che gran parte della soluzione non venga distrutta ad ogni iterazione. Il metodo di riparazione può essere basato su un problema euristico specifico, su un metodo esatto, su una programmazione intera mista (MIP) di uso generale oppure su un risolutore di programmazione tramite vincoli.

Vale la pena osservare che l'euristica LNS tipicamente si alterna tra una soluzione non fattibile e una soluzione fattibile: l'operazione di distruzione crea una soluzione non fattibile che viene riportata in forma fattibile dall'euristica di riparazione. I metodi di distruzione e riparazione possono anche essere visti come operazioni di correzione/ottimizzazione: il metodo di correzione (corrispondente al metodo di distruzione) fissa parte della soluzione al suo valore corrente mentre il resto rimane libero; il metodo di ottimizzazione (corrispondente al metodo di riparazione) cerca di migliorare la soluzione attuale rispettando i vincoli fissati. Tale interpretazione dell'euristica può essere più naturale se il metodo di riparazione viene implementato utilizzando la MIP oppure tramite risolutori di programmazione tramite vincoli.

3.3 Adaptive Large Neighborhood Search

La metauristica Adaptive Large Neighborhood Search ALNS è stata introdotta da Ropke e Pisinger nel 2006 [27] per risolvere il problema di VRPPDTW. La ALNS estende la LNS utilizzando diverse euristiche di distruzione e riparazione durante la stessa ricerca, il metodo di selezione è guidato dalle statistiche raccolte durante l'analisi. La scelta viene fatta di volta in volta perchè un'euristica può essere adatta ad un tipo di istanza, mentra un'altra può essere più adatta per un altro tipo di istanza. Si ritiene che l'alternanza tra le diverse euristiche di rimozione e inserimento ci dia la possibilità di averne una più robusta.

Parametro	Descrizione
σ_1	L'ultima operazione di rimozione e inserimento ha portato a una nuova migliore soluzione globale.
σ_2	L'ultima operazione di rimozione e inserimento ha prodotto una soluzione che non è stata accettata prima. Il costo della nuova soluzione è migliore del costo della soluzione attuale.
σ_3	L'ultima operazione di rimozione e inserimento ha prodotto una soluzione che non è stata accettata prima. Il costo della nuova soluzione è peggiore del costo della soluzione attuale, ma la soluzione è stata accettata.

Tabella 3.1: Parametri di regolazione del punteggio nella scelta dell'euristica di rimozione e inserimento nella ALNS

Per selezionare l'euristica da utilizzare, assegnamo dei pesi alle diverse operazioni di distruzione e riparazione e utilizziamo un principio di selezione della ruota della roulette. Se abbiamo k euristiche con pesi w_i, $i \in \{1,2,\ldots,k\}$, selezioniamo l'euristica j con probabilità:

$$p(j) = \frac{w_j}{\sum_{i=1..k} w_i} \tag{3.1}$$

Si nota che l'euristica di riparazione è selezionata indipendentemente dall'euristica di distruzione e viceversa. E' possibile impostare questi pesi manualmente, ma potrebbe divenire un processo molto complicato se venissero utilizzate molte euristiche di rimozione e inserimento. Al contrario viene proposto un algoritmo di aggiustamento adattivo del peso.

I pesi w_j possono essere aggiustati automaticamente usando le statistiche delle precedenti iterazioni. L'idea principale è quella di tracciare un punteggio per ogni euristica, che misura quanto sia buona l'operazione appena utilizzata. Un punteggio elevato corrisponde ad un'euristica andata a buon fine. L'intera ricerca è suddivisa in una serie di segmenti. Un segmento è un numero di iterazioni dell'euristica ALNS. Il punteggio di tutte le euristiche è impostato a zero all'inizio di ogni segmento. Il punteggio di un'euristica viene aumentato di σ_1, σ_2 o σ_3 nelle situazioni mostrate nella tabella 3.1.

Il caso per σ_1 è chiaro: se un'operazione è in grado di trovare una nuova migliore soluzione complessiva, allora ha funzionato bene. Allo stesso modo, se un'operazione è stata in grado di trovare una soluzione che non è stata visitata prima ed è stata accettata dai criteri di accettazione nella ricerca ALNS, allora l'euristica ha avuto successo, poiché ha portato avanti la ricerca. E' utile distinguere

le situazioni corrispondenti ai parametri σ_2 e σ_3 perché preferiamo euristiche che possano migliorare la soluzione, ma ci interessano anche euristiche che possano diversificare la ricerca, quindi queste ultime vengono premiate con σ_3. È importante notare che premiamo solo le soluzioni non visitate. Naturalmente teniamo traccia delle soluzioni visitate.

Ad ogni iterazione applichiamo due euristiche: un'euristica di rimozione e un'euristica di inserimento. I punteggi per entrambe le euristiche vengono aggiornati dello stesso importo, dato che non possiamo capire se sia stata la rimozione o l'inserimento il motivo del "successo".

Alla fine di ogni segmento calcoliamo i nuovi pesi utilizzando i punteggi salvati. Consideriamo w_{ij} dell'operazione i usata nel segmento j come peso usato nella formula (3.1). Nel primo segmento pesiamo tutte le euristiche allo stesso modo. Dopo aver terminato il segmento j, calcoliamo il peso per tutte le euristiche i da utilizzare nel segmento $j + 1$ come segue:

$$w_{i,j+1} = w_{ij}(1 - r) + r\frac{\pi_i}{\theta_i} \tag{3.2}$$

π_i definisce il punteggio dell'euristica i ottenuto durante l'ultimo segmento e θ_i è il numero di volte in cui abbiamo tentato di utilizzare l'euristica i durante l'ultimo segmento. Il fattore di reazione r controlla la rapidità con cui l'algoritmo di adattamento del peso reagisce ai cambiamenti nell'efficacia delle euristiche. Se r fosse uguale zero, allora non utilizzeremo affatto i punteggi e ci atterremmo ai pesi iniziali. Se r fosse impostato a uno, lasceremo che sia il punteggio ottenuto nell'ultimo segmento a decidere il peso.

Le considerazioni menzionate in precedenza per la selezione dei metodi di distruzione e riparazione nell'euristica di LNS valgono anche per l'euristica di ALNS. Tuttavia, quest'ultima offre una maggior libertà perché consente la scelta su più metodi di distruzione/riparazione. Nell'euristica di LNS pura dobbiamo selezionare un'unica operazione di distruzione e un'unica di riparazione che dovrebbero funzionare bene per un'ampia gamma di istanze. In un'euristica di ALNS possiamo permetterci di includere metodi di distruzione/riparazione che sono adatti solo in alcuni casi: la regolazione del peso adattivo garantirà che queste euristiche vengano utilizzate raramente nei casi in cui siano inefficaci. Pertanto, la selezione dei metodi di distruzione e riparazione può essere trasformata in una ricerca di metodi che siano validi sia per la diversificazione che per l'intensificazione.

Dopo aver presentato le principali differenze tra l'euristica di LNS e di ALNS, viene presentata questa nuova euristica in maniera più dettagliato. Lo pseudocodice per questa euristica viene mostrata nell'algoritmo 2.

Quando confrontiamo questa euristica con lo pseudocodice di LNS nell'algoritmo 1, si può notare che sono state aggiunte le righe (4), (5), (7), (8) e (16). Gli insiemi dei metodi di distruzione e riparazione vengono indicati rispettivamente con Ω^- e

Algorithm 2 Adaptive Large Neighborhood Search

```
1: procedure ALNS(x)
2:     ▷ x è un possibile input
3:     x^b = x;                                  ▷ Soluzione migliore nella ricerca
4:     p^- = (1, ...,1);                         ▷ Pesi delle operazioni di rimozione
5:     p^+ = (1, ...,1);                         ▷ Pesi delle operazioni di inserimento
6:     while il criterio di stop è soddisfatto do
7:         selezione del metodo di rimozione d ∈ Ω^- usando i pesi p^-
8:         selezione del metodo di inserimento r ∈ Ω^+ usando i pesi p^+
9:         x^t = r(d(x));                        ▷ Soluzione temporanea
10:        if accept(x^t, x) then
11:            x = x^t;
12:        end if
13:        if c(x^t) < c(x^b) then
14:            x^b = x^t;
15:        end if
16:        aggiornamento di p^- e p^+
17:    end while
18:    return x^b                                ▷ Ritorna la miglior soluzione
19: end procedure
```

Ω^+. Nelle righe (4) e (5) sono state introdotte due nuove variabili: $p^- \in \mathbb{R}^{|\Omega^-|}$ e $p^+ \in \mathbb{R}^{|\Omega^+|}$, per memorizzare rispettivamente il peso di ciascun metodo di distruzione e riparazione. Inizialmente tutti i metodi hanno lo stesso peso. Nelle righe (7) e (8), i vettori di peso p^- e p^+ vengono utilizzati per selezionare i metodi di distruzione e riparazione utilizzando il principio della ruota della roulette. Nella riga (16) verranno aggiornati i pesi degli algoritmi di distruzione e riparazione a seconda del risultato ottenuto dalla soluzione, come già spiegato in precedenza tramite l'equazione (3.2).

La diversificazione e l'intensificazione nei i metodi di distruzione possono essere realizzati come segue: per diversificare la ricerca, si possono selezionare casualmente le parti della soluzione che dovrebbero essere distrutte (metodo di "random destroy"). Per intensificare la ricerca si pssono provare a rimuovere q variabili "critiche", ovvero le variabili che hanno un costo elevato o quelle che danneggiare l'attuale struttura della soluzione (es. archi che si incrociano in un TSP euclideo). Questo è noto come "worst destroy" oppure "critical destroy".

Si può anche scegliere una serie di variabili correlate che sono facili da interscambiare pur mantenendo la fattibilità della soluzione. Questa operazione chiamata "related destroy" di vicinati è stata introdotta da Shaw [24]. Per il problema di CVRP si può definire una misura di parentela tra ciascuna coppia di clienti. Questa

grandezza potrebbe essere semplicemente la distanza tra i clienti e includerebbe anche le domande dei clienti (quelli con una domanda simile sono considerati correlati). Pertanto, un metodo di distruzione correlato selezionerebbe un insieme di clienti che hanno un'elevata misura di correlazione reciproca. L'idea è quella che dovrebbe essere facile scambiare clienti simili.

Infine, si può usare l'operazione di "history based destroy", dove le variabili q sono scelte in base ad alcune informazioni storiche, come presentato in [23]. Queste informazioni potrebbero ad esempio contare quante volte l'impostazione di una determinata variabile (o insieme di variabili) porta a una cattiva soluzione. Si può quindi provare a rimuovere le variabili che sono attualmente assegnate ad un valore improprio, in base alle informazioni storiche.

I metodi di riparazione sono spesso basati su euristiche specifiche e ben eseguite per il problema dato. Esse possono fare uso di varianti del paradigma greedy, ad es. eseguire la scelta migliore a livello locale in ogni passaggio oppure eseguire la scelta meno sbagliata in ogni passaggio. Algoritmi di miglioramento tradizionali che esplorano piccoli quartieri possono essere utilizzati come parte di un metodo di riparazione per migliorare l'output di un algoritmo greedy.

I metodi di riparazione possono anche essere basati su algoritmi di approssimazione o algoritmi esatti, i quali possono essere rilassati per ottenere tempi di soluzione più rapidi a scapito della qualità della soluzione. Alcuni esempi vengono presentati in "A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows" di R. Bent e P. V. Hentenryck [29] e in "A general heuristic for vehicle routing problems" di S. Ropke e D. Pisinger [23].

I metodi di riparazione dispendiosi in termini di tempo e rapidi possono essere mischiati penalizzando i metodi che richiedono tempo come descritto in precedenza. L'utilizzo di un risolutore MIP per eseguire la fase di riparazione sta diventando sempre più interessante, perché diventano sempre più potenti ad ogni nuova versione rilasciata. L'approccio MIP ha il vantaggio che i metodi di riparazione per applicazioni complesse possono essere implementati rapidamente. Un potenziale svantaggio è dato dalla necessaria attenzione per quanto riguarda il grado di distruzione, poiché il metodo di riparazione potrebbe altrimenti diventare estremamente lento. Vale la pena sottolineare che un'euristica di ALNS con un metodo di riparazione MIP può essere vista come un prototipo di una metaeuristica.

La Figura 3.2 illustra, in modo astratto, tanti quartieri in un'euristica ALNS. Ogni quartiere sulla figura può essere considerato come una combinazione unica di un metodo di distruzione e riparazione.

Nella tradizionale euristica di ricerca locale, la diversificazione è controllata implicitamente dal paradigma di ricerca locale (rapporto di accettazione, Tabù Search, ecc.). L'euristica di ALNS controlla tipicamente la diversificazione attraverso il criterio di accettazione e in molte applicazioni ad ALNS viene applicata un'ulteriore diversificazione aggiungendo del rumore o della randomizzazione nei metodi di

distruzione e riparazione. La logica è quella di evitare processi di ricerca stagnanti in cui essi continuano a eseguire le stesse modifiche ad una stessa soluzione.

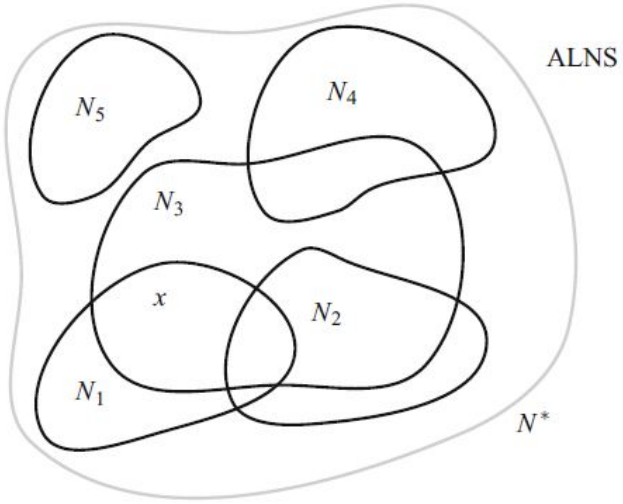

Figura 3.2: Illustrazione dei quartieri utilizzati da ALNS. La soluzione attuale è contrassegnata con x. ALNS opera su quartieri strutturalmente diversi $N_1, \ldots, N_k$ definito dalle corrispondenti euristiche di ricerca. Tutti i quartieri $N_1, \ldots, N_k$ in ALNS sono un sottoinsieme dell'intorno N^* definito modificando q variabili, dove q è una misura del massimo grado di distruzione

Il framework ALNS ha diversi vantaggi. Per la maggior parte dei problemi di ottimizzazione, conosciamo già una serie di euristiche ben performanti che possono costituire il nucleo di un algoritmo ALNS. A causa della grandezza e della diversità dei quartieri, l'algoritmo ALNS esplorerà grandi parti dello spazio delle soluzioni in modo strutturato. L'algoritmo risultante diverrà molto robusto in quanto potrà adattarsi a varie caratteristiche delle singole istanze e raramente rimarrà intrappolato negli ottimi locali.

La calibrazione della euristica ALNS è piuttosto limitata poiché il livello adattativo regola automaticamente l'influenza di ogni quartiere utilizzato. È ancora necessario calibrare le singole sottoeuristiche utilizzate per la ricerca dei quartieri di distruzione e riparazione, ma è possibile calibrarle individualmente o persino utilizzare i parametri degli algoritmi esistenti.

3.4 Simulated Annealing

La Simulated Annealing (SA) è uno dei metodi metaeuristici più semplici e conosciuti per affrontare difficili problemi di ottimizzazione globale di black box, la

cui funzione obiettivo non è data esplicitamente e può essere valutata solo tramite alcune costose simulazioni al computer.

All'inizio degli anni '80, tre ricercatori IBM, Kirkpatrick, Gelatt e Vecchi [30], hanno introdotto i concetti di ricottura nell'ottimizzazione combinatoria. Questi concetti si basano su una forte analogia con la ricottura fisica dei materiali. Questo processo comporta il portare un solido a uno stato di bassa energia dopo averne innalzato la temperatura. Può essere riassunto dai seguenti due passaggi (vedi Figura 3.3):

- Portare il solido ad altissima temperatura fino allo "scioglimento" della struttura;
- Raffreddare il solido secondo uno schema decrescente di temperatura molto particolare in modo da raggiungere uno stato solido di minima energia.

Nella fase liquida, le particelle sono distribuite casualmente. Si mostra che lo stato di minima energia viene raggiunto ad una condizione dove la temperatura iniziale è sufficientemente elevata e il tempo di raffreddamento è sufficientemente lungo. In caso contrario il solido si troverà allo stato ametastabile con energia non minima; questo è detto indurimento, che consiste nel raffreddamento improvviso di un solido.

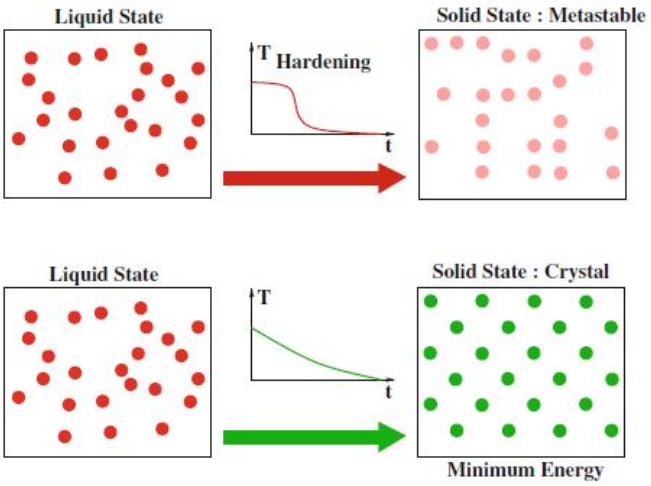

Figura 3.3: Grafico di comportamento di un materiale sottoposto a temperatura. Quando la temperatura è elevata, il materiale è allo stato liquido (a sinistra). Per un processo di indurimento, il materiale raggiunge uno stato solido con energia non minima (stato metastabile, in alto a destra). In questo caso, la struttura degli atomi non ha simmetria. Durante un processo di ricottura lenta, il materiale raggiunge anche uno stato solido per il quale gli atomi sono organizzati in simmetria (cristallo, in basso a destra)

Per descrivere l'algoritmo di SA, prima bisogna trattare l'algoritmo di Metropolis, in cui lo scopo è quello di riprodurre fedelmente l'evoluzione della struttura fisica di un materiale sottoposto a ricottura.

Questo algoritmo si basa su tecniche di Monte Carlo che consistono nel generare una sequenza di stati del solido nel modo seguente: partendo da uno stato iniziale i di energia E_i, si genera un nuovo stato j di energia E_j modificando la posizione di una particella; se la differenza di energia, $E_i - E_j$, è positiva (il nuovo stato è caratterizzato da un'energia inferiore), lo stato j diventa il nuovo stato corrente; se la differenza di energia è minore o uguale a zero, allora la probabilità che lo stato j diventi lo stato corrente è data da:

$$Pr(Current\ state = j) = e^{(\frac{E_i - E_j}{k_b * T})} \tag{3.3}$$

dove T rappresenta la temperatura del solido e k_b è la costante di Boltzmann ($k_b = 1.38 * 10^{-23} J/K$). Il criterio di accettazione del nuovo stato è chiamato criterio di Metropolis. Se il raffreddamento viene effettuato sufficientemente lentamente, il solido raggiunge uno stato di equilibrio ad ogni data temperatura T. Nell'algoritmo di Metropolis, questo equilibrio si ottiene generando un gran numero di transizioni ad ogni temperatura. L'equilibrio termico è caratterizzato dalla distribuzione statistica di Boltzmann. Questa distribuzione definisce la probabilità che il solido sia nello stato i di energia E_i alla temperatura T:

$$Pr(X = i) = \frac{1}{Z(T)} e^{-(\frac{E_i}{k_b \cdot T})} \tag{3.4}$$

dove X è una variabile casuale associata allo stato corrente del solido e $Z(T)$ è un coefficiente di normalizzazione, definito come:

$$Z(T) = \sum_{j \in S} e^{-(\frac{E_j}{k_b \cdot T})} \tag{3.5}$$

Quindi tornando all'algoritmo di SA, l'algoritmo Metropolis viene applicato per generare una sequenza di soluzioni nello spazio degli stati S. Per fare ciò, viene fatta un'analogia tra un sistema multiparticellare e il nostro problema di ottimizzazione utilizzando le seguenti equivalenze:

- I punti (soluzioni) dello spazio degli stati rappresentano i possibili stati del solido;
- La funzione da minimizzare rappresenta l'energia del solido.

Viene quindi introdotto un parametro di controllo c, che funge da temperatura. Questo parametro è espresso con le stesse unità dell'obiettivo ottimizzato.

Si assume inoltre che l'utente fornisca per ogni punto dello spazio degli stati, un quartiere e un meccanismo per generare una soluzione in questo quartiere. Definiamo quindi il principio di accettazione: sia (S,f) un'istanza di un problema di minimizzazione combinatoria, e i, j due punti nello spazio degli stati. Il criterio di accettazione per ammettere la soluzione j dalla soluzione corrente i è data dalla seguente probabilità:

$$Pr(accept\ i) = \begin{cases} 1 & se f(j) < f(i) \\ e^{(\frac{f(i)-f(j)}{c})} & altrimenti \end{cases} \tag{3.6}$$

Per analogia, il principio di generazione di un vicino corrisponde al meccanismo perturbativo dell'algoritmo di Metropolis, e il principio di accettazione rappresenta il criterio Metropolis.

Una transizione rappresenta la sostituzione della soluzione corrente con una soluzione vicina. Questa operazione viene eseguita in due fasi: generazione e accettazione.

In seguito, definiamo c_k come il valore del parametro di temperatura e L_k come il numero di transizioni generate dopo qualche iterazione k. Il principio di SA può essere riassunto come nell'algorirmo 3.

Algorithm 3 Simulated Annealing

procedure SIMULATEDANNEALING(i_{start}, c_0, L_0)
  ▷ i_{start} sol. iniziale, c_0 temperatura iniziale, L_0 numero transizioni iniziali
  $i = i_{start}$; $k = 0$; $c_k = c_0$; $L_k = L_0$; $l = 0$;
  while $c_k \cong 0$ **do**
    while $l \leq L_k$ **do**
      genera una soluzione j dal vicino S_i della soluzione corrente i;
      if $f(j) < f(i)$ **then** $i = j$;
      else j diventa la soluzione corrente con probabilità $e^{(\frac{f(i)-f(j)}{c})}$;
      end if
    end while
    $k = k + 1$;
    Calcola(L_k,c_k);
  end while
end procedure

Una delle caratteristiche principali della ricottura simulata è la sua capacità di accettare transizioni che degradano la funzione obiettivo. All'inizio del processo, il valore della temperatura c_k è elevato, il che consente di accettare transizioni con un elevato degrado oggettivo e quindi di esplorare a fondo lo spazio degli stati. Al diminuire di c_k vengono accettate solo le transizioni che migliorano l'obiettivo, o con un basso deterioramento. Infine, quando c_k tende a zero, non si accetta alcun deterioramento dell'obiettivo e l'algoritmo SA si comporta come un algoritmo di Monte Carlo.

Capitolo 4

Descrizione e formulazione del problema

In questo capitolo viene definito formalmente il problema Multi-D&T VRPPDTW con soste opzionali e viene formulato il relativo problema MIP, ponendo attenzione prima alla descrizione del modello ed in seguito alla sua formulazione.

4.1 Descrizione del problema

Sia $G = (V, A)$ un grafo orientato. Un insieme di veicoli V offre un servizio ai clienti soddisfando le seguenti ipotesi:

- I veicoli possono partire dal relativo deposito o da una stazione di parcheggio, con l'eccezione del primo viaggio che parte sempre dal deposito.
- I veicoli possono completare un viaggio nel deposito o nelle stazioni di parcheggio, con l'eccezione dell'ultimo viaggio che termina sempre al deposito.
- Ad ogni nodo è associata una finestra temporale.

Definiamo ora la notazione per il modello matematico del problema che sará utilizzata nella formulazione del matematica:

- $\mathcal{P} = \{1, \ldots, n\}$ sono i punti di partenza dei clienti da trasportare (nodi di prelievo);
- $\mathcal{D} = \{n + 1, \ldots, 2n\}$ è l'insieme dei punti di destinazione dei clienti (nodi di consegna);
- $\mathcal{N} = \mathcal{P} \cup \mathcal{D}$ è l'insieme di tutti i nodi relativi alle posizioni dei clienti;

- $\mathcal{S}$ è l'insieme delle stazioni di parcheggio. Il deposito può essere usato come parcheggio e per tale motivo una sua copia è inserita in $\mathcal{S}$
- $\mathcal{V} = \{1, \ldots, V\}$ è l'insieme dei veicoli che servono i clienti;
- $\mathcal{K} = \{1, \ldots, K\}$ è l'insieme dei possibili viaggi effettuati dai veicoli;
- $\mathcal{O}$, insieme dei depositi dei veicoli;
- $\mathcal{N}_0 = \mathcal{N} \cup \mathcal{O}$ insieme dei nodi cliente che include i depositi;
- $\mathcal{A} = \{(i,j) : i,j \in \mathcal{N}, i \neq j, i \neq j+n\} \cup \{(h,j) : h \in \mathcal{O}, j \in \mathcal{P}\} \cup \{(i,h) : i \in \mathcal{D}, h \in \mathcal{O}\} \cup \{(i,s) : i \in \mathcal{D}, s \in \mathcal{S}\} \cup \{(s,j) : j \in \mathcal{P}, s \in \mathcal{S}\}$ insieme degli archi che connettono i nodi e formano la rete;
- $\mathcal{A}^S = \{(i,j) : i \in \mathcal{D}, j \in \mathcal{P}, i \neq j+n\}$ insieme delle coppie di nodi cliente che possono essere visitati in sequenza con una sosta intermedia ad una stazione di parcheggio;
- $C_v, v \in \mathcal{V}$ capacità dei veicoli (numero massimo di passeggeri);
- $O_v, v \in \mathcal{V}$ deposito del veicolo v;
- T^{max} durata massima dei viaggi;
- D_{ij} è il costo di trasporto (cioè la distanza) per l'arco $(i,j) \in A$; la matrice dei costi corrisponde quindi alla matrice delle distanze;
- T_{ij} è il tempo di percorrenza per viaggiare per ogni arco $(i,j) \in A$;
- $G_i, i \in N$ è il tempo di servizio relativo al nodo specifico i, $G_i = G \cdot Q_i$ con G una costante pari al tempo medio di salita e discesa dal veicolo;
- $Q_i, i \in N$ è il numero totale di clienti da trasportare dal nodo iniziale;
- $[E_i, L_i], i \in N_0$ è la finestra temporale dei nodi;
- $U_i, i \in N$ due date del nodo cliente i;
- R tempo di sosta minimo ai parcheggi;
- $M_{i,j} = \max\{0, L_i + G_i + T_{i,j} - E_j\}, (i,j) \in \mathcal{A}$ bigM per i vincoli sui tempi di arrivo ai nodi;
- $M'_{i,j,s} = \max\{0, L_i + G_i + T_{i,s} + R + T_{s,j} - E_j\}, (i,j) \in \mathcal{A}^S, s \in S$ bigM per i vincoli associati al primo nodo visitato dopo una sosta ad un parcheggio;

- Π^T e Π^D sono i pesi delle componenti della funzione obiettivo, rispettivamente per il ritardo totale e la distanza totale percorsa;

Definiamo le variabili:

- $x_{i,j,v,k} \in \{0,1\}, (i,j) \in \mathcal{A}, v \in \mathcal{V}, k \in \mathcal{K}$ variabile binaria di routing tale che $x_{i,j,v,k} = 1$ se il veicolo v viaggia sull'arco (i,j) durante il suo trip k; $x_{i,j,v,k} = 0$ altrimenti;
- $z_{i,j,s} \in \{0,1\}, (i,j) \in \mathcal{A}^S, s \in S$ variabile binaria di flusso dei viaggi tale che $z_{i,j,s} = 1$ se un veicolo, dopo aver visitato il nodo i di delivery, completa il viaggio al parcheggio s e quindi inizia il nuovo viaggio visitando il nodo di pickup j;
- $a_j \in [E_j, L_j], j \in \mathcal{N}_0$ di arrivo al nodo j;
- $f_{i,j} \geq 0, (i,j) \in \mathcal{A}$ variabile di flusso che fornisce il carico del veicolo che viaggia sull'arco (i,j);
- $e_i, i \in N$ è l'anticipo dell'arrivo al nodo i rispetto alla sua due date;
- $t_i, i \in N$ è il ritardo dell'arrivo al nodo i rispetto alla sua due date.

4.2 Il modello matematico

Il problema può essere formulato per mezzo del seguente modello di programmazione matematica a numeri misti interi:

$$\min \ \Pi^D \cdot \sum_{v \in \mathcal{V}} \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} D_{i,j} x_{i,j,v,k} + \Pi^T \cdot \sum_{i \in \mathcal{N}} t_i + \Pi^E \cdot \sum_{i \in \mathcal{N}} E_i \tag{4.1}$$

soggetto ai vincoli

$$\sum_{v \in \mathcal{V}} \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} x_{i,j,v,k} = 1 \quad i \in \mathcal{P} \tag{4.2}$$

$$\sum_{(i,j) \in \mathcal{A}} x_{i,j,v,k} = \sum_{(h,i+n) \in \mathcal{A}} x_{h,i+n,v,k} \quad i \in \mathcal{P}, v \in \mathcal{V}, k \in \mathcal{K} \tag{4.3}$$

$$\sum_{j \in \mathcal{P}} x_{O_v,j,v,1} \leq 1 \quad v \in \mathcal{V} \tag{4.4}$$

$$\sum_{(i,j) \in \mathcal{A}} x_{i,j,v,k} = \sum_{(j,i) \in \mathcal{A}} x_{j,i,v,k} \quad i \in \mathcal{N}, v \in \mathcal{V}, k \in \mathcal{K} \tag{4.5}$$

$$\sum_{k=2}^{K} \sum_{j \in \mathcal{P}} x_{O_v,j,v,k} = 0 \quad v \in \mathcal{V} \tag{4.6}$$

$$\sum_{j \in \mathcal{P}} x_{O_v,j,v,1} = \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{D}} x_{i,O_v,v,k} \quad v \in \mathcal{V} \tag{4.7}$$

$$\sum_{v \in \mathcal{V}} \sum_{k \in \mathcal{K}} x_{i,s,v,k} \geq \sum_{(i,j) \in \mathcal{A}^S} z_{i,j,s} \quad i \in \mathcal{D}, s \in S \tag{4.8}$$

$$\sum_{v \in \mathcal{V}} \sum_{k \in \mathcal{K}} x_{s,j,v,k} \geq \sum_{(i,j) \in \mathcal{A}^S} z_{i,j,s} \quad j \in \mathcal{P}, s \in S \tag{4.9}$$

$$x_{s,j,v,k+1} \geq x_{i,s,v,k} + z_{i,j,s} - 1 \quad v \in \mathcal{V}, s \in S, k = 1, \ldots, K-1, (i,j) \in \mathcal{A}^S \tag{4.10}$$

$$\sum_{v \in \mathcal{V}} \sum_{k \in \mathcal{K}} \sum_{(s,j) \in \mathcal{A}} x_{s,j,v,k} \leq \sum_{(i,j) \in \mathcal{A}^S} z_{i,j,s} \quad s \in S \tag{4.11}$$

$$\sum_{v \in \mathcal{V}} \sum_{k \in \mathcal{K}} \sum_{(i,s) \in \mathcal{A}} x_{i,s,v,k} \leq \sum_{(i,j) \in \mathcal{A}^S} z_{i,j,s} \quad s \in S \tag{4.12}$$

$$\sum_{v \in \mathcal{V}} \sum_{k \in \mathcal{K}} x_{i,j,v,k} \leq 1 \quad (i,j) \in \mathcal{A} \tag{4.13}$$

$$\sum_{(i,s) \in \mathcal{A}} x_{i,s,v,k} = \sum_{(s,j) \in \mathcal{A}} x_{s,j,v,k+1} \quad v \in \mathcal{V}, k = 1, \ldots, K-1, s \in S \tag{4.14}$$

$$\sum_{(i,j) \in \mathcal{A}} (G_i + T_{i,j}) x_{i,s,v,k} \leq T^{max} \quad v \in \mathcal{V}, k \in \mathcal{K} \tag{4.15}$$

$$a_j + e_j - t_j = U_j \quad j \in \mathcal{N} \tag{4.16}$$

$$a_{j+n} \geq a_j + G_j + T_{j,j+n} \quad j \in \mathcal{P} \tag{4.17}$$

$$a_j \geq E_0 + T_{0,j} - (L_0 + T_{0,j})(1 - \sum_{v \in \mathcal{V}} x_{O_v,j,v,1}) \quad (O_v, j) \in \mathcal{A}, v \in \mathcal{V} \tag{4.18}$$

$$a_j \geq a_i + G_i + T_{i,j} - M_{i,j}(1 - \sum_{v \in \mathcal{V}} \sum_{k \in \mathcal{K}} x_{i,j,v,k}) \quad (i,j) \in \mathcal{A}, i \notin \mathcal{O}, j \notin S, M_{i,j} > 0 \tag{4.19}$$

$$a_j \geq a_i + G_i + T_{i,s} + R + T_{s,j} - M'_{i,j,s}(1 - z_{i,j,s}) \quad (i,j) \in \mathcal{A}^S, s \in \mathcal{S}, M'_{i,j,s} > 0 \tag{4.20}$$

$$\sum_{(j,i)\in A} f_{j,i} - \sum_{(i,j)\in A} f_{i,j} = Q_i \quad i \in \mathcal{N} \tag{4.21}$$

$$f_{i,j} \leq \sum_{v\in\mathcal{V}} C_v \sum_{k\in\mathcal{K}} x_{i,j,v,k} \quad (i,j) \in \mathcal{A} \tag{4.22}$$

$$\begin{aligned}
&a_j \in [E_j, L_j], j \in \mathcal{N} \\
&t_j \geq 0, j \in \mathcal{N} \\
&e_j \geq 0, j \in \mathcal{N} \\
&x_{i,j,v,k} \in \{0,1\}, (i,j) \in \mathcal{A}, v \in \mathcal{V}, k \in \mathcal{K} \\
&z_{i,j,s} \in \{0,1\}, (i,j) \in \mathcal{A}^S, s \in S \\
&f_{ij} \geq 0, (i,j) \in \mathcal{A}
\end{aligned} \tag{4.23}$$

L'obiettivo (4.1) modella la minimizzazione del costo totale che include il costo dovuto alla distanza totale percorsa e i costi dovuti agli arrivi in ritardo o in anticipo ai nodi da servire.

I vincoli (4.2) e (4.3) impongono che ciascun nodo di pickup e di delivery sia servito. I vincoli (4.4) stabiliscono che i veicoli disponibili possono essere opzionalmente utilizzati. I vincoli (4.5) sono le condizioni di conservazione del flusso ai nodi cliente. I vincoli (4.6) impongono che i viaggi successivi al primo di ciascun veicolo non possano partire dai depositi ma solo dalla stazioni di parcheggio. I vincoli (4.7) impongono che ogni veicolo utilizzato alla fine rientri al suo deposito.

La coppia di vincoli (4.8) e (4.9) impongono che le variabili di routing assumano valore 1 quando un veicolo effettua una sosta al parcheggio s in cui termina un viaggio dopo aver servito il nodo di delivery i, per poi proseguire con un nuovo viaggio che serve inizialmente il nodo di pickup j, ossia quando $z_{i,j,s} = 1$. Quindi i vincoli (4.10) impongono che il veicolo che ha terminato in un parcheggio s il proprio viaggio k parta dallo stesso parcheggio con il viaggio $k+1$ in accordo con quando stabilito dalle variabili $z_{i,j,s}$. I vincoli (4.11) e (4.12) impongono che le variabili $z_{i,j,s}$ assumano valore 1 se le variabili di routing $x_{i,s,v,k}$ e $x_{s,j,v,k}$ valgono 1, ossia quando un viaggio di un veicolo termina o inizia nella stazione s. I vincoli (4.13) impongono che tra due nodi cliente possa passare un solo viaggio di un solo veicolo. I vincoli (4.14) stabiliscono la continuità dei viaggi alle stazioni di parcheggio, mentre i (4.15) impongono la durata massima dei viaggi.

I vincoli (4.16) definiscono la tardiness e la earliness per i nodi cliente. I vincoli (4.17) impongono il tempo di arrivo minimo ai nodi di delivery quando questi sono

serviti direttamente dopo i corrispondenti nodi di pickup. I vincoli (4.18) calcolano le condizioni sui tempi di arrivo ai nodi di pickup quando raggiunti direttamente dopo la partenza da un deposito. Similmente, i (4.19) forniscono le condizioni sui tempi di arrivo a un nodo cliente o per il rientro al deposito quando si viaggia da un diverso nodo cliente. I vincoli (4.20) danno le condizioni sui tempi di arrivo al primo nodo di pickup dopo la sosta ad un parcheggio. I vincoli (4.21) sono le condizioni di continuità del carico dei veicoli che garantiscono che le variabili f_{ij} conteggino il corretto numero di passeggeri presente sul veicolo che viaggia sull'arco (i, j), mentre i (4.22) impongono che tale numero non superi mai la capacità del veicolo. Infine le (4.23) sono la definizione delle variabili decisionali.

Capitolo 5

Euristiche e metodi utilizzati

Oltre ad aver tenuto conto delle euristiche e dei metodi definiti nel capitolo 3, che verranno utilizzati per la risoluzione del problema affrontato, bisogna considerare altri aspetti che lo rendono unico nel suo genere, valutando gli aspetti che sono stati adattati per trovare una soluzione personalizzata.

In seguito vengono descritti questi aspetti dell'algoritmo di risoluzione: inizialmente viene presentata l'euristica di ALNS adattata alla risoluzione di questo problema nella Sezione 5.1, in seguito vengono definite tutte le operazioni utilizzate dalla ALNS, ovvero le euristiche di rimozione delle richieste nella Sezione 5.2, le euristiche di inserimento di richieste nella Sezione 5.3, le euristiche di rimozione delle aree di sosta nella Sezione 5.4 e le euristiche di inserimento delle aree di sosta nella Sezione 5.5. Si prosegue con la definizione della gestione delle finestre temporali nella Sezione 5.6, successivamente si mostrano quali siano i vincoli del problema non violabili nella Sezione 5.7 e quelli che possono essere violati nella Sezione 5.8. Infine viene analizzata nel dettaglio la formulazione della soluzione iniziale utilizzata come primo passo della ALNS nella Sezione 5.5.

5.1 ALNS sviluppata

Per lo sviluppo di questa ALNS lo studio è vertito su una personalizzazione della metaeuristica di base dove viene utilizzato l'algoritmo di SA come criterio di stop nella ricerca della soluzione, oltre al numero massimo di iterazioni fissate. Inoltre utilizza le operazioni di rimozione ed inserimento già definite precedentemente per la parte di richieste, mentre si hanno degli algoritmi che sono stati creati ad hoc per la gestione delle operazioni per le aree di sosta. Questa parte viene definita più nel dettaglio nello pseudocodice dell'algoritmo 4.

Algorithm 4 ALNS VRPPDTW & MT & MD & OP

procedure ALNS-VRPPDTW-MT-MD-OP($q_1 \in \mathbb{N}^*$, $q_2 \in \mathbb{N}^*$)
  s = FindInitSol();
  $s_{best} = s$;
  $weight_i = 1; score_i = 0; times_i = 0; i$ indice per tutte le operazioni
  $P_0 = 0.99; \lambda = 0.1; r = 0.1; c = 0.99;$
  $T_0 = -\frac{\lambda}{\ln P_0} \cdot s; T_f = 1 \cdot 10^{-6}; T = T_0;$
  $\sigma_1 = 40; \sigma_2 = 10; \sigma_3 = 2;$
  $maxIt = 5 \cdot 10^4; updNIt = 150; updIt = 10; iter = 0; worseSeq = 0;$
  while $T > T_f$ and $worseSeq < updNIt$ and $iter < maxIt$ **do**
    $s_{new} = s;$
    $selected = random[0,3); selected \in \mathbb{Z}^+$
    seleziono operaz. casuale Rr, Dr_{q1}, Rp, Dp_{q2} rispetto ai loro pesi;
    if $selected = 0$ **then**
      $s_{new} = Rr(Dr_{q1}(s_{new}));$
    else if $selected = 1$ **then**
      $s_{new} = Rp(Dp_{q2}(Rr(Dr_{q1}(s_{new}))));$
    else
      $s_{new} = Rr(Dr_{q1}(Rp(Dp_{q2}(s_{new}))));$
    end if
    if $f(s_{new}) < f(s_{best})$ **then**
      $s_{best} = s = s_{new}; worseSeq = 0;$
      $score_i = score_i + \sigma_1; i$ indice operazioni selezionate
    else if $f(s_{new}) < f(s)$ **then**
      $s = s_{new}; worseSeq = worseSeq + 1;$
      $score_i = score_i + \sigma_2; i$ indice operazioni selezionate
    else if $p = random[0,1) < e^{-\frac{s_{new}-s}{T}}, p \in \mathbb{R}$ **then**
      $s = s_{new}; worseSeq = worseSeq + 1;$
      $score_i = score_i + \sigma_3; i$ indice operazioni selezionate
    else
      $worseSeq = worseSeq + 1;$
    end if
    if $worseSeq = \frac{2}{3} \cdot updNIt$ **then**
      $T = \frac{3}{4} T_0$
    end if
    if sono passate $updIt$ iterazioni prima dell'ultimo aggiorn. dei pesi **then**
      $weight_i = weight_i(1-r) + r\frac{score_i}{times_i + 1}; i$ indice operazioni selezionate
    end if
    $T = c \cdot T; iter = iter + 1;$
  end while
  return $s_{best};$
end procedure

Analizziamo più nel dettaglio questa metauristica. La funzione prende in input due numeri interi positivi q_1, che consiste nel numero di richieste che devono essere rimosse dall'utilizzo di una operazione di distruzione sulle richieste, e q_2, che consiste nel numero di aree di sosta che devono essere rimosse tramite l'utilizzo di una operazione di distruzione nelle aree di sosta. Inizialmente si trova in s la soluzione iniziale dell'algoritmo (riga 2, l'algoritmo viene definito in seguito nell'algoritmo 7 nella Sezione 5.9) e la si assegna come soluzione migliore in s_{best} (riga 3). Si procede con la definizione di tutte le variabili utilizzate per l'algoritmo, inizialmente definendo le variabili di tutte le operazioni: $weight_i$ corrisponde a ciascun peso delle operazioni i, che avranno valore 1, $score_i$ che memorizzano il punteggio totale ottenuto dall'utilizzo delle operazioni i ogni volta che viengono utilizzate per trovare una soluzione migliore e $times_i$ che determinano il numero di volte che sono state utilizzate le operazioni i nella metauristica, le quali avranno valore 0 (riga 3). i è un indice che appartiene a tutte le operazioni esistenti. Quindi definiamo alcune delle costanti del problema derivanti dalle iterazioni, come la probabilità di accettazione P_0 impostata al 99%, la probabilità di accettazione λ definita a 0.1, il fattore di reazione r al 10% e il cooling rate c pari al 99% (riga 5). In seguito vengono definite le variabili che operano sulle temperature dell'algoritmo di SA, dove troviamo la temperatura iniziale T_0, calcolata come la negativa del rapporto tra la probabilità di accettazione (λ) e il logaritmo naturale della probabilità iniziale (P_0), il tutto moltiplicato per il costo della soluzione iniziale, successivamente si definisce la temperatura finale T_f pari a $1 \cdot 10^{-}6$ e la temperatura corrente T uguale alla temperatura iniziale (riga 6). Si definiscono, in seguito, i punteggi da aggiungere alle operazioni utilizzate per distruggere e riparare la soluzione ad ogni iterazione, con valore σ_1 pari a 40 se si trova una soluzione migliore di tutte le precedenti, σ_2 pari a 10 se si trova una soluzione locale migliorativa e σ_3 pari a 2 e se si trova una soluzione accettabile (riga 7). Si definiscono le ultime costanti che vanno ad agire sulle iterazioni dell'algoritmo, dove $maxIt$ sarà il numero massimo di iterazioni da effettuare, pari a $5 \cdot 10^4$, $updIt$ come il numero di operazioni dopo le quali si aggiornano tutti i pesi delle operazioni, pari a 10, $updNIt$ come il numero di iterazioni massimo nella quale avviene la terminazione per soluzione *notimproving*, pari a 150, *iter* che definisce il numero della iterazione corrente, che partirà da 0 ed infine *worseSeq* che memorizza il numero di iterazioni che non hanno portato al ritrovamento di una soluzione migliore rispetto a quella globale corrente, pari a 0 (riga 8). Dopo aver definito tutte le costanti e le variabili del problema, si entrerà in un ciclo dove verrà eseguito l'algoritmo e non terminerà finchè la temperatura corrente T non sarà minore della temperatura finale T_f, il numero di iterazioni *notimproving* sarà uguale a $updNIt$ oppure finchè il numero di iterazioni correnti *iter* non sarà uguale al massimo delle iterazioni $maxIt$ (righe 9-37). Analizzando il ciclo di ogni iterazione della ALNS, verrà definita una variabile s_{new} pari alla soluzione s trovata al precedente passo (riga 10), una variabile *selected* che conterrà

un numero randomico tra 0, 1 e 2 che selezionerà quale tipo di strategia operare successivamente (riga 11). Quindi si seleziona una operazione casuale Rr tra tutte quelle di rimozione delle richieste, una operazione casuale Dr_{q1} tra tutte quelle di inserimento delle richieste, una operazione casuale Rp tra tutte quelle di rimozione delle aree di sosta e una operazione casuale Dp_{q2} tra tutte quelle di inserimento delle aree di sosta (riga 12). Se è stato selezionato il metodo 0, allora si applicherà l'operazione di distruzione delle richieste Dr_{q1} e in seguito l'operazione di riparazione delle richieste Rr alla soluzione s_{new} e verrà aggiornata (righe 13-14). Altrimenti se è stato selezionato il metodo 1, allora si applicherà l'operazione di distruzione delle richieste Dr_{q1}, seguita dall'operazione di riparazione delle richieste Rr, successivamente dall'operazione di distruzione delle aree di sosta Dp_{q2} ed infine dall'operazione di riparazione delle aree di sosta Rp alla soluzione s_{new} e verrà aggiornata (righe 15-16). Se non sono state rispettate le due condizioni precedenti, allora è stato selezionato il metodo 2, quindi si applicherà l'operazione di distruzione delle aree di sosta Dp_{q2}, in seguito si utilizzerà l'operazione di riparazione delle aree di sosta Rp, successivamente l'operazione di distruzione delle richieste Dr_{q1} ed ininfine l'operazione di riparazione delle richieste Rr alla soluzione s_{new} e verrà aggiornata la soluzione (righe 17-19). Se il costo della nuova soluzione s_{new} è minore del costo della soluzione globale s_{best}, allora si assegna la nuova soluzione s_{new} come globale s_{best} e locale s, si azzera il numero di iterazioni not improving $worseSeq$ e si aggiornano i punteggi delle operazioni utilizzate precedentemente aggiungendo il punteggio σ_1 (righe 20-22). Altrimenti se il costo della nuova soluzione è minore del costo della soluzione locale s, allora si assegna la nuova soluzione s_{new} come locale s, si aumenta di 1 il numero di iterazioni not improving $worseSeq$ e si aggiornano i punteggi delle operazioni utilizzate precedentemente sommando il punteggio σ_2 (righe 23-25). Nell'ultimo in caso in cui non sono state rispettate le due condizioni precedenti, se un valore p randomico tra 0 compreso e 1 non compreso è minore del valore $e^{-\frac{s_{new}-s}{T}}$ (criterio di accettazione della soluzione tramite l'algoritmo di SA), allora si assegna la nuova soluzione s_{new} come accettata in s, si aumenta di 1 il numero di iterazioni not improving $worseSeq$ e si aggiornano i punteggi delle operazioni utilizzate precedentemente aggiungendo il punteggio σ_3 (righe 26-29). Successivamente se il numero di iterazioni not improving è pari a $\frac{2}{3}$ del massimo di iterazioni not improving, la temperatura corrente verrà impostata a $\frac{3}{4}$ di quella iniziale (righe 30-32), questo effetto è definito come rialzo della temperatura per dare la possibilità di trovare nuove soluzioni migliori. In seguito si aggiornano i pesi di tutte le operazioni, solo se sono passate $updIt$ iterazioni dall'ultimo aggiornamento fatto (righe 33-35). Per terminare il ciclo, aggiorniamo la temperatura corrente T moltiplicandola per il cooling rate c, aumentiamo di 1 le iterazioni e passiamo così all'iterazione successiva (riga 36). Alla fine quando sarà stato raggiunto uno dei tre criteri di stop dell'algoritmo, allora la ALNS terminerà e restituirà la soluzione migliore trovata (riga 38).

5.2 Euristiche di rimozione di richieste

In questa sezione vengono descritte le euristiche di rimozione delle richieste. Tutte queste euristiche accettano una soluzione e un intero q come input. L'output restituito è una soluzione in cui un numero di richieste q sono state rimosse. Inoltre, alcune di queste euristiche hanno un parametro p che determina il grado di randomizzazione nell'operazione.

5.2.1 Rimozione di Shaw

Questa euristica di rimozione è stata proposta da Shaw (1997, 1998). Poniamo attenzione riguardo al fatto che questo algoritmo è stato leggermente modificato per adattarsi al problema di VRPPDTW. L'idea generale è quella di rimuovere le richieste in qualche modo simili tra di loro, poiché ci aspettiamo che sia ragionevolmente facile mescolare richieste simili e quindi di creare nuove (forse anche migliori) soluzioni. Se scegliamo di rimuovere richieste molto diverse tra loro, potremmo non guadagnare nulla nel reinserire le richieste perché potremmo essere in grado di inserirle solo nelle loro posizioni originali o in certe posizioni errate.

Definiamo la somiglianza di due richieste i e j utilizzando una misura di relazione R_{ij}. Più basso è R_{ij}, più correlate sono le due richieste. La misura di correlazione utilizzata in questa sezione è costituita da quattro termini: un termine di distanza, un termine di tempo, un termine di capacità e un termine che considera i veicoli che possono essere utilizzati per soddisfare le due richieste. Questi termini vengono pesati utilizzando rispettivamente i pesi ϕ, χ, ψ e ω. La misura di parentela è data da:

$$R_{i,j} = \phi(c_{A(i),A(j)} + c_{B(i),B(j)}) + \chi(|T_{A(i)} - T_{A(j)}| + |T_{B(i)} - T_{B(j)}|) + \psi|d_i - d_j| + \omega(\frac{|K_i \cap K_j|}{min\{|K_i|, |K_j|\}}) \tag{5.1}$$

$A(i)$ e $B(i)$ indicano i luoghi di ritiro e consegna della richiesta i e T_i indica l'ora in cui viene visitata la posizione i. c_{ij} , d_i e K_i sono definiti in 2.2. Il termine ponderato ϕ misura la distanza, il termine ponderato χ misura la connessione temporale, il termine ponderato ψ confronta la domanda di capacità delle richieste e il termine ponderato ω assicura che due richieste ottengano una misura di correlazione elevata se solo pochi veicoli o nessun veicolo è in grado di soddisfare entrambe le richieste. Si assume che c_{ij} , T_x e d_i siano normalizzati in modo tale che $0 \leq R(i,j) \leq 2(\phi + \chi) + \psi + \omega$. Questo viene fatto scalando c_{ij}, T_x e d_i in modo che assumano solo valori in [0,1]. Si noti che non possiamo calcolare R_{ij}, se la richiesta i o j non appartengono ad alcuna rotta.

La relazione di parentela viene utilizzata per rimuovere le richieste nello stesso modo descritto da Shaw (1997). La procedura per rimuovere le richieste è mostrata in pseudocodice nell'algoritmo (5). La procedura seleziona inizialmente una richiesta casuale da rimuovere e nelle iterazioni successive quindi sceglie richieste simili alle richieste già rimosse. Un parametro di determinismo $p \geq 1$ introduce una certa casualità nella selezione delle richieste (un valore basso di p corrisponde a molta casualità).

Algorithm 5 Shaw Removal

procedure SHAWREMOVAL(s, q, p)
  ▷ s è una soluzione ottenuta, $q \in \mathbb{N}$, $p \in \mathbb{R}_+$
  richiesta: r = una richiesta selezionata casualmente da s;
  insieme di richieste: $D = \{r\}$;
  while $|D| < q$ **do**
    r = richiesta selezionata randomicamente da D;
    L = array contenente tutte le richieste da s che non sono in D;
    ordina L tale che $i \leq j \implies R(r, L[i]) \leq R(r, L[j])$;
    scegli un numero casuale y nell'intervallo [0,1);
    $D = D \bigcup \{L[y^p|L|]\}$;
  end while
  rimuovi tutte le richieste in D da s;
end procedure

Nella figura 5.1 viene mostrato un esempio di utilizzo di questa euristica di rimozione di richieste. Dall'immagine vengono mostrate due rotte differenti e inizialmente viene scelta casualmente la richiesta 2, che verrà eliminata dalla rotta di appartenenza e verrà aggiunta all'insieme delle richieste rimosse (l'insieme L presente a sinistra). Il turno successivo verrà calcolato il "valore di parentela" tra la richiesta 2 e tutte le altre richieste presenti nelle rotte. Nel caso di utilizzo di questa ricerca verranno considerati solo i "valori di parentela" riguardanti la distanza e l'appartenenza allo stesso percorso. Si deduce quindi che per quanto ci siano delle richieste con distanza minore sulla rotta 1, verrà rimossa la richiesta 1 (appartenente alla rotta 2) essendo sullo stesso percorso e, quindi, avendo un'influenza maggiore per questo motivo. Al turno successivo si ripeterà lo stesso passaggio valutando il "valore di parentela" di tutte le richieste presenti nelle rotte rispetto ad una richiesta scelta casualmente tra 1 e 2, trovandosi entrambe nelle richieste già rimosse.

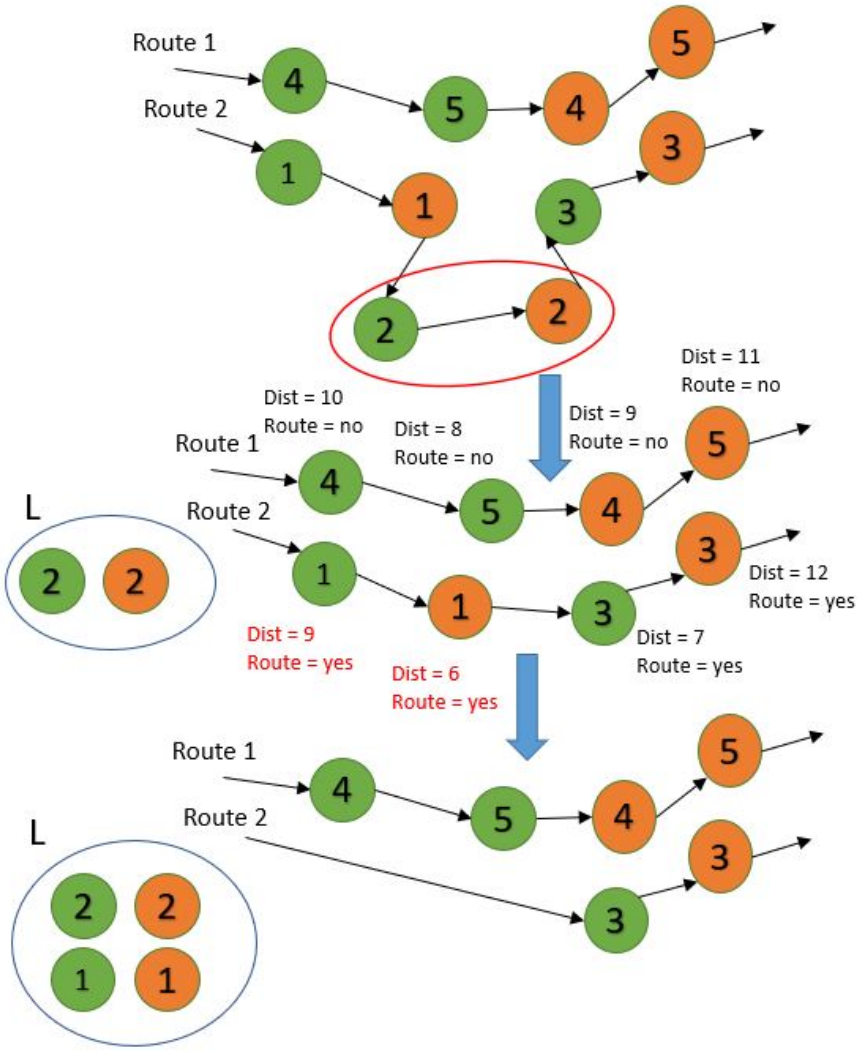

Figura 5.1: Esempio di una applicazione dell'euristica di rimozione delle richieste di Shaw.

5.2.2 Rimozione casuale

L'algoritmo di rimozione casuale seleziona semplicemente q richieste a caso e le rimuove dalla soluzione. Questa euristica può essere vista come un caso speciale dell'euristica della rimozione di Shaw (vedi 5.2.1) con $p = 1$. Tuttavia, viene definita separatamente, essendo implementata per funzionare più velocemente dell'euristica di rimozione di Shaw.

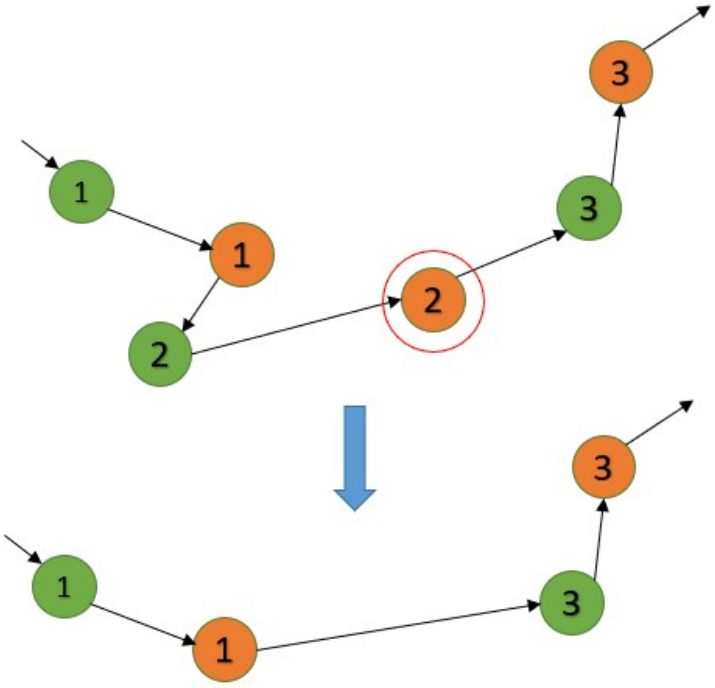

Figura 5.2: Esempio di una applicazione dell'euristica di rimozione casuale delle richieste.

Nella figura 5.2 viene mostrato un esempio di utilizzo di questa euristica di rimozione delle richieste. Dalla rotta viene rimossa una richiesta di prelievo o deposito casuale, verrà scelta casualmente la richiesta 2 di deposito che verrà tolta dalla rotta. Quindi, di conseguenza, verrà rimossa anche la richiesta 2 di prelievo.

5.2.3 Rimozione del peggior elemento

Data una richiesta servita da qualche veicolo in una soluzione s, definiamo il costo di richiesta come $cost(i, s) = f(s) - f_{-i}(s)$ dove $f_{-i}(s)$, che sarebbe il costo della soluzione senza la richiesta i. Sembra ragionevole provare a rimuovere le richieste con un costo elevato e inserirle in un altro punto della soluzione per ottenere un valore di soluzione migliore. Pertanto, proponiamo un'euristica di rimozione che rimuove le richieste con alti $cost(i, s)$. L'euristica di rimozione del peggior elemento è mostrata nel pseudocodice dell'algoritmo 6. Si noti che la rimozione è randomizzata, dove il grado di randomizzazione è controllato dal parametro p come nelle Sezioni 5.2.1 e 5.2.2. Questo viene fatto per evitare situazioni in cui le stesse richieste vengono rimosse più volte.

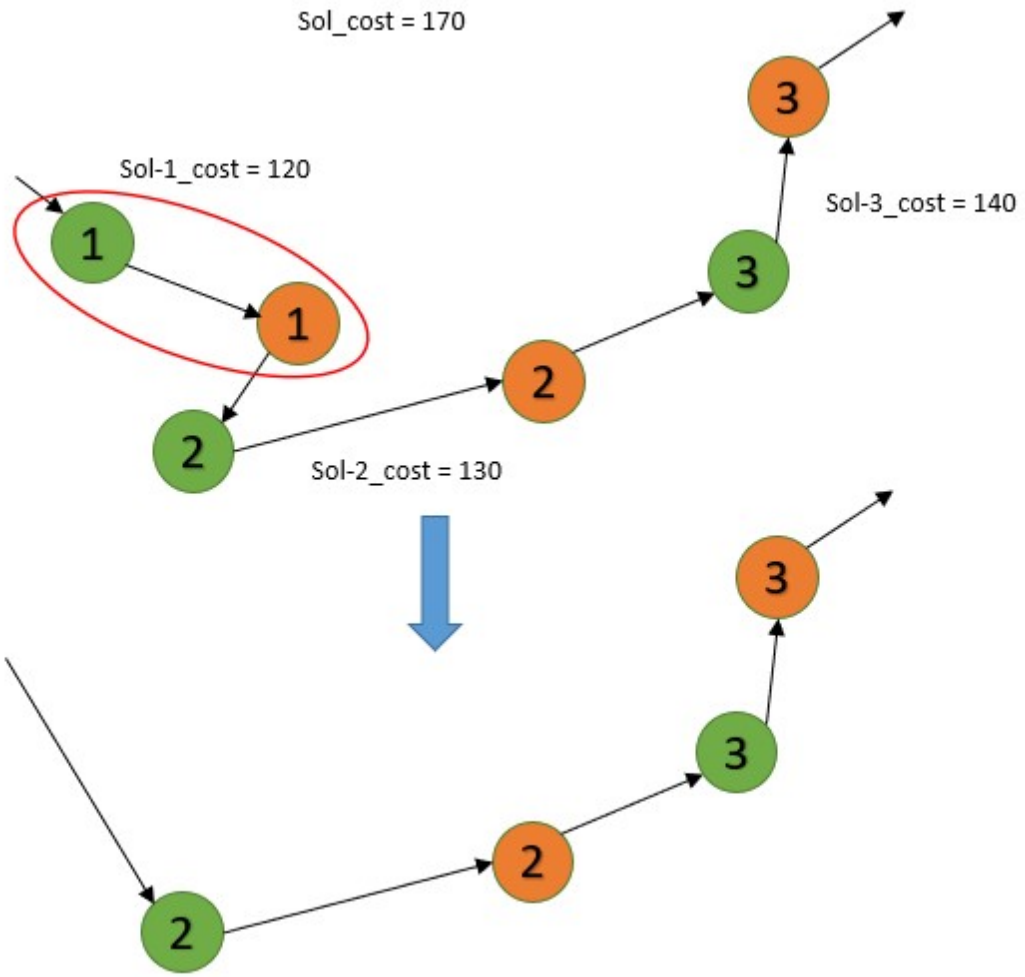

Figura 5.3: Esempio di una applicazione dell'euristica di rimozione delle richieste del peggior elemento.

Nella figura 5.3 viene mostrato un esempio di utilizzo di questa euristica di rimozione delle richieste. Dall'immagine si può vedere che il costo totale della soluzione è di 170, il costo della soluzione senza la richiesta 1 è di 120, il costo della soluzione senza la richiesta 2 è di 130 e il costo della soluzione senza la richiesta 3 è di 140. Quindi per la differenza tra il costo totale e quello della richiesta rimossa, avremo che il costo per la richiesta 1 sarà di 50, per la richiesta 2 il costo sarà di 40 e per la richiesta 3 il costo sarà di 30, quindi verrà rimossa la richiesta 1 in quanto quella con la differenza di costo più alta.

Algorithm 6 Worst Removal

```
1: procedure WorstRemoval(s, q, p)
2:     ▷ s è una soluzione ottenuta, q ∈ ℕ, p ∈ ℝ+
3:     while q > 0 do
4:         L = array richieste pianificate i, decrescente in base al valore cost(i, s);
5:         scegli un numero casuale y nell'intervallo [0,1);
6:         richiesta: r = L[y^p |L|];
7:         rimuovi r dalla soluzione s;
8:         q = q − 1;
9:     end while
10: end procedure
```

5.3 Euristiche di inserimento di richieste

Le euristiche di inserimento delle richieste per i problemi di instradamento dei veicoli sono tipicamente divise in due categorie: euristiche di inserimento sequenziale e parallela. La differenza tra le due classi è quella che l'euristica sequenziale costruisce un percorso alla volta, mentre l'euristica parallela costruisce più percorsi contemporaneamente. Le euristiche trattate successivamente sono tutte del tipo parallele. Le euristiche di inserimento delle richieste qui proposte verranno utilizzate in un ambiente in cui vengono forniti un numero di percorsi parziali e un numero di richieste di inserimento che raramente costruiscono la soluzione da zero.

5.3.1 Inserimento greedy di base

L'euristica greedy di base è una semplice costruzione euristica. Esegue al massimo n iterazioni poiché inserisce una richiesta ad ogni iterazione. Denotiamo con $\Delta f_{i,k}$ la variazione di valore obiettivo subita inserendo la richiesta i nel percorso k nella posizione che aumenta minormente il valore della soluzione. Se non possiamo inserire la richiesta i nel percorso k, allora impostiamo $\Delta f_{i,k} = \infty$. Definiamo quindi c_i come $c_i = min_{k \in K} \Delta f_{i,k}$, dove c_i è il costo di inserimento della richiesta i nella sua migliore posizione complessiva, che indichiamo con la posizione di costo minimo. Infine, scegliamo la richiesta i che minimizza

$$\min_{i \in U} c_i \tag{5.2}$$

e la inseriamo nella posizione di costo minimo. U è definito come l'insieme delle richieste non pianificate. Questo processo continua fino a quando tutte le richieste sono state inserite o non è possibile inserire altre richieste.

Osserviamo che ad ogni iterazione cambiamo solo un percorso (quello in cui abbiamo inserito la richiesta) e che non dobbiamo ricalcolare i costi di inserimento in tutti gli altri percorsi. Questa proprietà viene utilizzata nell'implementazione concreta per velocizzare l'euristica di inserimento.

Un problema ovvio che sorge con questa euristica è che questa spesso rimanda il posizionamento delle richieste *hard* (che sarebbero costose da inserire, ovvero con grandi valori di c_i) alle ultime iterazioni in cui non hanno molte opportunità di inserimento perché molti dei percorsi sono "pieni". L'euristica presentata nella sezione successiva cerca di aggirare questo problema.

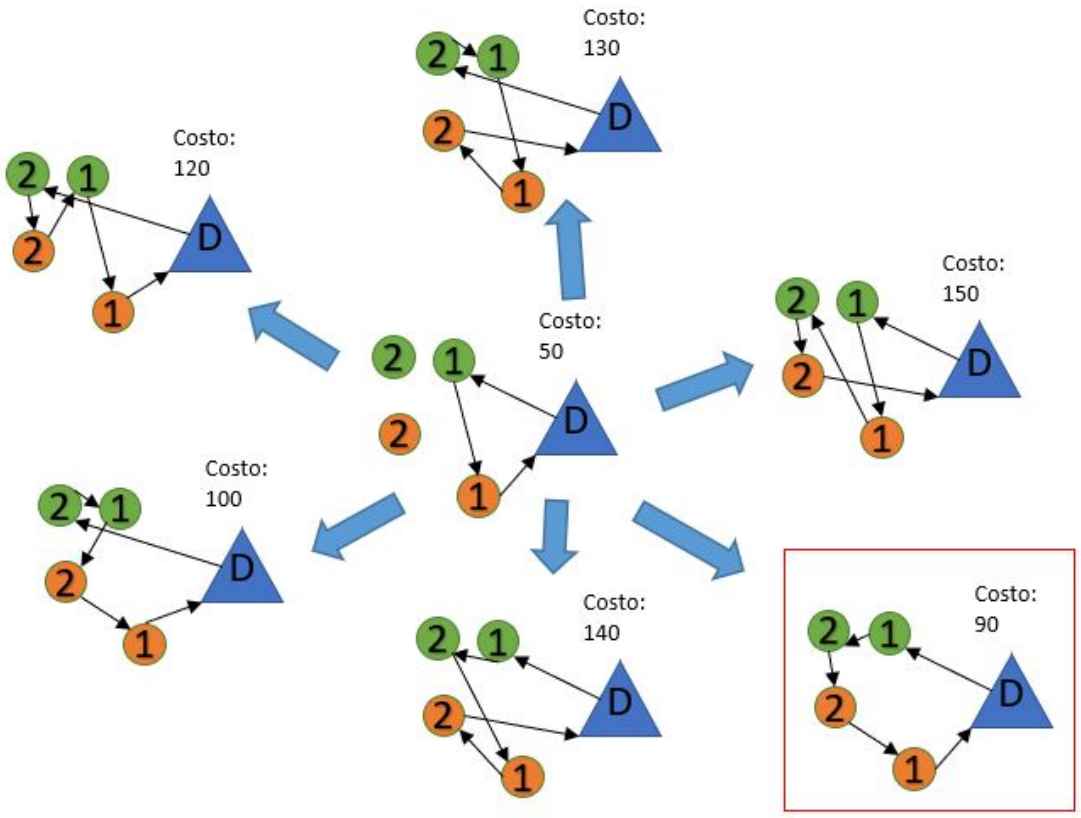

Figura 5.4: Esempio di una applicazione dell'euristica di inserimento greedy di base delle richieste.

Nella figura 5.4 viene mostrato un esempio di utilizzo di questa euristica di inserimento delle richieste. Nell'esempio mostrato precedentemente, si può vedere che la situazione iniziale è quella al centro dell'immagine e per semplificare l'algoritmo è già stata presa la richiesta 2 da aggiungere. Quest'ultima verrà aggiunta in modo che i nodi abbiano un impatto minore sulla soluzione. Da come si può

vedere vengono provate tutte le combinazioni di inserimento e verrà scelta quella col costo minore che impatterà sulla soluzione.

5.3.2 Inserimento regret

L'euristica regret cerca di migliorare l'euristica greedy di base incorporando una sorta di informazione preventiva quando si seleziona la richiesta da inserire. Sia $x_{ik} \in \{1, \ldots, m\}$ una variabile che indica il percorso per il quale la richiesta i ha il k-esimo costo di inserzione più basso, ovvero $\Delta f_{i,x_{i,k}} \leq \Delta f_{i,x_{i,k'}}$ per $k \leq k'$. Usando questa notazione, possiamo esprimere c_i dalla sezione 5.3.1 come $c_i = \Delta f_{i,x_{i,k}}$. Nell'euristica regret definiamo un valore di regret $c_i^\star$ come $c_i^\star = \Delta f_{i,x_{i2}} - \Delta f_{i,x_{i1}}$. In altre parole, il valore di regret corrisponde alla differenza del costo di inserimento della richiesta nel suo posto migliore nel percorso e nel suo secondo posto migliore nel percorso. In ogni iterazione l'euristica regret sceglie di inserire la richiesta i che massimizza

$$\max_{i \in U} c_i^\star \tag{5.3}$$

La richiesta viene inserita nella sua posizione di costo minimo. I pareggi vengono risolti selezionando l'inserimento con il costo più basso. Informalmente, scegliamo l'inserimento di cui ci "pentiremo" di più se non venisse fatto subito.

L'euristica può essere estesa in modo naturale per definire una classe di euristiche regret: l'operazione k-regret è quella di costruzione che in ogni fase sceglie di inserire la richiesta i che massimizza:

$$\max_{i \in U} \{\sum_{j=1}^{k} \Delta(f_{i,x_{ij}} - \Delta f_{i,x_{i1}})\} \tag{5.4}$$

Se alcune richieste non possono essere inserite in almeno $m - k + 1$ percorsi, viene inserita la richiesta che può essere aggiunta nel minor numero di percorsi (ma può comunque essere inserita in almeno un percorso). I pareggi vengono risolti selezionando la richiesta con il miglior costo di inserzione. La richiesta viene inserita nella sua posizione di costo minimo. L'euristica regret presentata precedentemente in questa sezione è un'euristica 2-regret e quella di inserimento greedy di base della Sezione 5.3.1 è un'euristica 1-regret a causa delle regole di spareggio. L'operazione con $k > 2$ ricerca il costo di inserimento di una richiesta sulle k rotte migliori e inserisce la richiesta la cui differenza di costo tra l'inserimento nella migliore rotta e le $k - 1$ migliori rotte è maggiore. Rispetto a un'euristica di 2-regret, le euristiche regret con grandi valori di k scoprono in precedenza che le possibilità di inserimento di una richiesta diventano limitate.

L'euristica regret è stata utilizzata da Potvin e Rousseau (1993) [31] per il problema di VRPTW. Nel loro articolo può essere classificata come un'euristica k-regret con $k = m$, perché tutti i percorsi sono considerati in un'espressione simile a (5.4). Nell'articolo precedente non viene utilizzata la variazione del valore oggettivo per valutare il costo di un inserimento, ma utilizzano una funzione di costo speciale. Essa può essere utilizzata anche per problemi di ottimizzazione combinatoria al di fuori del dominio di routing dei veicoli. Un esempio di applicazione al problema di assegnazione generalizzato viene descritto da Martello e Toth (1981). Come nella sezione precedente, cambiamo solo un percorso ad ogni iterazione per accelerare l'euristica regret.

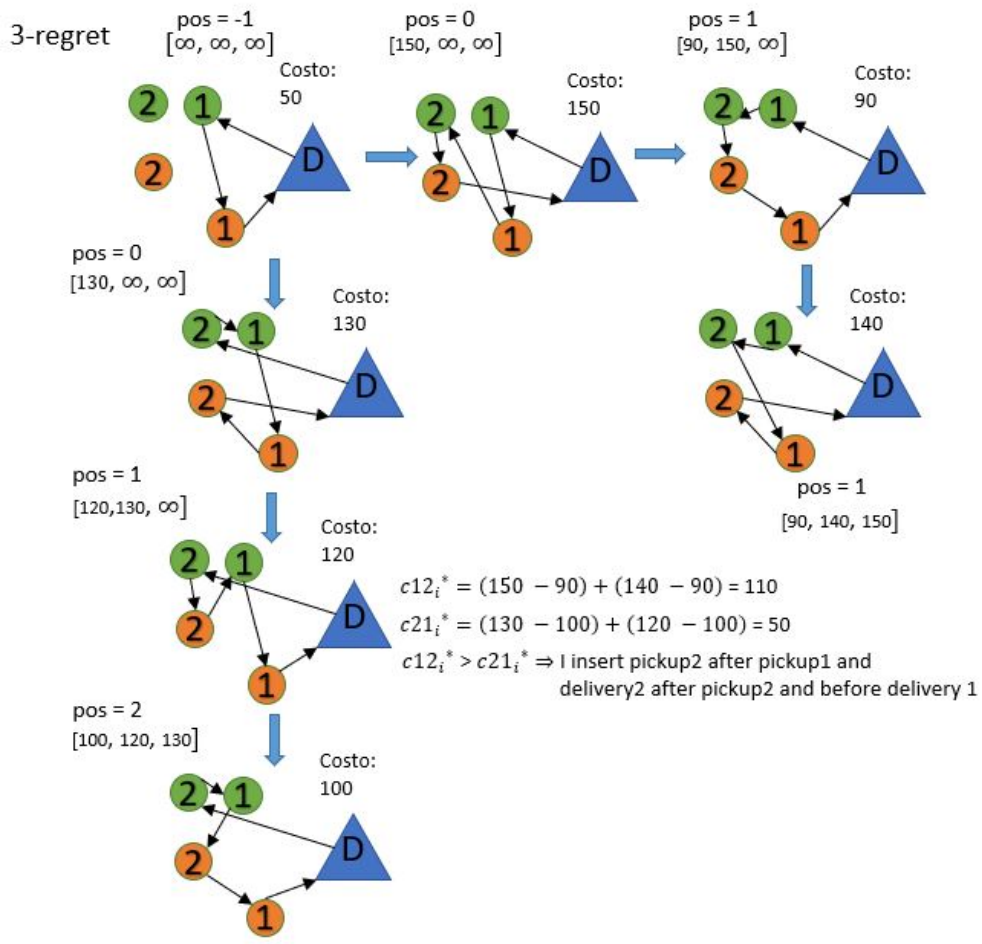

Figura 5.5: Esempio di una applicazione dell'euristica di inserimento regret delle richieste.

Nella figura 5.5 viene mostrato un esempio di utilizzo di questa euristica di

inserimento delle richieste. In questo esempio viene utilizzato il tipo di inserimento 3-regret che salverà i 3 migliori costi di inserimento di ogni richiesta. In questo caso si può vedere che la situazione iniziale è quella definita in alto a sinistra e per semplificare l'algoritmo è già stata presa la richiesta 2 da aggiungere. A questo punto si vedrà quale impatto può avere l'inserimento della richiesta 2 dopo la richiesta 1 (nella figura è definita dalla ramificazione che avviene verso destra) e la richiesta 2 prima della richiesta 1 (la ramificazione che avviene verso il basso). Si tengono salvati tutti i costi in ordine crescente e verrà memorizzata, inoltre, qual è la soluzione che viene restituita col costo minore in ciascuna ramificazione. Infine verranno salvate le migliori 3 soluzioni e per ogni ramificazione verrà sommata la differenza tra la seconda migliore e la migliore e la differenza tra la terza migliore e la migliore. Quindi si prenderà la somma maggiore tra quelle ottenute (ovvero la ramificazione che va verso destra) e così si potrà inserire la richiesta corrispondente alla migliore nella ramificazione scelta. In questo caso si sceglierà di aggiungere il pickup 2 dopo il pickup 1 e il delivery 2 dopo il pickup 2 e prima del delivery 1, in quanto ha la somma maggiore dei delta (110) ed è quella con costo minore nella ramificazione (90).

5.4 Euristiche di rimozione di aree di sosta

In questa sezione vengono descritte le euristiche di rimozione delle aree di sosta. Tutte accettano una soluzione e un intero q come input. L'output delle euristiche è una soluzione in cui un numero di richieste q sono state rimosse. Visto che nella soluzione ci potrebbero essere dei doppioni della stessa area di sosta, quando l'algoritmo ne sceglierà una, rimuoverà solo quel clone e non tutte le altre aree di sosta presenti nelle rotte che identificano la stessa.

5.4.1 Rimozione casuale

L'algoritmo di rimozione casuale seleziona semplicemente q aree di sosta casuali presenti nelle rotte e le rimuove dalla soluzione.

Nella figura 5.6 viene mostrato un esempio di utilizzo di questa euristica di rimozione delle aree di sosta. Nell'immagine si può vedere che nella rotta verrà scelto casualmente il parcheggio 2 e questo verrà rimosso da essa e quindi dalla soluzione.

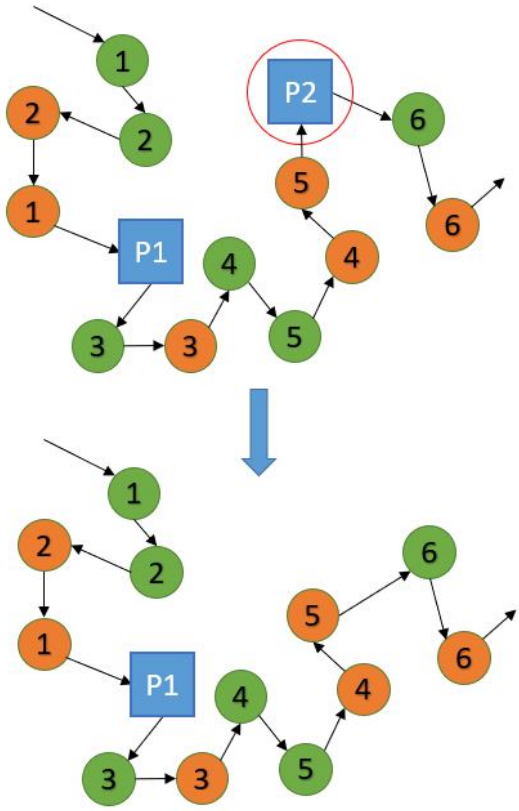

Figura 5.6: Esempio di una applicazione dell'euristica di rimozione casuale delle aree di sosta.

5.4.2 Rimozione del peggior elemento

La rimozione del peggior elemento delle aree di sosta prende tutte le zone di sosta presenti nelle rotte e le ordina decrescentemente per costo peggiore (come definito per le richieste nella sezione 5.2.3), ovvero calcola per ogni parcheggio la differenza tra il costo della funzione totale e il costo della funzione senza la presenza di quel nodo. Questo significa che la differenza maggiore rappresenterà il costo del parcheggio che impatta più negativamente sulla soluzione, quindi verrà rimosso quello.

Nella figura 5.7 viene mostrato un esempio di utilizzo di questa euristica di rimozione delle aree di sosta. Dall'immagine si può vedere che il costo totale della funzione è 430, il costo della soluzione senza il parcheggio 1 è di 320 e il costo della soluzione senza il parcheggio 2 è di 390. Quindi per differenza tra il costo della soluzione e il costo della soluzione senza l'area di sosta nella rotta, avremo che senza l'area di sosta 1 il costo sarà di 110 e che senza l'area di sosta 2 il costo sarà

di 40, perciò verrà rimossa l'area di sosta 1 in quanto sarà quella con la differenza di costo più alta.

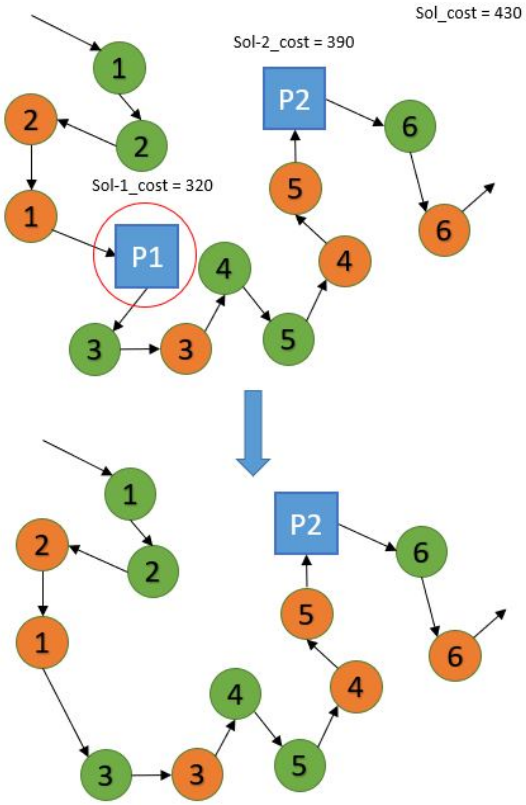

Figura 5.7: Esempio di una applicazione dell'euristica di rimozione del peggior elemento delle aree di sosta.

5.5 Euristiche di inserimento di aree di sosta

Per tutti gli algoritmi di inserimento delle aree di sosta, si tiene conto che l'aggiunta del nodo avviene nel punto in cui la soluzione peggiora gravemente, ovvero dove l'autista va oltre il suo tempo massimo di guida. Quindi si prendono tutte i subtours presenti tra un deposito e l'altro, tra un parcheggio e un deposito, tra un deposito e un parcheggio e tra un parcheggio e un parcheggio per valutare se si può inserire un'area in quella zona. Negli esempi successivi viene delineato con un cerchio marcato di rosso con spessore dove la soluzione grava.

5.5.1 Inserimento del migliore elemento

L'inserimento del miglior elemento delle aree di sosta valuta l'aggiunta di una tra le aree di sosta disponibili in un subtour, partendo dal nodo in cui la situazione grava e da quel punto valuta andando indietro fino all'inizio del subtour dove posizionare le aree si sosta e lo inserisce nel posto che restituisce il costo minore nella soluzione.

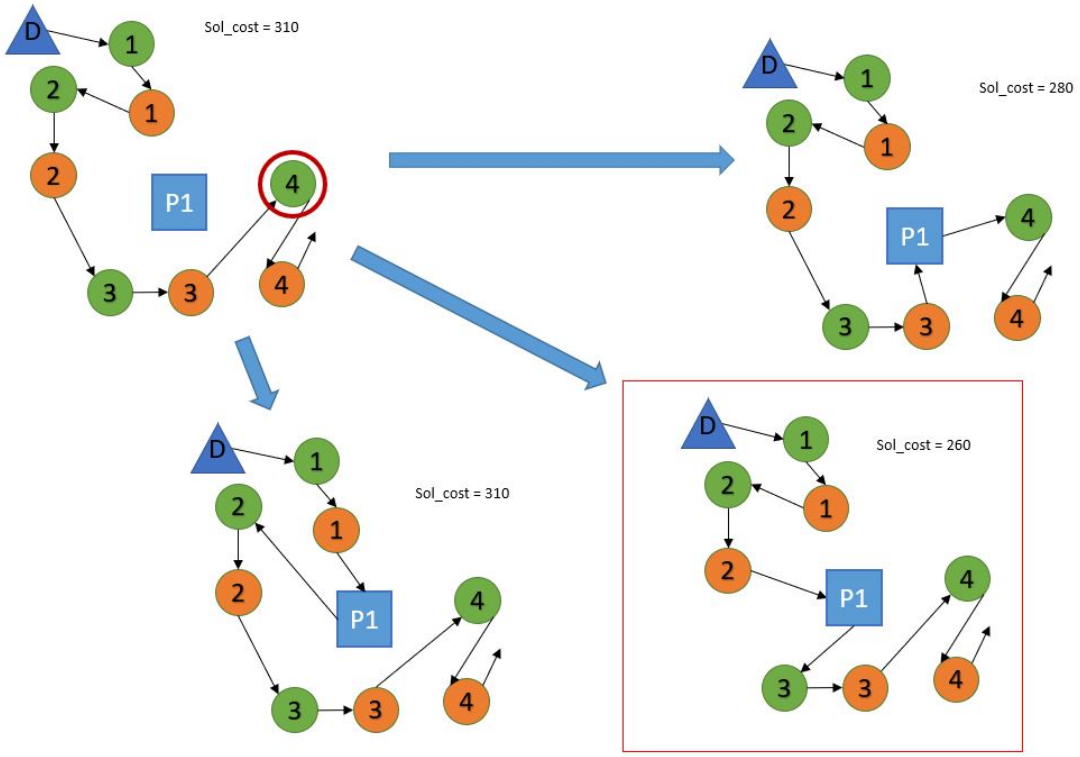

Figura 5.8: Esempio di una applicazione dell'euristica di inserimento del miglior elemento delle aree di sosta.

Nella figura 5.8 viene mostrato un esempio di utilizzo di questa euristica di inserimento delle aree di sosta. Dall'immagine si può vedere che il subtour parte dal deposito e l'autista va oltre il suo massimo tempo di guida alla richiesta 4, quindi tornando indietro gli unici posti dove si potrà inserire l'area di sosta sarà tra le richieste 4 e 3, tra le richieste 3 e 2 e tra le richieste 2 e 1. Supponiamo di aggiungere solamente l'area di sosta 1 non considerando le altre aree e il deposito. Quindi si inserirà l'area di sosta 1 in tutte le posizioni elencate precedentemente e si selezionerà quella che impatta minormente sulla soluzione. Si nota che inserendolo tra le richieste 2 e 3, avremo costo pari a 260, ovvero il minore tra tutti quelli disponibili.

5.5.2 Inserimento greedy di base

L'inserimento greedy di base delle aree di sosta valuta l'aggiunta di una tra le zone di sosta disponibili in un subtour, partendo dal nodo in cui la situazione grava e da quel punto valuta andando indietro il primo posto disponibile, e inserisce l'area di sosta che riduce maggiormente la soluzione in quel punto.

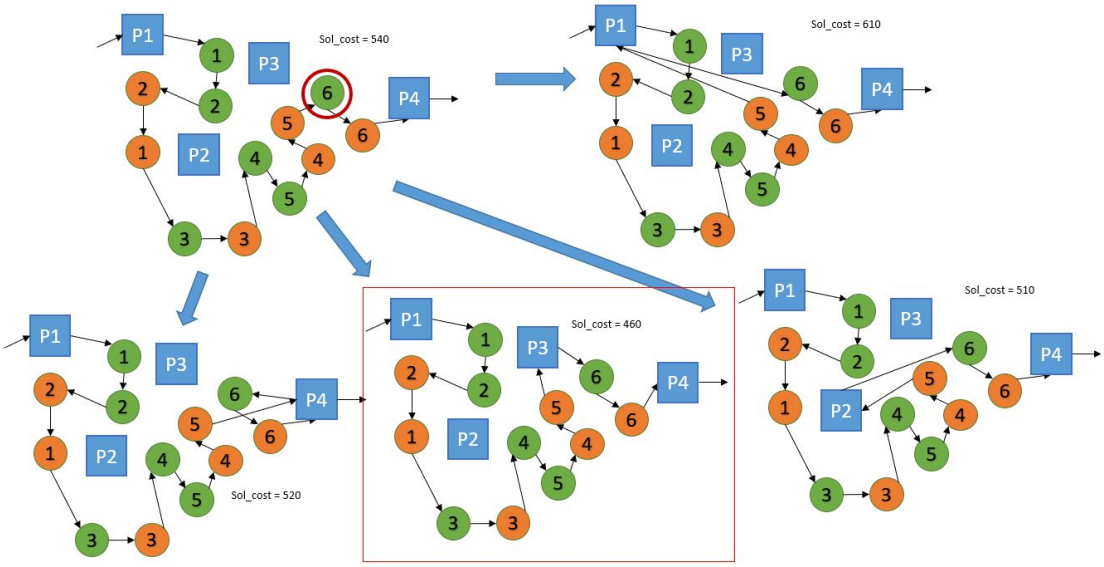

Figura 5.9: Esempio di una applicazione dell'euristica di inserimento greedy di base delle aree di sosta.

Nella figura 5.9 viene mostrato un esempio di utilizzo di questa euristica di inserimento delle aree di sosta. Dall'immagine si può vedere che nella rotta tra l'area di sosta 1 e l'area di sosta 4 la situazione grava alla richiesta 6, quindi verrà scelto il primo posto disponibile dove inserire l'area di sosta (ovvero tra la richiesta di delivery 5 e la richiesta di pickup 6) ed aggiungerà tutte le aree di sosta disponibili (ovvero le aree di sosta 1, 2, 3 e 4). Quindi sceglierà quella che ridurrà maggiormente il costo totale della soluzione. In questo caso l'inserimento dell'area di sosta 3 sarà quella che la diminuirà maggiormente. Nel caso ci fosse stata una situazione in cui tutti i costi delle soluzioni fossero stati maggiori, allora sarebbe stato valutato un nuovo punto precedente della rotta in cui inserire l'area di sosta.

5.5.3 Inserimento greedy con comparazione

La riparazione greedy con comparazione delle aree di sosta valuta l'inserimento di una tra le zone di sosta disponibili in un subtour partendo dal nodo in cui la situazione grava e da quel punto valuta andando indietro il primo posto disponibile

e lo compara con il secondo posto disponibile (quello successivo al primo), quindi inserisce il parcheggio nel punto migliore che riduce maggiormente la soluzione comparando le due zone. Come nel caso precedente se il costo della soluzione fosse maggiore dopo avere comparato gli inserimenti in quelle due zone, allora l'algoritmo valuterebbe i due posti precedenti dove è possibile inserire un'area di sosta.

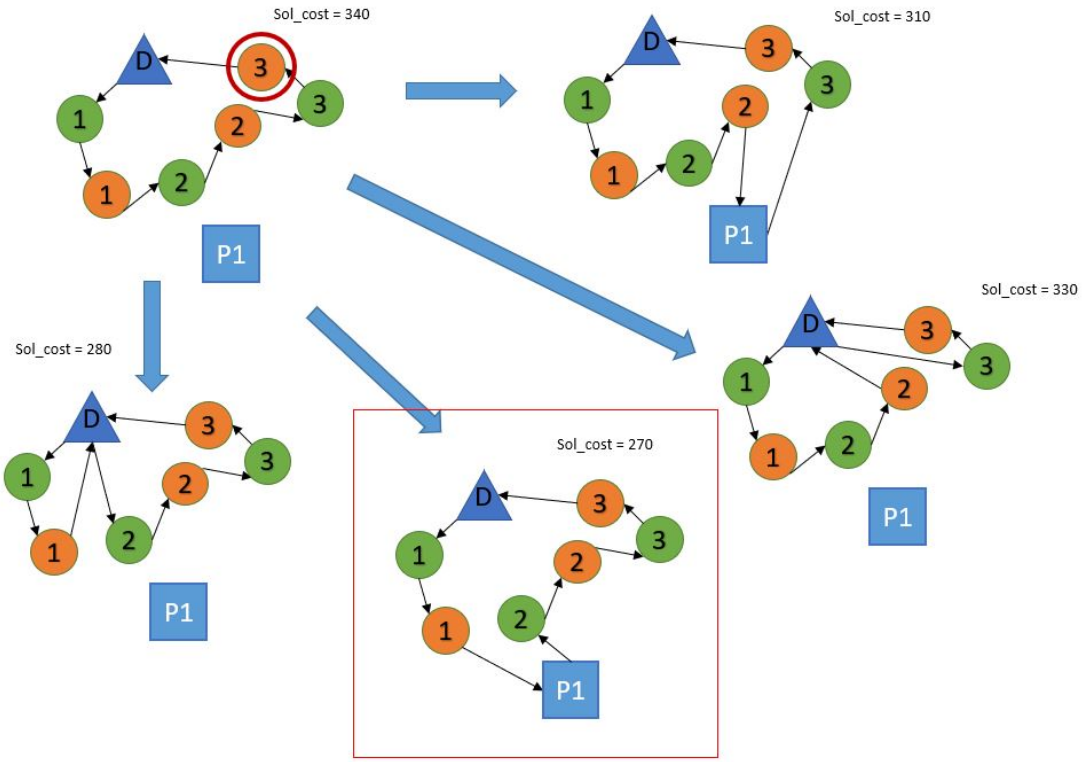

Figura 5.10: Esempio di una applicazione dell'euristica di inserimento greedy con comparazione delle aree di sosta.

Nella figura 5.10 viene mostrato un esempio di utilizzo di questa euristica di inserimento delle aree di sosta. Dall'immagine si può vedere un subtour che parte e ritorna allo stesso deposito, ma nel tratto la situazione grava quando raggiunge la richiesta 3. Quindi l'algoritmo prova a inserire l'area di sosta 1 e il deposito (che vale come area di sosta) nella prima posizione disponibile, ovvero tra le richieste 3 e 2 e lo compara con quella precedente, ovvero tra le richieste 2 e la 1. In seguito valuterà tutti i costi totali delle soluzioni ottenute e alla fine inserirà l'area di sosta 1 tra la richiesta 2 e 1, in quanto sarà quella con il costo di soluzione più basso tra tutte le possibilità.

5.6 Finestre temporali di prenotazione

Nel problema posto, le finestre temporali di richiesta di un cliente non hanno solamente un inizio e una fine come usualmente viene affrontato in letteratura, ma vengono divise sia per il prelievo che per il deposito delle persone in tre differenti momenti: l'ora in cui l'utente vuole essere prelevato dal veicolo ($StartTimeWindowBegin$), l'ora di minima di tolleranza entro la quale vuole anticipare il prelievo ($StartTimeRequest$, che è sempre minore o uguale al tempo di $StartTimeWindowBegin$) e l'ora massima in cui vuole essere prelevato ($StartTimeWindowFinish$, che è sempre maggiore o uguale al tempo di $StartTimeWindowBegin$). Prima dell'ora minima di tolleranza e dopo l'ora massima di prelievo, non è accettabile che il veicolo si presenti a prendere la persona.

Lo stesso ragionamento che viene fatto precedentemente, viene realizzato anche per la prenotazione del deposito dello stesso cliente. Si definiscono, quindi, l'ora in cui l'utente vuole essere depositato dal veicolo ($EndTimeWindowBegin$), l'ora di minima di tolleranza fino alla quale vuole anticipare il deposito ($EndTimeRequest$, che è sempre minore o uguale al tempo di $EndTimeWindowBegin$) e l'ora massima in cui vuole essere depositato ($EndTimeWindowFinish$, che è sempre maggiore o uguale al tempo di $EndTimeWindowBegin$).

Per questo motivo, sono state definite delle aree all'interno delle quali si ha un peso che influisce in un aumento del costo della soluzione quando non si soddisfa la richiesta del cliente. Queste aree sono definite proprio come in letteratura:

- $Infeseability$: tempo per cui si è al di fuori dell'area di tolleranza della richiesta del cliente, quindi il peso del costo sarà linearmente molto crescente per ogni secondo che si arriverà prima od oltre all'ora minima e massima di prenotazione.

- $Earliness$: tempo per cui si è in orario rispetto al tempo richiesto dal cliente (quindi si è in anticipo rispetto alla richiesta del cliente). Questa avrà un costo linearmente poco crescente per ogni secondo prima dell'effettiva ora di prenotazione del cliente.

- $Tardiness$: tempo per cui si è in ritardo rispetto al tempo richiesto dal cliente (quindi si è in ritardo rispetto alla richiesta del cliente). Questa avrà un costo lineare poco crescente per ogni secondo dopo l'effettiva ora di prenotazione del cliente.

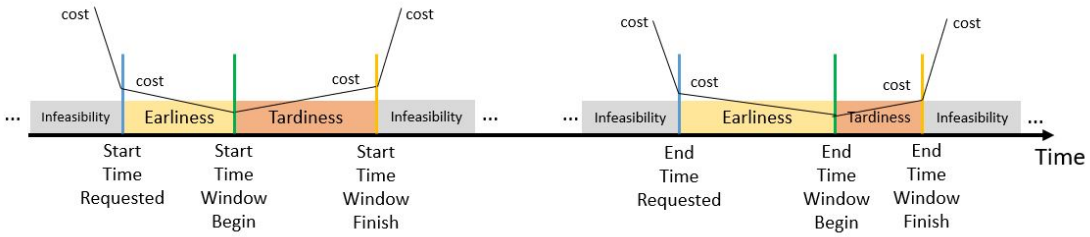

Figura 5.11: Grafico temporale delle finestre temporali del problema affrontato e delle sue aree di definizione. Vengono considerate le aree dove si è fuori delle soglie di tolleranza (infeasibility), l'area di anticipo (earliness) e quella di ritardo (tardiness) rispetto all'orario di prenotazione. Dalle considerazioni fatte precedentemente, si può notare che più ci si allontana dal tempo richiesto dal cliente (linea verde), più aumenta linearmente il costo della soluzione finale, ma se si va oltre al tempo minimo (linea blu) oppure oltre al tempo massimo (linea arancio) di richiesta del cliente, il costo della soluzione finale aumenterà linearmente molto più velocemente.

Nella figura 5.11 viene riportato un grafico temporale nel quale si possono vedere le aree di definizione di $infeasibility$, *earliness* e *tardiness* rispetto alle finestre temporali, sia per il prelievo che per il deposito.

5.7 Vincoli non violabili

La formulazione del modello (assumendo la sua correttezza) ci porta a definire un insieme di vincoli che possano restituire in output delle soluzioni ammissibili. In questa sezione vengono presentati tutti quei vincoli che non possono essere assolutamente violati (comunemente detti anche vincoli *hard*).

I vincoli che sono *hard* per le soluzioni restituite dalla ALNS personalizzata, sono i seguenti:

- Il veicolo non può superare la sua capienza totale: quando si prelevano dei clienti, non vogliamo che si ecceda il numero massimo di posti consentiti dal veicolo (ovvero la sua capacità massima), altrimenti si incorrerebbe nel problema di non aver spazio per far salire qualcuno. Quindi dall'inizio alla fine di ogni rotta, un veicolo non dovrà mai eccedere la capienza massima.

- Il veicolo quando effettua una sosta, non deve avere passeggeri a bordo: l'autista dopo aver guidato entro un massimo di tempo, dovrebbe fermarsi in una area di sosta per riposare. Quando viene effettuata questa operazione, dato che il conducente è in pausa per riposare, deve fermarsi con il mezzo vuoto

(non avendo ancora delle persone sopra). Questo significa che in ogni subtour quando preleva una persona, prima della sosta deve portarla a destinazione.

5.8 Vincoli violabili

A differenza della parte definita nella Sezione 5.7, in una soluzione ci potrebbero essere dei vincoli che possono essere tollerabili, ovvero sarebbero accettati anche se non fossero rispettati, ma per farlo ci sarà una penale da pagare tramite un punteggio pesato sulla soluzione finale.

I vincoli che sono *soft* per le soluzioni restituite dalla ALNS personalizzata, sono i seguenti:

- Ciascun cliente che ha prenotato una corsa deve essere servito, altrimenti ciascun utente non servito penalizzerà il costo totale della soluzione con un prezzo da pagare molto alto.
- Un cliente che viene prelevato e/o depositato in ritardo (quando ci troviamo in zona di *tardiness*) o in anticipo (quando ci troviamo in zona di *earliness*) rispetto al tempo da lui richiesto (ma non eccedendo i tempi di tolleranza minima e massima), penalizzerà il costo totale della soluzione con un prezzo da pagare basso per ogni secondo in eccesso.
- Un cliente che viene prelevato e/o depositato al di fuori dei tempi di minima e massima tolleranza (quando ci troviamo in zona di $infeseability$), penalizzerà il costo totale della soluzione con un prezzo da pagare alto per ogni secondo in eccesso.
- Se un autista eccede il limite massimo di tempo imposto per guidare, penalizzerà il costo totale della soluzione con un prezzo da pagare alto per ogni secondo di guida in eccesso.
- Se un autista non si ferma nella propria area di sosta per il minimo tempo consentito, penalizzerà il costo totale della soluzione con un prezzo da pagare alto per ogni secondo in difetto.

5.9 Definizione della soluzione iniziale

Prima di arrivare nel cuore della metauristica di ALNS, dove si procede iterativamente a trovare una soluzione, si deve trovare una soluzione iniziale dalla quale partire. Nel caso di questa tesi, è stata definita una soluzione iniziale personalizzata che trova una soluzione accettabile come primo passo dell'algoritmo.

Viene definito, in pseudocodice, l'algoritmo per trovare la soluzione iniziale in 7.

Algorithm 7 Find Initial Solution

```
1: procedure FindInitSol(reqs, depots, parks, vehs, maxTrip, minIdle)
2:     for each v ∈ vehs do
3:         dep = depot ∈ depots relativo al veicolo v; routes(v) = {dep, dep};
4:     end for
5:     reqs ordinata crescentemente tramite il campo StartTimeWindowBegin;
6:     while reqs ≠ ∅ do
7:         bestCost = +∞; bestRoute_i = −1; bestPck_i = −1; bestDlv_i = −1;
8:         (pck, dlv) = estraggo pickup e delivery della prima richiesta in reqs;
9:         for each route ∈ routes do                ▷ route è una copia della rotta
10:            if len(route) = 2 then
11:                route = {dep, pck, dlv, dep}
12:                if cost(routes) < bestCost then
13:                    bestCost = cost(routes); bestRoute_i = idx(route);
14:                    bestPck_i = 1; bestDlv_i = 2;
15:                end if
16:            else
17:                l = posizione ultima area sosta o, in mancanza, deposito iniziale;
18:                dep_i = indice del deposito finale nella rotta route;
19:                for each p_i ∈ {l + 1, ..., dep_i}, d_i ∈ {l + 2, ..., dep_i}, p_i < d_i do
20:                    route(p_i) = pck; route(d_i) = dlv;
21:                    if cost(routes) < bestCost then
22:                        bestCost = cost(routes); bestRoute_i = idx(route)
23:                        bestPck_i = p_i; bestDlv_i = d_i;
24:                    end if
25:                end for
26:            end if
27:        end for
28:        bR = routes(bestRoute_i); subR = bR(lastPark_i||depIniz_i, ..., depFin_i)
29:        bR(bestPck_i) = pck; bR(bestDlv_i) = dlv;
30:        if time(subR) > maxTrip then
31:            bR = bR \ {pck, dlv};
32:            bestPark = area sosta con costo minore in pos. prima del dep. fin.;
33:            bR(len − 1) = bestPark
34:        else
35:            reqs = reqs \ (pck, dlv)
36:        end if
37:    end while
38:    si rimuovono tutte le aree di sosta presenti prima di un dep. fin. nelle rotte;
39:    return routes
40: end procedure
```

Analizziamo il codice precedente, partendo dalla descrizione dei parametri in input dell'algoritmo:

- *reqs* corrisponde alla lista delle richieste di tutti i clienti;
- *depots* corrisponde alla lista di tutti i depositi dei veicoli;
- *parks* corrisponde alla lista delle aree di sosta;
- *vehs* corrisponde alla lista dei veicoli disponibili;
- *maxTrip* definisce il tempo massimo di viaggio dell'autista prima di sostare;
- *minIdle* definisce tempo minimo di sosta dell'autista prima di ripartire.

Analizzando il corpo principale dell'algoritmo, inizialmente verranno create tutte le rotte associate a ciascun veicolo, inserendo come primi nodi il deposito iniziale e quello finale (righe 2-4). Successivamente viene ordinata la lista di richieste rispetto al campo di prenotazione del prelievo desiderato, ovvero comparando il campo $StartTimeWindowBegin$ (riga 5). Verranno poi scandita la lista di tutte le richieste presenti in *reqs*, finchè non ce ne saranno più all'interno (righe 6-37). Andando ad esplorare all'interno di questo ciclo, inizialmente vengono inzializzate alla riga 7 tutte le variabili che terranno traccia di dove verrà aggiunta la richiesta, memorizzando in *bestCost* il costo totale migliore della soluzione, in $bestRoute_i$ l'indice della rotta del miglior inserimento della richiesta, in $bestPck_i$ l'indice del miglior posto dove poter inserire la richiesta di pickup nella rotta con indice $bestRoute_i$ e in $bestDlv_i$ l'indice del miglior posto dove poter inserire la richiesta di delivery nella rotta con indice $bestRoute_i$. Alla riga 8 si estrae la prima richiesta disponibile in *reqs*, memorizzando il relativo pickup in *pck* e il relativo delivery in *dlv*. In seguito si esplorano tutte le rotte disponibili in *routes* memorizzando una copia della rotta corrente in *route*, alle righe 9-27. Per ciascuna rotta corrente, se questa contiene solamente il deposito iniziale e quello finale e se il costo della rotta inserendo il pickup dopo il deposito iniziale e il delivery prima del deposito finale è minore del miglior costo, allora memorizzeremo il nuovo miglior costo della soluzione e gli indici che ci ricorderanno dove inserire le richieste (righe 10-15). Nel caso in cui la rotta abbia almeno una richiesta inserita (righe 16-26) cicliamo tutte le possibili posizioni dove poter inserire il pickup e il delivery partendo dall'indice dell'ultima area di sosta inserita, o dal deposito iniziale nel caso fossero assenti quest'ultime (righe 17-18). Nel caso in cui si trovassero le posizioni migliori del pickup (p_i) e del delivery (d_i) che ci fanno ottenere un nuovo costo della soluzione minore rispetto alla migliore, allora si memorizzeranno tutte le variabili per tener conto di dove inserire la richiesta per migliorare la soluzione (righe 19-25). Dopo aver ciclato tutte le rotte, avremo trovato la rotta e i posti migliori dove poter inserire

la richiesta selezionata. Si definiscono la rotta migliore bR dove inserire la richiesta e la subroute $subR$ che parte dall'indice dell'ultima area di sosta $lastPark_i$ (se esiste) oppure dall'indice del deposito iniziale $depIniz_i$ fino all'indice del deposito finale $depFin_i$ (riga 28) e quindi si inseriscono il pickup e il delivery nei posti migliori trovati nella rotta migliore (riga 29). Se con l'inserimento di questa nuova richiesta, il tempo totale del subtour $subR$ dovesse eccedere il tempo massimo di guida dell'autista $maxTrip$, allora dovremo rimuovere la richiesta precedente dalla rotta e inserire alla fine l'area di sosta che graverà minormente sulla soluzione finale, altrimenti si potrà rimuovere la richiesta dalla lista delle richieste da inserire (righe 30-36). Quando saranno inserite tutte le richieste, allora verranno rimossi i parcheggi che sono presenti alla fine delle rotte (riga 38). L'algoritmo viene terminato, verranno restituite tutte le rotte con tutte le richieste inserite (riga 39).

Capitolo 6

Analisi sperimentale

In questa sezione, sono stati condotti degli esperimenti numerici utilizzando un PC (Intel Core i7-8750H, 2.20 GHz, 16 GB di RAM) per convalidare l'efficacia del modello proposto e l'efficienza dei metodi di soluzione sviluppati. Il modello è implementato tramite l'utilizzo del framework .NET Core 6.0 con linguaggio di programmazione C# e l'uso del tool Visual Studio 2022. La scelta è ricaduta su questa implementazione in quanto il tool CPLEX riesce a risolvere problemi solo su piccole istanze tramite l'utilizzo della MIP, con questa metodologia di affronto del problema invece è stato possibile affrontare istanze di problemi su grande scala tramite l'implementazione della metauristica ALNS.

Il progetto risolutivo sviluppato è stato diviso in due sezioni: inizialmente nella parte di generazione dei dati, descritto nella Sezione 6.1 e nella parte di risoluzione della soluzione, come già descritto precedentemente nel capitolo 5. Per la parte di sperimentazione, sono stati eseguiti differenti test su piccole e grandi istanze del problema, come descritto nella Sezione 6.2.

6.1 Generatore di dati

Per avere una fase sperimentale del modello costruito in questa tesi, si aveva la necessità di implementare un generatore di dati stocastico che potesse creare dinamicamente degli input da poter utilizzare. Per la realizzazione di questo tool, l'idea è stata quella di caricare dei dati di base da un file di testo contenente tutte le variabili del problema e quindi di generare dinamicamente da essi delle istanze su cui applicare l'algoritmo sviluppato. Questo ci da modo di poter fare data augmentation sui dati di input, avendo la possibilità di creare istanze differenti partendo dalle stesse variabili.

In seguito vengono definiti tutti i dati di input che vengono utilizzati dal generatore di dati.

- $MinLat$: coordinata di latitudine minima dell'area interessata.
- $MinLng$: coordinata di longitudine minima dell'area interessata.
- $MaxLat$: coordinata di latitudine massima dell'area interessata.
- $MaxLng$: coordinata di longitudine massima dell'area interessata.
- $StartTime$: data e ora di inizio del problema.
- $EndTime$: data e ora di fine del problema.
- $MaxReq$: tempo massimo in minuti tra l'inizio e la fine di una richiesta.
- $MaxFromReq$: tempo massimo in minuti tra una richiesta e l'inizio della richiesta successiva.
- $MaxIntervReq$: tempo massimo in minuti tra la massima tolleranza di pickup e la minima tolleranza di delivery.
- $DepotsNum$: numero di depositi dei veicoli.
- $ParkingsNum$: numero delle aree di sosta.
- $CustomersNum$: numero delle richieste effettuate dai clienti.
- $VehicleNum$: numero dei veicoli.
- $Capacity$: massima capacità di persone raggiungibile da ciascun veicolo.
- $MaxQuantity$: massima quantità di clienti generati per ciascuna richiesta.
- $StartAvailability$: data e ora di inizio di disponibilità dei veicoli.
- $MaxTripDuration$: massimo tempo di viaggio dell'autista prima di sostare.
- $MinIdleParking$: minimo tempo di sosta di riposo dell'autista ad un'are di sosta.
- $MaxReqWaitTime$: tempo massimo di sosta del veicolo per far salire o scendere il/i cliente/i.
- $PenalizationRequestNotServed$: penalizzazione per ogni richiesta non servita nell'arco giornaliero.
- $PenalizationTimeOutOfTW$: penalizzazione per ogni secondo totale delle richieste servite al di fuori della finestra temporale.

- $PenalizationTimeInTW$: penalizzazione per ogni secondo totale delle richieste servite all'interno della finestra temporale ma non nel tempo prestabilito.
- $PenalizationTimeTripDuration$: penalizzazione per ogni secondo totale in cui ciascun autista guida oltre il suo massimo tempo prestabilito.
- $PenalizationTimeIdleParking$: penalizzazione per ogni secondo totale in cui ciascun autista non sosta il minimo di tempo necessario nell'area di sosta.
- $PenalizationDistanceVehicle$: penalizzazione per ogni metro percorso dal veicolo.
- $UseRandomRequestDestroy$: impostazione per stabilire l'utilizzo della operazione di random request destroy nella ALNS.
- $UseShawRequestDestroy$: impostazione per stabilire l'utilizzo della operazione di shaw request destroy nella ALNS.
- $UseWorstCostRequestDestroy$: impostazione per stabilire l'utilizzo della operazione di worst cost request destroy nella ALNS.
- $UseGreedyRequestRepair$: impostazione per stabilire l'utilizzo della operazione di greedy request repair nella ALNS.
- $UseRegret2RequestRepair$: impostazione per stabilire l'utilizzo della operazione di 2-regret request repair nella ALNS.
- $UseRegret3RequestRepair$: impostazione per stabilire l'utilizzo della operazione di 3-regret request repair nella ALNS.
- $UseRegret4RequestRepair$: impostazione per stabilire l'utilizzo della operazione di 4-regret request repair nella ALNS.
- $UseRandomParkingDestroy$: impostazione per stabilire l'utilizzo della operazione di random parking destroy nella ALNS.
- $UseWorstCostParkingDestroy$: impostazione per stabilire l'utilizzo della operazione di worst cost parking destroy nella ALNS.
- $UseBestParkingRepair$: impostazione per stabilire l'utilizzo della operazione di best parking repair nella ALNS.
- $UseGreedyParkingRepair$: impostazione per stabilire l'utilizzo della operazione di greedy parking repair nella ALNS.
- $UseGreedyParkingWithComparisonRepair$: impostazione per stabilire l'utilizzo della operazione di greedy parking with comparison repair nella ALNS.

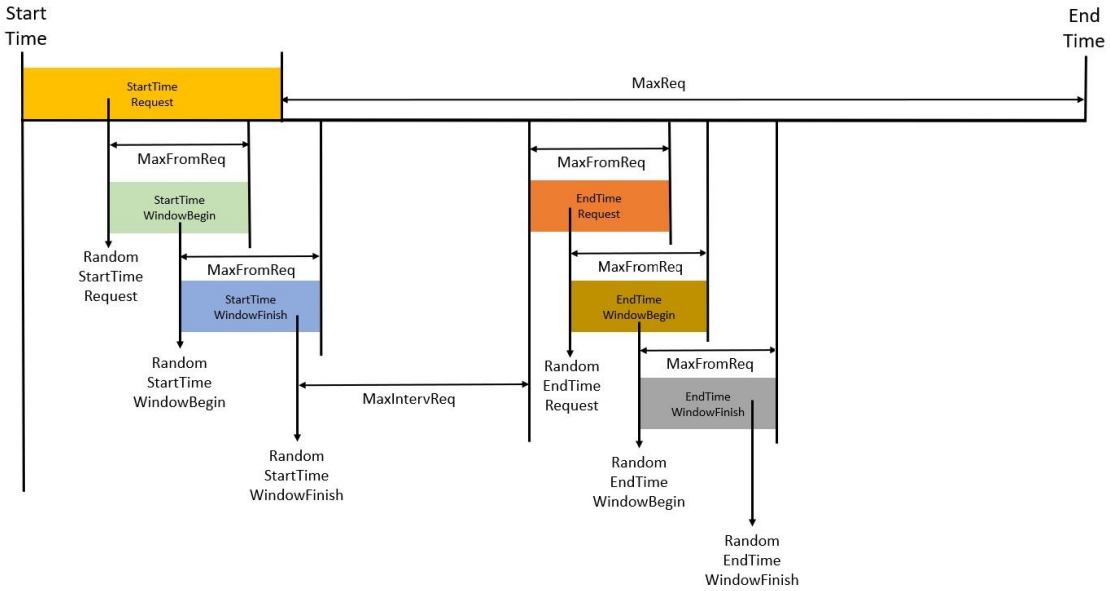

Figura 6.1: Grafico delle aree di definizione delle finestre temporali.

Andando più nello specifico, sono state adottate le variabili $MinLat$, $MinLng$, $MaxLat$ e $MaxLng$ per poter delimitare un'area nella quale vengano generati tutti i nodi del problema (depositi, aree di sosta e richieste), dove nel caso delle nostre prove è stata selezionata l'area urbana di una città metropolitana. $StartTime$ (St) ed $EndTime$ (Et) sono state definite per avere un intervallo di tempo minimo e massimo del problema. $MaxReq$ (Mr), $MaxFromReq$ (Mfr) e $MaxIntervalReq$ (Mir) sono tre variabili che sono utilizzate per impostare le fineste temporali di ogni richiesta, ovvero le variabili $StartTimeRequested$ (Str), $StartTimeWindowBegin$ (Stb), $StartTimeWindowFinish$ (Stf), $EndTimeRequested$ (Etr), $EndTimeWindowBegin$ (Etb) e $EndTimeWindowFinish$ (Etf) come definite in figura 6.1.

I tempi delle finestre temporali sono decretati con le seguenti formule, dove definiamo $time\_rand(a, b)$ come una funzione che restituisce un tempo casuale tra nell'intervallo $[a, b]$:

$$Str = time\_rand(St, Et - Mr) \tag{6.1}$$

$$Stb = time\_rand(Str, Str + Mr) \tag{6.2}$$

$$Stf = time\_rand(Stb, Stb + Mfr) \tag{6.3}$$

$$Etr = time\_rand(Stf + Mir, Stf + Mir + Mfr) \tag{6.4}$$

$$Etb = time\_rand(Etr, Etr + Mfr) \tag{6.5}$$

$$Etf = time\_rand(Etb, Etb + Mfr) \tag{6.6}$$

Successivamente si creano, nell'area selezionata precedentemente, tutti i nodi inerenti al problema, ovvero i depositi, le aree di sosta, le richieste di pickup e le richieste di delivery, posizionandoli casualmente. Alle richieste verranno inoltre assegnate delle finestre temporali causali, utilizzando la metodologia descritta precedentemente. Ciascuna richiesta avrà una quantità di persone che deve essere prelevata e depositata, che viene generata casualmente attraverso l'utilizzo di questa formula:

$$Quantity = \lceil \frac{-\ln p \cdot (MaxQuantity - 1)}{6} \rceil + 1, \quad p \in [0,1) \tag{6.7}$$

dove $MaxQuantity$ è la massima quantità di persone che si possono prelevare, e p un numero decimale casuale nell'intervallo di 0 compreso e 1 non compreso.

Viene gestita inoltre la possibilità di configurare tutte le penalizzazioni dei vincoli violabili ($soft$) per poter decidere, prima di avviare il resolver, quali vincoli si preferisce pesare maggiormente rispetto ad altri, così da poter confrontare tutti i risultati alla fine delle prove. Ad esempio se in una prova si volesse evitare che l'autista faccia viaggi entro il suo massimo tempo prima di riposarsi, basterà aumentare notevolmente il valore di peso della variabile $PenalizationTimeTripDuration$.

Un altro fattore che ci permette di studiare al meglio quali algoritmi di distruzione e riparazione possono essere più utili rispetto ad altri nel trovare una soluzione meno costosa, sarebbe quello di configurare le variabili che ci permettono di non utilizzare certe operazioni, ad esempio se non volessimo utilizzare la randomicità nella distruzione delle rotte, potremo settare le variabili $UseRandomRequestDestroy$ e $UseRandomParkingDestroy$ a $false$ per non essere utilizzate dall'algoritmo di ALNS.

6.2 Prove e analisi sperimentali

In primo luogo, viene qui nel seguito illustrato come sono state effettuate le prove sperimentali. Utilizzando il generatore di dati descritto nella sezione 6.1, sono state create delle istanze su cui effettuare le prove; in particolare si è scelto di creare 5 istanze per ciascun tipo, come descritto nella tabella 6.1.

Si può notare che all'aumentare del numero delle richieste, i valori assegnati agli altri parametri crescono più lentamente.

Tutte le istanze inoltre condividono lo stesso valore dei seguenti dati aggiuntivi:

Istanze	[0,5)	[5, 10)	[10,15)	[15,20)	[20,25)
Numero depositi	1	1	3	3	3
Numero aree sosta	2	2	4	4	4
Numero richieste	10	20	40	60	100
Numero veicoli	2	2	3	5	7
Capacità veicoli	4	5	8	10	10
Massima quantità persone per richiesta	4	5	8	8	5

Tabella 6.1: Parametri assegnati a ciascun range di istanze sui quali sono state effettuate le analisi.

- Definizione della stessa area urbana.
- Tempo di inizio ore 8:00.
- Tempo di fine ore 20:00.
- Disponibilità dei veicoli dalle ore 8:00.
- Massimo tempo di guida di ogni autista: 3 h.
- Minimo tempo di riposo di ogni autista in un'area di sosta: 15 min.
- Peso di penalizzazione per ogni cliente non servito: 100.
- Peso di penalizzazione per richiesta servita al di fuori della sua finestra temporale: 100.
- Peso di penalizzazione per richiesta servita nella finestra temporale ma non servendo il cliente all'orario prestabilito: 2.
- Peso di penalizzazione per ogni volta che un autista è andato oltre al massimo tempo di guida di un'autista: 100.
- Peso di penalizzazione per ogni volta che un autista si è fermato meno tempo rispetto a quello minimo di pausa: 100.
- Peso di penalizzazione per ogni chilometro percorso da ciascun mezzo: 1.
- Utilizzo di tutte le operazioni di inserimento e rimozione nella ALNS.

Il risolutore dopo aver preso in input le istanze generate, ha creato dei documenti Excel contenenti tutti i risultati, così da analizzare i diversi tipi di dati e capire quali effetti abbiano su ogni tipo di problema studiato.

In questa fase l'obiettivo principale è stato di capire quanto sia l'effettivo miglioramento della soluzione partendo dalla iniziale. Per ogni istanza sono stati raccolti i seguenti dati: il numero di esecuzione dell'istanza definita, il numero dell'istanza, il costo della soluzione iniziale, il costo della soluzione finale, il tempo di esecuzione dell'algoritmo e il miglioramento percentuale tra la soluzione iniziale e quella finale. Analizziamo i dati ottenuti con le istanze da 20 richieste, considerando la tabella 6.2.

Analizzando tale tabella, si nota che eseguendo più volte la stessa istanza, il miglioramento della soluzione tende a rimanere in un range di valori molto ristretto (tranne in qualche caso dove si ha un comportamento diverso dall'andamento medio), come per il tempo di esecuzione. Questi comportamenti sono dovuti alla casualità nella scelta delle operazioni di distruzione e riparazione, che producono sempre percorsi differenti e danno un tasso di stocasticità alla metauristica. Per l'istanza 2, ad esempio, si nota che l'algoritmo della soluzione iniziale agisce già in maniera da restituire una soluzione molto vicina a quella migliore, mentre negli altri casi la ALNS migliora notevolemente durante l'esecuzione.

In seguito abbiamo operato la stessa idea anche su un numero differenti di istanze, sia di piccola che grande scala. Oltre alle richieste mostrate precedentemente (20), abbiamo eseguito l'euristica su differenti istanze da 10, 40, 60 e 100 richieste. Per ciascuna sono stati calcolati il costo iniziale, il costo medio della migliore soluzione finale, la media e deviazione standard del miglioramento percentuale rispetto alla soluzione iniziale e infine il tempo medio, come riportato nella tabella 6.3.

Si può notare che in molte soluzioni ci si è avvicinati allo stesso risultato finale in tutte le esecuzioni della stessa istanza, dove naturalmente la deviazione del miglioramento percentuale è molto basso e bilanciato con la media corrispondente. In alcuni casi c'è stato un alto valore di dispersione causata da una esecuzione che ha dato un risultato completamente sbilanciato, in quanto lo sviluppo dell'eustica porta a risultati dinamici in cui possono essere presi dei percorsi differenti che hanno portato ad un miglioramento repentino, e allo stesso tempo ad un incremento del tempo di esecuzione. Questo effetto si può notare, ad esempio, nelle istanze da 10 richieste, dove dei risultati che si sono scostati sensibilmente dalla media, hanno portato ad una maggiore dispersione del costo della soluzione finale media. Questo effetto si perde all'aumentare del numero di richieste, in quanto si stabilizza maggiormente in media sul valore finale.

Un altro dato rilevante che si osserva è che all'aumentare del numero delle richieste, incrementa il tempo di esecuzione, in quanto cresce la difficoltà del problema. Come definito all'inizio di questa tesi, l'algoritmo sviluppato sarà eseguito il giorno precedente rispetto alla realizzazione dei viaggi, operazione che

Esec.	**Istanza**	**Costo iniziale**	**Costo finale**	**Tempo**	**Miglioram. percent.**
1	1	95642	67153	34	29,79%
2	1	95642	73808	18	22,83%
3	1	95642	66789	43	30,17%
4	1	95642	67153	39	29,79%
5	1	95642	65737	28	31,27%
1	2	136994	123765	24	9,66%
2	2	136994	136743	30	0,18%
3	2	136994	136589	14	0,3%
4	2	136994	124666	14	9%
5	2	136994	136834	31	0,12%
1	3	99515	62893	69	36,8%
2	3	99515	62893	56	36,8%
3	3	99515	63603	60	36,09%
4	3	99515	66972	26	32,7%
5	3	99515	48443	189	51,32%
1	4	356267	71408	26	79,96%
2	4	356267	71335	33	79,98%
3	4	356267	84188	52	76,37%
4	4	356267	71335	30	79,98%
5	4	356267	74718	18	79,03%
1	5	447982	61689	19	86,23%
2	5	447982	64981	21	85,49%
3	5	447982	60005	27	86,61%
4	5	447982	60818	26	86,42%
5	5	447982	62047	24	86,15%

Tabella 6.2: Risultati medi dell'ALNS per le istanze da 20 richieste (5 ripetizioni per istanza).

sarà effettuata dopo aver ricevuto le richieste in input dai clienti tramite un'app mobile o un portale web. Quindi si possono tollerare anche esecuzioni con tempi più lunghi in quanto la pianificazione viene eseguita fuori linea con sufficiente anticipo per poter ottenere i risultati per la mattinata del giorno successivo. Anche considerando i tempi maggiori (ovvero quelli relativi alla pianificazione con 100 richieste), questi risultano limitati a circa 20 minuti, che quindi possono essere considerati certamente accettabili.

Aumentando il numero dei veicoli, il problema viene risolto con più facilità. Si

Num. rich.	**Istanza**	**Costo iniziale**	**Costo finale medio**	**Miglior. perc. medio**	**Miglior. perc. dev. std.**	**Tempo medio**
10	1	140106	135723	3,13	0,64	4,2
	2	2394489	540071	77,44	10,23	22,4
	3	69259	56558	18,34	3,98	6,4
	4	49981	42339	15,29	14,44	5,8
	5	64065	56103	12,43	11,34	4,0
20	6	95642	68128	28,77	3,38	32,4
	7	136995	134365	3,82	5,00	22,6
	8	99515	60961	38,74	7,23	80,0
	9	356267	74597	79,06	1,56	31,8
	10	447982	61908	86,18	0,42	23,4
40	11	573970	180984	68,47	8,99	221,4
	12	540092	225205	58,30	19,80	184,8
	13	243271	125838	48,27	9,88	107,0
	14	261619	108649	58,47	3,10	104,2
	15	333180	143547	56,92	9,73	137,6
60	16	143070	140681	1,67	1,54	190,4
	17	214669	151450	29,45	8,74	325,8
	18	175319	123770	29,40	1,57	201,6
	19	206926	138634	33,00	1,12	252,6
	20	180393	151683	15,91	9,29	161,6
100	21	217457	182137	13,00	7,94	964,0
	22	234761	187489	20,13	3,39	755,6
	23	243607	201048	17,47	7,31	1140,8
	24	241561	225529	5,63	5,63	671,8
	25	219747	187058	14,88	3,34	1293,0

Tabella 6.3: Risultati medi dell'ALNS per 10, 20, 40, 60 e 100 richieste (5 istanze per ogni numero di richieste e 5 ripetizioni per istanza).

deve osservare che il numero di veicoli per le varie istanze è stato mantenuto basso; ciò è stato fatto per valutare le prestazioni della pianificazione quando è impegnato un numero limitato di risorse, considerando anche il fatto che tale numero non viene penalizzato nel costo finale. Quindi si impostato questo valore tenendo conto dell numero delle richieste prenotate.

L'analisi ha evidenziato come per alcune soluzioni in media ALNS ha ottenuto un miglioramento relativamente basso. La ragione di ciò è dovuta alla buona

qualità della soluzione iniziale determinata dall'algoritmo utilizzato come primo step della ALNS, soluzione che probabilmente si trova vicino ad un minimo locale da cui ALNS non è stata in grado di allontanarsi. In altri casi invece la soluzione iniziale non ci porta ad avere subito una buona soluzione e si osserva come già nelle prime iterazioni della ALNS la soluzione abbia un notevole miglioramento. tali miglioramenti decrescono col crescere delle iterazioni fino a stabilizzarsi dopo un tempo di esecuzione sufficientemente lungo. Come si nota dalla figura 6.2, si ha questo effetto descritto precedentemente in cui la soluzione nelle prime 10 iterazioni circa migliora quella iniziale quasi del 30%, per poi assestarsi per circa 200 iterazioni, diminuire linearmente nelle successive 200 e scendere del 10% in una iterazione intorno alla 400, per poi stabilizzarsi trovando l'ultima soluzione migliorativa intorno all'iterazione 600.

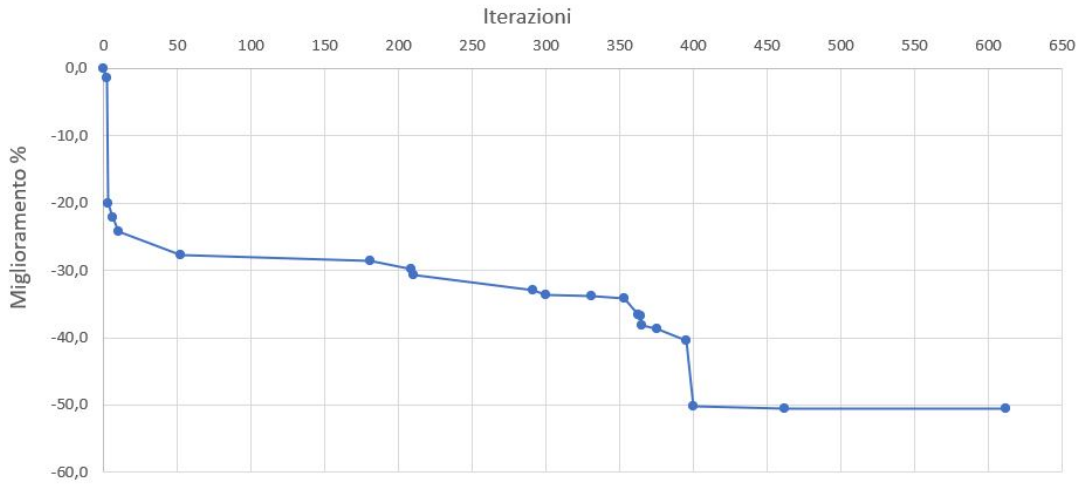

Figura 6.2: Grafico del miglioramento percentuale dalla soluzione iniziale determinata dalla ALNS per un'istanza.

Una possibilità per poter migliorare la qualità delle soluzioni determinate dall'algoritmo è quella di effettuare un *tuning* dei valori dei parametri. Inoltre è possibile rimuovere quelle operazioni di distruzione e inserimento il cui utilizzo in genere non porta a miglioramenti del costo, effettuando un'analisi sperimentale preliminare che consideri come variano i pesi in base ai quali vengono scelte le euristiche.

Le operazioni di distruzione e riparazione vengono inizializzate tutte con lo stesso peso e quindi con la stessa probabilità di essere scelte dall'euristica ALNS. In seguito vengono aggiornate ad ogni iterazione, a seconda se è stata trovata una soluzione migliore, locale oppure se è stata accettata. Come si vede dalla figura 6.3, le operazioni partono tutte dallo stesso peso, ma se in quelle delle richieste si hanno delle variazioni iniziali che poi tendono ad assestarsi a quel

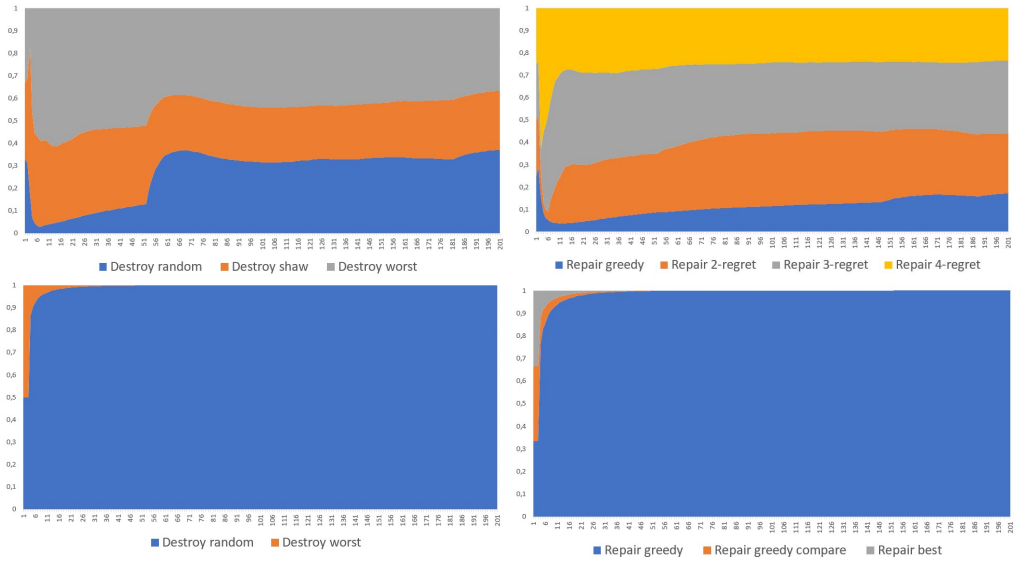

Figura 6.3: Quattro grafici che mostrano l'aggiornamento dei pesi delle operazioni di distruzione delle richieste (in alto a sinistra), delle operazioni di riparazione delle richieste (in alto a destra), delle operazioni di distruzione delle aree di sosta (in basso a sinistra) e delle operazioni di riparazione delle aree di sosta (in basso a destra). I pesi partono con lo stesso valore iniziale e variano durante l'esecuzione della ALNS.

valore, in quelle delle aree di sosta un'operazione prende il sopravvento sulle altre e viene sempre richiamata avendo una probabilità vicina al 100% di essere scelta. Nel caso delle operazioni di riparazione, inizialmente viene richiamato più spesso l'algoritmo di distruzione dell'elemento peggiore (worst destroy) e meno la distruzione casuale (random destroy), per poi, verso l'iterazione 60, aumentare quest'ultima ridiventando competitiva con le altre due. Nel caso delle operazioni di riparazione delle richieste, inizialmente l'operazione di riparazione 4-regret incrementa la probabilità di essere utilizzata per poi diminuire e mediare con le altre; si nota inoltre che l'operazione di riparazione greedy avrà meno possibilità di essere utilizzata. Nelle operazioni di distruzione e riparazione delle aree di sosta, l'operazione di distruzione casuale e l'operazione di riparazione greedy hanno un incremento della probabilità di essere scelte vicina al 100% dopo 30 iterazioni, dimostrando di essere le operazioni che operano nel miglior modo in questa specifica soluzione.

Come ultima parte di questa sezione vengono svolte delle comparazioni sulle soluzioni ottenute dalle istanze piccole tramite l'utilizzo del tool CPLEX, usando la programmazione MIP. Viene quindi utilizzato il modello descritto nel capitolo 4

per controntare i risultati dati dalla ALNS definita nel capitolo 5.

In tabella 6.4 sono mostrati i risultati medi ottenuti dalla ALNS per le istanze con 20 richieste confrontati con i risultati ottenuti dal solver CPLEX per il modello MIP presentato nel capitolo 3. Il tempo di computazione massimo dato al solver è stato di 1800 secondi.

	ALNS		**Cplex**		
Istanza	**Obiettivo**	**Tempo**	**Obiettivo**	**Tempo**	**Deviazione %**
1	68128,46	32,4	133607,07	1800,2	-49,01%
2	134365,7	22,6	135105,08	1800,2	-0,55%
3	60961,11	80,0	117173,77	1800,1	-47,97%
4	74597,4	31,8	115766,85	1800,4	-35,56%
5	61908,49	23,4	124173,7	1800,5	-50,14%

Tabella 6.4: Confronto tra i risultati medi di ALNS e CPLEX per le istanza con 20 richieste).

Cplex in 30minuti non è mai stato in grado di determinare la soluzione ottima e, come si può osservare dalla tabella, i risultati ottenuti da ALNS sono risultati sempre migliori con una deviazione percentuale media pari a 36,65% in tempi di calcolo medi pari a 38 secondi. Questi test, anche se limitati, mostrano come la meta-euristica sia in grado di fornire soluzioni di buona qualità in tempi molto contenuti e che per tale motivo possa essere utilizzata per risolvere istanze reali del problema considerato.

Capitolo 7

Conclusioni

In questo elaborato di tesi è stato modellato un problema di VRPPDTW Multi-D&T con pause di sosta opzionali, tramite lo sviluppo di una metauristica basata su ALNS. Questo si è dimostrato essere un buon metodo di risoluzione delle problematiche nel contesto reale considerato, caratterizzato dal considerare le normative nell'ambito del trasporto pubblico che richiedono che gli autisti facciano delle pause dopo un tempo prolungato di guida.

Nel capitolo 5 sono stati riportati tutti i metodi utilizzati al fine di ottenere una completa metauristica che risolvesse un problema reale e che potesse essere utilizzata per un progetto realistico in corso. L'idea di utilizzo della ALNS è motivato dai suoi eccellenti risultati su problemi di routing complessi.

La differenza principale di questo studio rispetto ad altri esistenti consiste nella gestione delle pause che effettua ciascun autista tramite l'utilizzo di predefinite zone di sosta. Inoltre viene tenuto conto che debbano sostenere dei tempi massimi di guida e dei tempi minimi di sosta prima di riprendere la guida del veicolo. Inoltre in questo elaborato vengono definiti dei nuovi metodi di inserimento e rimozione delle aree di sosta nelle rotte, riportando ciò che è stato menzionato nelle sezioni 5.4 e 5.5.

Un altro fattore importante di questo studio è la gestione delle finestre temporali: ogni cliente che vuole prenotare la propria richiesta, avrà a disposizione un orario di scelta e due orari di tolleranza minimo e massimo in cui vorrà essere prelevato e depositato, come è stato illustrato nella sezione 5.6. Questi orari sono importanti in quanto il costo totale dell'algoritmo sarà minore se si riuscirà a rispettare l'orario di prenotazione dell'utente. Il peso totale aumenterà leggermente se non si rispetterà l'orario predefinito, ma si rimarrà nella soglia minima e massima di tolleranza. Invece se si andrà oltre a questi limiti fissati, il peso incrementerà maggiormente sul costo totale. Questa parte viene definita nella sezione 5.8.

Per valutare l'algoritmo, è stato utilizzato un set di benchmark realizzato da un generatore di dati casuali derivanti dalla configurazione di un set di parametri

definiti, come definito nella sezione 6.1. Grazie a questi set siamo riusciti a fare delle sperimentazioni ed a confrontare le varie istanze sia sullo stesso numero che su un numero differente di richieste.

Poniamo attenzione anche a possibili futuri sviluppi, per così poter migliorare o espandere la problematica ad altri settori.

Inizialmente si potrebbe perfezionare l'algoritmo a livello di complessità di codice: attualmente il valore di questo aspetto è molto alto, infatti all'aumentare del numero di nodi (richieste, depositi e aree di sosta), aumenta anche il tempo di esecuzione. Attualmente l'aggiornamento del costo della soluzione finale parte dal deposito iniziale ed arriva al deposito finale. Si potrebbe aggiornare il costo delle rotte a partire dal nodo che viene rimosso oppure aggiunto alla rotta designata, oppure aggiungere delle nuove operazioni di inserimento che operano sullo stesso subtour dell'operazione di rimozione per poter così ridurre l'area di aggiornamento della soluzione.

Un'altra possibilità sarebbe quella di comparare il modello costruito con altri competitor (come OR-Tools di Google) per poter verificare se l'algoritmo lavora bene con grandi istanze, oltre a quelle piccole che abbiamo già verificato con il modello costruito per CPLEX.

Un possibile nuovo sviluppo del problema, guardando all'ambito della sostenibilità, potrebbe essere quello di sostituire i veicoli a carburante con quelli elettrici, avendo una possibilità di impatto a livello ambientale ed ecologico.

Il problema potrebbe essere ampliato anche a centri urbani più grandi, dove la mole di richieste giornaliere è sicuramente maggiore rispetto alle città metropolitane, come vengono considerate in questo problema.

Bibliografia

[1] Osservatori.net digital innovation. *Logistica e Covid-19: cala il mercato (-9,3%) ma non si ferma l'innovazione*. Nov. 2020. URL: https://www.osservatori.net/it/ricerche/comunicati-stampa/logistica-2020-mercato-fatturato-innovazione (cit. a p. 1).

[2] G.B. Dantzig e J.H. Ramser. «The Truck Dispatching Problem». In: 6.1 (ott. 1959), pp. 1–140. URL: https://doi.org/10.1287/mnsc.6.1.80 (cit. a p. 4).

[3] G. Clarke e J.W. Wright. «Scheduling of vehicles from a central depot to a number of delivery points». In: 12.4 (ago. 1964), pp. 519–643. URL: https://doi.org/10.1287/opre.12.4.568 (cit. a p. 4).

[4] Wikipedia. *Vehicle Routing Problem*. URL: https://en.wikipedia.org/wiki/Vehicle_routing_problem (cit. a p. 4).

[5] P. Toth e D. Vigo. «The Vehicle Routing Problem». In: (gen. 2002). URL: https://doi.org/10.1137/1.9780898718515 (cit. a p. 5).

[6] L. Zhen, C. Ma, K. Wang, L. Xiao e W. Zhang. «Multi-depot multi-trip vehicle routing problem with time windows and release dates». In: 135 (mar. 2020). URL: https://doi.org/10.1016/j.tre.2020.101866 (cit. alle pp. 5, 19).

[7] M.L. Fisher. «Optimal Solution of Vehicle Routing Problems Using Minimum K-Trees». In: 42.4 (ago. 1994), pp. 574–788. URL: https://doi.org/10.1287/opre.42.4.626 (cit. a p. 10).

[8] D.L. Miller. «A Matching Based Exact Algorithm for Capacitated Vehicle Routing Problems». In: 7.1 (feb. 1995), pp. 1–116. URL: https://doi.org/10.1287/ijoc.7.1.1 (cit. a p. 10).

[9] C.E. Miller, A.W. Tucker e R.A. Zemlin. «Integer Programming Formulation of Traveling Salesman Problems». In: 7.4 (ott. 1960), pp. 326–329. URL: https://doi.org/10.1145/321043.321046 (cit. a p. 10).

[10] N. Christofides, N. Mingozzi e A. Toth. «Combinatorial Optimization». In: (1979), pp. 315–338 (cit. a p. 10).

[11] M. Desrochers e G. Laporte. «Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints». In: 10.1 (feb. 1991), pp. 27–36. URL: https://doi.org/10.1016/0167-6377(91)90083-2 (cit. a p. 10).

[12] Y. Dumas, J. Desrosiers e F. Soumis. «The pickup and delivery problem with time windows». In: (set. 1991), pp. 7–22. URL: https://doi.org/10.1016/0377-2217(91)90319-Q (cit. a p. 17).

[13] G. Desaulniers, J. Desrosiers, I. loachim, M.M. Solomon, F. Soumis e D. Villeneuve. «A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems». In: (gen. 1998), pp. 57–93. URL: https://doi.org/10.1007/978-1-4615-5755-5_3 (cit. a p. 17).

[14] S. Salhi, A. Imran e N.A. Wassan. «The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation». In: 52.B (dic. 2014), pp. 315–325. URL: https://doi.org/10.1016/j.cor.2013.05.011 (cit. a p. 18).

[15] G. Desaulniers, J. Lavigne e F. Soumis. «Multi-depot vehicle scheduling problems with time windows and waiting costs». In: 111.3 (dic. 1998), pp. 479–494. URL: https://doi.org/10.1016/S0377-2217(97)00363-9 (cit. a p. 18).

[16] R.G. Dondo e J. Cerdà. «A hybrid local improvement algorithm for large-scale multi-depot vehicle routing problems with time windows». In: 33.2 (feb. 2009), pp. 513–530. URL: https://doi.org/10.1016/j.compchemeng.2008.10.003 (cit. a p. 18).

[17] A. Bettinelli, A. Ceselli e G. Righini. «A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows». In: 19.5 (ago. 2011), pp. 723–740. URL: https://doi.org/10.1016/j.trc.2010.07.008 (cit. a p. 18).

[18] H. Bae e I. Moon. «Multi-depot vehicle routing problem with time windows considering delivery and installation vehicles». In: 40.13-14 (lug. 2016), pp. 6536–6549. URL: https://doi.org/10.1016/j.apm.2016.01.059 (cit. a p. 18).

[19] Naveed Wassan, Niaz Wassan, G. Nagy e S. Salhi. «The Multiple Trip Vehicle Routing Problem with Backhauls: Formulation and a Two-Level Variable Neighbourhood Search». In: 78 (feb. 2017), pp. 454–467. URL: https://doi.org/10.1016/j.cor.2015.12.017 (cit. a p. 18).

[20] P.K. Nguyen, T.G. Crainic e M. Toulouse. «A tabu search for Time-dependent Multi-zone Multi-trip Vehicle Routing Problem with Time Windows». In: 231.1 (nov. 2013), pp. 43–56. URL: https://doi.org/10.1016/j.ejor.2013.05.026 (cit. a p. 19).

[21] S. Yang, J.C. Chu, F.Y. Hsiao e H.J. Huang. «A planning model and solution algorithm for multi-trip split-delivery vehicle routing and scheduling problems with time windows». In: 87 (set. 2015), pp. 383–393. URL: https://doi.org/10.1016/j.cie.2015.05.034 (cit. a p. 19).

[22] J. Tang, Y. Yu e J. Li. «An exact algorithm for the multi-trip vehicle routing and scheduling problem of pickup and delivery of customers to the airport». In: 249.2 (mar. 2016), pp. 551–559. URL: https://doi.org/10.1016/j.ejor.2015.08.040 (cit. a p. 19).

[23] S. Ropke e D. Pisinger. «A general heuristic for vehicle routing problems». In: 34.8 (ago. 2007). URL: https://doi.org/10.1016/j.cor.2005.09.012 (cit. alle pp. 25, 34).

[24] P. Shaw. «Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems». In: 1520 (giu. 1999), pp. 417–431. URL: https://doi.org/10.1007/3.540-49481-2_30 (cit. alle pp. 25, 26, 29, 33).

[25] S. Ropke e D. Pisinger. «Large Neighborhood Search». In: 272 (set. 2018), pp. 99–127. URL: https://doi.org/10.1007/978-3-319-91086-4_4 (cit. a p. 26).

[26] R. K. Ahuja, O. Ergun, J. B. Orlin e A. P. Punnen. «A survey of very large-scale neighborhood search techniques». In: 123.1-3 (nov. 2002), pp. 75–102. URL: https://doi.org/10.1016/S0166-218X(01)00338-9 (cit. a p. 26).

[27] S. Ropke e D. Pisinger. «Large Neighborhood Search». In: 40.4 (2006), pp. 455–472. URL: http://dx.doi.org/10.1287/trsc.1050.0135 (cit. alle pp. 29, 30).

[28] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt e G. Dueck. «Record Breaking Optimization Results Using the Ruin and Recreate Principle». In: 159.2 (apr. 2000), pp. 139–171. URL: https://doi.org/10.1006/jcph.1999.6413 (cit. a p. 29).

[29] R. Bent e P. V. Hentenryck. «A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows». In: 38.4 (nov. 2004), pp. 395–534. URL: http://dx.doi.org/10.1287/trsc.1030.0049 (cit. a p. 34).

[30] S. Kirkpatrick, C. D. Gelatt e M. P. Vecchi. «Optimization by simulated annealing». In: 220.4598 (giu. 1983), pp. 671–680. URL: https://doi.org/10.1126/science.220.4598.671 (cit. a p. 36).

[31] J. Y. Potvin e J. M. Rousseau. «A parallel route building algorithm for the vehicle routing and scheduling problem with time windows». In: 66.3 (mag. 1993), pp. 331–340. URL: https://doi.org/10.1016/0377-2217(93)90221-8 (cit. a p. 58).