ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE
Corso di Laurea in Scienze e Tecnologie Informatiche

Metodi Euristici per il Vehicle Routing Problem

Relazione finale in

RICERCA OPERATIVA

Relatore:
Prof. Aristide Mingozzi

Presentata da:
Andrea Battaglia

Sessione 2

Anno Accademico 2012 · 2013

Indice

1. Introduzione

Il problema della consegna di prodotti da un deposito/impianto ai clienti mediante una flotta di automezzi è un problema centrale nella gestione di una catena di produzione e distribuzione (*supply chain*). Questo problema, noto in letteratura come *Vehicle Routing Problem* (VRP), nella sua versione più semplice consiste nel disegnare per ogni veicolo disponibile presso un dato deposito aziendale un viaggio (*route*) di consegna dei prodotti ai clienti, che tali prodotti richiedono, in modo tale che (*i*) la somma delle quantità richieste dai clienti assegnati ad ogni veicolo non superi la capacità del veicolo, *(ii)* ogni cliente sia servito una ed una sola volta, *(iii)* sia minima la somma dei costi dei viaggi effettuati dai veicoli.

Il VRP è un problema trasversale ad una molteplicità di settori merceologici dove la distribuzione dei prodotti e/o servizi avviene mediante veicoli su gomma, quali ad esempio: distribuzione di generi alimentari, distribuzione di prodotti petroliferi, raccolta e distribuzione della posta, organizzazione del servizio scuolabus, pianificazione della manutenzione di impianti, raccolta rifiuti, etc.

Esistono diverse varianti di VRP. Il caso generale, noto come *Capacitated Vehicle Routing Problem* (CVRP), considera una flotta di veicoli dotati della stessa capacità di carico e impone che ciascuno di essi possa effettuare al più un percorso. Tuttavia, nella pratica, i veicoli possono eseguire diversi viaggi e, in molti contesti, una compagnia di distribuzione preferisce affittare veicoli di terzi per servire i clienti, incorrendo in un costo significativo per ogni veicolo utilizzato. Pertanto, generalmente, la preoccupazione primaria della compagnia è minimizzare il costo dei veicoli impiegati.

In questa tesi viene considerato il *Multi-Trip VRP* (MTVRP), un'estensione del CVRP in cui ogni veicolo può eseguire un sottoinsieme di percorsi, chiamato *vehicle schedule* (schedula del veicolo), soggetto a vincoli di durata massima. Nonostante la sua importanza pratica, il MTVRP ha ricevuto poca attenzione in letteratura: sono stati proposti diversi metodi euristici e un solo algoritmo esatto di risoluzione, presentato da Mingozzi et al. (2013).

In questa tesi viene presentato un metodo euristico in grado di risolvere istanze di MTVRP in presenza di vincoli reali, quali flotta di veicoli non omogenea e *time windows*. L'euristico si basa sul modello di Prins (2002), attualmente considerato uno dei più efficienti euristici per il MTVRP. Sono presentati inoltre due approcci di *local search* per migliorare la soluzione finale. I risultati computazionali evidenziano l'efficienza di tali approcci.

La tesi si articola come segue. Nel capitolo 2 viene descritto il MTVRP ed è illustrata una procedura che permette di minimizzare il numero di veicoli utilizzati risultante dall'esecuzione di un euristico sequenziale. Nel capitolo 3 è fornita una breve descrizione dell'algoritmo esatto di Mingozzi, Roberti e Toth e sono presentate le formulazioni matematiche su cui si basa tale metodo. Nel capitolo 4 vengono descritti i principali approcci euristici per il MTVRP; vengono trattati con maggior attenzione i metodi di Taillard et al. (1996) e di Prins (2002). Nel capitolo 5 viene proposto un metodo euristico per il MTVRP basato sul modello di Prins e i risultati conseguiti su un insieme di problemi test.

2. Multi-Trip VRP

Il problema che consiste nel calcolare un insieme di percorsi a costo totale minimo, per consegnare quantità fissate di beni ai clienti, con una flotta di veicoli identici, è noto con il nome di Vehicle Routing Problem (VRP). Tale problema è stato proposto da Dantzig e Ramser (1959); più in generale, VRP è il nome con cui in letteratura ci si riferisce ad un'intera classe di problemi dalle numerose implicazioni pratiche diffuse in tutti i settori concernenti il trasporto di merci, come ad esempio:

- raccolta di posta nelle cassette postali,
- servizio scuolabus,
- visite mediche a domicilio,
- visite di manutenzione preventiva,
- raccolta rifiuti.

La varietà di casi a cui il VRP si riferisce fa si che tali problemi siano difficilmente risolubili all'ottimo. Il caso generale si basa sull'organizzazione dei percorsi in presenza di consegne multiple ed un numero fissato di veicoli. Ciascuno di questi veicoli ha una capacità di trasporto limitata e può servire diversi clienti, ognuno dei quali richiede una data quantità di prodotto. I metodi di risoluzione di un VRP assegnano un insieme di clienti ad ogni veicolo ed elaborano, per ciascun veicolo, uno o più percorsi. Il problema richiede di minimizzare la somma dei costi associati a ciascun percorso. Tale costo è solitamente inteso in termini di distanza percorsa o di tempo impiegato.

Generalmente, il VRP considera:

- $n + 1$ nodi, con un deposito al nodo 1 ed un insieme $S_C = \{2, 3, ..., n + 1\}$ di n clienti;
- Una matrice delle distanze $D((n + 1) \times (n + 1))$, dove d_{ij} indica la distanza tra una coppia (i,j) di nodi;

- Ogni cliente *i* richiede una quantità q_i di prodotto.

Il deposito contiene una flotta omogenea $S_V = \{1, 2, ..., m\}$ di veicoli identici di capacità Q. Questa flotta è sufficiente, ovvero la somma delle richieste dei clienti non supera mQ. Un percorso è definito come una sequenza di visite ai clienti, effettuata da un veicolo, partendo dal deposito e tornando al deposito una volta visitati tutti i clienti. La quantità totale di prodotto consegnata in un percorso non può superare la capacità di trasporto del veicolo che lo effettua e la richiesta di ogni cliente deve essere soddisfatta in un singolo viaggio (supponendo, chiaramente, che la sua richiesta sia minore o al più uguale alla capacità di trasporto del veicolo). La risoluzione di un problema VRP consiste nel trovare un insieme di percorsi che minimizzano la distanza totale percorsa, soddisfacendo tutte le richieste dei clienti.

Un caso particolare di VRP è il noto *Traveling Salesman Problem* (TSP), problema nobile dell'ottimizzazione combinatoria, in cui si ipotizza l'esistenza di un unico veicolo dotato di capacità di trasporto infinita. Come il TSP, il VRP rientra nell'ordine di complessità dei problemi NP-Hard, ossia esiste un problema NP-completo Turing-riducibile in tempo polinomiale al VRP.

In base alla possibilità o meno, da parte dei veicoli, di eseguire più di un percorso, si distinguono fondamentalmente due modalità di VRP:

- *Single-Trip* (ciascun veicolo effettua al più un percorso);
- *Multi-Trip* (un veicolo può effettuare diversi percorsi, con diversi autisti se necessario).

La durata totale dei percorsi assegnati ad un veicolo non deve superare il tempo massimo (working time) T_{MAX} di guida di ciascun autista.

2.1 Single-Trip VRP

Nel Single-Trip case un veicolo non può effettuare più di un percorso. É ben noto che la durata totale della soluzione finale tende ad aumentare quando la capacità Q del veicolo diminuisce.

Una regola usata dagli algoritmi euristi é quella generare iterativamente per ciascun veicolo un percorso utilizzando i veicoli con maggior capacità per primi,

ordinando l'insieme dei veicoli S_V per valori non crescenti di capacità. Indicando con m il numero di veicoli, si ha dunque che $Q_1 > Q_2 > \ldots > Q_m$. Nel caso in cui ogni veicolo serva una e una sola richiesta e n > m, si aggiungono n – m veicoli fittizi di cpacità inferiore a Q_m a completamento dell'insieme S_V, in modo tale da garantire l'esistenza di una soluzione ammissibile di partenza (questo caso è anche la configurazione iniziale per alcuni euristici). I dati in output degli algoritmi consistono in:

- un numero di percorsi *nt*,
- un insieme di percorsi $S_t = \{1, 2, \ldots, nt\}$,
- una durata totale *td*.

Nel single-trip case, il numero di veicoli usati *nu* è uguale al numero di percorsi *nt*. Ogni percorso p di S_t è definito da:

- una sequenza di clienti E_p;
- un carico L_p;
- una durata T_p;
- un veicolo compatibile tale che $L_p \leq Q_{V_p}$.

2.2 Multi-Trip VRP

Data la natura dei problemi pratici, la risoluzione tramite euristici basati sul Single-Trip case spesso richiede l'utilizzo di veicoli fittizi e li rende pertanto non risolvibili esclusivamente tramite la flotta attuale. Questo problema può essere superato permettendo a ciascun veicolo di effettuare diversi percorsi.

Nel Multi-Trip case, ciascun veicolo può effettuare più percorsi, purchè non venga violato il vincolo di working-time, ovvero la durata totale di viaggio percorso da un veicolo non deve superare il tempo massimo di lavoro T_{MAX}. Il numero di veicoli è, in questo caso, uguale al numero di multi-trips. L'idea più semplice per estendere gli euristici sequenziali al caso multitrip, è quella di costruire anche i multi-trips uno per uno. La costruzione dell'ultimo percorso di un multi-trip viene arrestata quando qualsiasi successivo inserimento supererebbe il massimo working

time T_{MAX} del veicolo. Alcune valutazioni numeriche mostrano che questa strategia fornisce un numero accettabile di percorsi, ma anche un costo totale maggiore rispetto a quella che si otterrebbe operando in modalità Single-Trip. Risultati migliori in termini di durata totale sono ottenuti se si esegue un euristico sequenziale in modalità Single-Trip e successivamente si riduce il numero dei risultanti nt percorsi ad un numero minimo di veicoli. Ciò è possibile grazie ad una procedura chiamata *PackTrips*.

2.2.1 ***PackTrips Procedure***

La procedura *PackTrips* (impacchettamento di percorsi) è un ottimo criterio di risoluzione per i cosiddetti *euristici Bin-Packing* (impacchettamento in contenitori), la cui funzione consiste appunto nell'assegnamento dei vari percorsi ai veicoli.

Ogni veicolo k può essere visto come un "contenitore" avente una capacità di carico Q_k ed una capacità di tempo T_{MAX}. Mentre la durata totale dei percorsi assegnati a k non deve superare T_{MAX}, qualsiasi percorso p assegnato al veicolo k deve solo avere un carico

$$L_p \leq Q_k,$$

dal momento che il veicolo viene riutilizzato.

La procedura PackTrips rientra nell'ordine di complessità $O(nt^2)$. Essa può essere considerata come un'estensione del classico euristico *First Fit Decreasing* (FFD). Quando la flotta è omogenea, PackTrips eredita il buon caso pessimo delle performance di FFD, asintoticamente pari a 11/9.

Nella soluzione ottenuta attraverso un euristico sequenziale, prima che venga applicata tale procedura, ciascun percorso è assegnato ad un veicolo e ciascuno di questi veicoli effettua un solo percorso. È ben visibile che la procedura PackTrips non fallisce mai se applicata a tali soluzioni Single-Trip: nel caso peggiore, infatti, costruisce multitrips ridotti ad un percorso ciascuno.

Noti metodi esatti per il *bin-packing problem* possono essere estesi per risolvere l'assegnamento *Trip-To-Truck* con capacità di carico dei veicoli differenti. Per una singola applicazione, dopo un euristico sequenziale, il tempo computazionale

dovrebbe essere ragionevole nel caso in cui il numero di percorsi da impacchettare sia relativamente piccolo. Tuttavia, tali metodi esatti risultano essere significativamente costosi qualora vengano applicati all'interno di euristici che eseguono un impacchettamento per ogni iterazione.

In pratica e in media, gli euristici bin-packing sono sufficienti, dal momento che fissano un lower bound LB pari all'indice del veicolo con minor capacità di trasporto tale che i veicoli da 1 a LB siano sufficienti per impacchettare gli nt percorsi, ossia:

$$LB = \min\{\, j \in \{1, 2, \dots, m\} : \sum_{i=1}^{nt} T_i \;\leq\; \sum_{k=1}^{j} Q_k \,\}$$

3. Metodi esatti

Il MTVRP è una variante del VRP in cui ciascun cliente può effettuare un sottoinsieme di percorsi. Questo sottoinsieme viene chiamato *vehicle schedule* (piano di veicoli) ed è soggetto a vincoli di durata massima. Nonostante la sua importanza nella pratica, il MTVRP ha fin ora ricevuto una scarsa attenzione in letteratura: sono stati sviluppati alcuni euristici e un solo metodo esatto, presentato da Mingozzi et al. (2013). Nel corso di questo capitolo, sono descritte le formulazoni matematiche su cui si basa tale metodo.

3.1 L'algoritmo di Mingozzi, Roberti e Toth

Il MTVRP può essere descritto come segue: sia $G = (V', E')$ un grafo non direzionato, dove V' è l'insieme dei vertici ed E è l'insieme degli archi. Si ha $V' = \{0\} \cup V$, dove il vertice 0 rappresenta il deposito e l'insieme $V = \{1, \dots, n\}$ rappresenta gli n clienti, ognuno dei quali richiede q_i unità di prodotto dal deposito. Ogni vincolo ha una capacità Q e un tempo massimo di guida T. Ad ogni arco $(i, j) \in E$ è associato un costo α_{ij} e un tempo di viaggio τ_{ij}. Si assume che tutti i dati in input siano non-negativi. Un percorso effettuato da un veicolo è visto come un ciclo elementare a costo minimo in G che passa attraverso il deposito e un sottoinsieme di clienti tali che la domanda totale dei clienti visitati non superi la capacità Q del veicolo. Il costo di una route è dato dalla somma dei costi degli archi attraversati. Uno *schedule* di un veicolo è un sottoinsieme di routes aventi una durata totale minore o uguale al massimo working time T. Il costo di uno schedule è uguale alla somma dei costi delle sue routes. Il MTVRP prevede la progettazione di un insieme di m schedules di costo totale minimo tale che ogni cliente sia visitato esattamente una volta dalle routes degli schedules.

Il metodo esatto di Mingozzi et al. (2013) si basa su due formulazioni *set-partioning-like* (di partizionamento dell'insieme) del MTVRP, chiamate $(F1)$ e $(F2)$. La prima prevede una variabile binaria per ogni route e una variabile che, per ciascun veicolo, indica se un dato percorso è assegnato o meno alla schedula di quel veicolo. La seconda utilizza una variabile binaria che indica, per ogni schedula, se essa viene effettuata o meno. La presenza di tali variabili rende questo problema un problema di *programmazione intera binaria*. Come nel caso più generale della

programmazione lineare intera (PLI), i problemi di programmazione binaria sono relativamente *difficili* da risolvere. In questo caso, tuttavia, è possibile ottenere delle limitazioni superiori ed inferiori del valore ottimo della funzione obiettivo. Consideriamo un generico problema di programmazione binaria

$$P = \begin{cases} z(P) = min & \sum_{j=1}^{n} c_j x_j & & (3.1) \\ s.t. & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & i = 1, 2, \dots, m & (3.2) \\ & x_j \in \{0, 1\} & j = 1, 2, \dots, n & (3.3) \end{cases}$$

A partire da questo problema è possibile costruire uno o più problemi, detti *rilassamenti*, che forniscono, nel caso di un problema di minimizzazione, una limitazione inferiore del valore ottimo della funzione obiettivo (3.1). Definiamo *lower bound* tale limite inferiore. Consentendo alle variabili binarie di poter assumere un qualsiasi valore reale compreso tra 0 e 1, ossia sostituendo la (3.3) con il vincolo

$$0 \leq x_j \leq 1 \qquad j = 1, 2, \dots, n \qquad (3.4)$$

trasformiamo il precedente problema P di PLI in un problema di programmazione linerare (PL). Il valore ottimo di un rilassamento è minore o uguale al valore ottimo del problema originario.

Il metodo di risoluzione di Mingozzi, Roberti e Toth si basa sui *rilassamenti lineari* di entrambe le formulazioni e sulla relazione che intercorre tra i relativi *lower bounds*. Inoltre prevede l'utilizzo di una procedura che usa i lower bounds raggiunti attraverso processi di costruzione per generare sia un insieme ridotto di tutte le routes di $(F1)$, sia un insieme ridotto di tutti le schedule di $(F2)$, contenenti qualsiasi soluzione MTVRP ottima. I risultanti problemi ridotti sono risolti direttamente mediante l'utilizzo di un *solver* di programmazione intera. Nei paragrafi seguenti sono riportate le formulazioni matematiche $(F1)$ e $(F2)$ e la relazione che intercorre tra i corrispondenti rilassamenti lineari. Infine, è presente una breve descrizione riguardo i risultati raggiunti da tale metodo.

3.2 La Formulazione Matematica (F1)

Sia $\mathcal{R}$ l'insieme degli indici di tutti le routes possibili nel grafo G e sia $\mathcal{R}_i \subseteq \mathcal{R}$ il sottoinsieme degli indici delle routes che visitano i clienti $i \in V$. Ad ogni route $l \in \mathcal{R}$ è associato un costo d_l e una durata τ_l. Indichiamo con $\mathcal{R}_l$ e $E(\mathcal{R}_l)$ l'insieme dei clienti visitati e l'insieme degli archi attraversati dalla route $l \in \mathcal{R}$, rispettivamente. Sia ξ_l^j una variabile binaria uguale a 1 se e solo se la route $l \in \mathcal{R}$ è assegnata al veicolo $j \in M$. La formulazione matematica $(F1)$ è la seguente:

$$F1 = \begin{cases} z(F1) = min & \sum_{l \in \mathcal{R}} d_l \sum_{j \in M} \xi_l^j & & (3.5) \\ s.t. & \sum_{l \in \mathcal{R}_i} \sum_{j \in M} \xi_l^j = 1 & \forall i \in V & (3.6) \\ & \sum_{l \in \mathcal{R}} \tau_l \xi_l^j \leq T & \forall j \in M & (3.7) \\ & \xi_l^j \in \{0,1\} & \forall j \in M, \forall l \in \mathcal{R} & (3.8) \end{cases}$$

I vincoli (3.6) impongono che ogni cliente sia visitato esattamente una volta; i vincoli (3.7) definiscono la schedula ammissibile per ogni veicolo usato.

3.3 La Formulazione Matematica (F2)

Sia $\mathcal{H}$ l'insieme degli indici di tutti i possibili schedules. Per ogni schedula $k \in \mathcal{H}$, indichiamo con $\Omega_k \subseteq \mathcal{R}$ il sottoinsieme degli indici delle routes della schedula, con $c_k = \sum_{l \in \Omega_k} d_l$ il suo costo, e con $\tau(\Omega_k) = \sum_{l \in \Omega_k} \tau_l$ la sua durata totale. Inoltre, definiamo $V(\Omega_k) = \bigcup_{l \in \Omega_k} \mathcal{R}_l$ e $E(\Omega_k) = \bigcup_{l \in \Omega_k} E(\mathcal{R}_l)$. Assumiamo che $\mathcal{H}$ contenga solo schedule non dominate, ossia, dato $k \in \mathcal{H}$, non esiste $k' \in \mathcal{H} \setminus \{k\}$ tale che $V(\Omega_k) = V(\Omega_{k'})$ e $c_k > c_{k'}$. Si noti che, ogni volta che $m = 1$, $\mathcal{H}$ contiene solo le schedule corrispondenti alle soluzioni ottime intere del CVRP, dove la durata totale delle routes è minore o uguale a T. Sia y_k una variabile binaria uguale a 1 se e solo se lo schedule $k \in \mathcal{H}$ è assegnato a un veicolo. La formulazione matematica $(F2)$ è la seguente:

$$F2 = \begin{cases} z(F2) = min \quad \sum_{k \in \mathcal{H}} c_k y_k & (3.9) \\ s.t. \quad \sum_{k \in \mathcal{H}: i \in V(\Omega_k)} y_k = 1 \quad \forall i \in V & (3.10) \\ \sum_{k \in \mathcal{H}} y_k \leq m & (3.11) \\ y_k \in \{0,1\} \quad \forall k \in \mathcal{H} & (3.12) \end{cases}$$

I vincoli (3.10) specificano che ogni cliente $i \in V$ deve essere visitato esattamente una volta. Il vincolo (3.11) impone un upper bound sul numero di veicoli usati.

3.4 Relazione tra i Rilassamenti di (F1) e (F2)

Indichiamo con $(LF1)$ il rilassamento lineare di $(F1)$ e con $z(LF1)$ il costo della sua soluzione ottima. Analogamente, indichiamo con $(LF2)$ il rilassamento lineare di $(F2)$ e con $z(LF2)$ il costo della sua soluzione ottima. Si ha:

$$z(LF1) \leq z(LF2).$$

3.5 Risultati Computazionali

L'algoritmo è stato testato su un sottoinsieme di istanze standard proposte da Taillard et al. (1996), usate per testare tutti gli algoritmi euristici presenti in letteratura. Queste istanze sono generate a partire dai grafi, dalle richieste dei clienti e dalle capacità dei veicoli di cinque problemi CVRP (CMT-1, CMT-2, CMT-3, CMT-11 e CMT-12), proposti da Christofides et al. (1979) con una variazione del numero di veicoli m. Per ogni problema e per ogni valore di m, sono state generate due istanze con diversi valori di tempo massimo di guida T: nella prima istanza, $T = [1{,}05 z_{RT}/m]$, e nella seconda, $T = [1{,}10 z_{RT}/m]$, dove $[x]$ indica il valore intero più vicino a x e z_{RT} è il costo della soluzione CVRP ottenuto da Rochat e Taillard (1995). Tutte le istanze utilizzano coordinate $x - y$. I costi di viaggio coincidono con i tempi di viaggio e sono valori reali calcolati come distanze Euclidee tra i vertici. L'algoritmo esatto può risolvere all'ottimo 42 delle 52 istanze di test. Solo una di queste istanze è stata risolta in un tempo di CPU di un'ora. Trentacinque delle 42 istanze risolte all'ottimo hanno restituito un costo della soluzione ottima pari al costo delle corrispondenti istanze CVRP. Le altre sette istanze hanno restituito un costo della soluzione ottima leggermente più alto rispetto

al costo delle corrispondenti istanze CVRP. I risultati computazionali mostrano inoltre che l'algoritmo di Mingozzi, Roberti e Toth può risolvere all'ottimo istanze che coinvolgono più di 120 clienti.

4. Metodi euristici

La grande complessità dei problemi reali di VRP rende difficile il calcolo della soluzione ottima tramite metodi di risoluzione esatti. Per tale motivo, si usa procedere attraverso algoritmi euristici. In relazione alla costruzione delle routes componenti la soluzione, distinguiamo fondamentalmente due classi di metodi euristici: *sequenziali* e *paralleli*. I primi si basano sulla creazione progressiva di routes, ovvero sulla costruzione di una nuova route ogni qualvolta nessun cliente non servito possa entrare a far parte della route attualmente in costruzione. I secondi prevedono una soluzione che si compone di più route emergenti che vengono espanse in parallelo inserendo i clienti non ancora serviti.

4.1 Euristici sequenziali

Il principio degli euristici sequenziali è quello di costruire una route alla volta. Di seguito è mostrato lo pseudocodice della struttura principale di un euristico sequenziale. Nel caso in cui siano utilizzati veicoli fittizi per la costruzione della soluzione finale, essa risulta non ammissibile per la corrente flotta di veicoli. Nella pratica, l'utente può comunque decidere di accettare ed attuare tale soluzione affittando i veicoli mancanti.

$$\mathrm{nt} := 0; \tag{4.1}$$
$$\mathrm{td} := 0; \tag{4.2}$$
$$\mathrm{U} := \mathrm{S_C}; \tag{4.3}$$
$$\text{while } \mathrm{U} \neq \mathrm{NULL} \tag{4.4}$$
$$\quad \mathrm{nt} := \mathrm{nt} + 1; \tag{4.5}$$
$$\quad \mathrm{H}\,(\mathrm{U}, \mathrm{Q_{nt}}, \mathrm{E_{nt}}, \mathrm{L_{nt}}, \mathrm{T_{nt}}); \tag{4.6}$$
$$\quad V_{nt} = nt; \tag{4.7}$$
$$\quad td = td + T_{nt} \tag{4.8}$$
$$\quad U = U \setminus E_{nt} \tag{4.9}$$

ossia:

- il numero di routes nt è inizializzato a zero (4.1);
- la durata totale td è inizializzata a zero (4.2);
- nell'insieme U sono inseriti i clienti ancora non visitati S_C (4.3);

- finchè esistono clienti non visitati, viene incrementato il numero nt di percorsi (4.5); viene eseguita una procedura H che costruisce un percorso ammissibile nt (4.6); il percorso nt è assegnato al veicolo V_{nt} (4.7); alla durata totale td viene sommata la durata del percorso appena costruito (4.8); l'insieme dei clienti serviti da tale percorso sono eliminati dall'insieme U.

La procedura $\mathrm{H}\left(\mathrm{U}, \mathrm{Q_k}, \mathrm{E_p}, \mathrm{L_p}, \mathrm{T_p}\right)$, dato l'insieme U dei clienti liberi (ancora non visitati), la capacità di trasporto Q_k del veicolo k, una sequenza di clienti E_p serviti dal percorso p, un carico L_p trasportato dal percorso p tale che $L_p \leq Q_k$ e la durata T_p del percorso p tale che $T_p \leq T_{MAX}$, seleziona un nuovo percorso p per un veicolo k.

Nei sottoparagrafi seguenti, sono descritti i concetti chiave di alcuni metodi euristici per il MTVRP presenti in letteratura.

4.1.1 *Euristico Nearest Neighbor (NNB)*

Mentre nella maggior parte dei metodi euristici per il MTVRP un percorso è inteso come un circuito che, partendo da un'origine, serve un certo numero di clienti, per poi tornare al punto di partenza, nel *Nearest Neighbor Heuristic*, il viaggio che emerge è un percorso aperto a partire dal deposito, non un circuito. Ogni iterazione collega l'ultimo nodo del percorso al client libero e ammissibile più vicino, in modo da avere abbastanza tempo per tornare al deposito entro il termine $TMAX$. Quando il viaggio non può più essere esteso, viene aggiunto un arco di ritorno al deposito.

4.1.2 *Euristico Best Insertion (BIS)*

Nel *Best Insertion Heuristic,* il viaggio è inizializzato come un loop sul deposito. Ogni iterazione enumera i possibili inserimenti di ogni cliente libero tra nodi consecutivi del viaggio. Viene eseguito l'inserimento con l'aumento minimo della durata.

4.1.3 *Euristico di Gillet e Miller (GIL)*

L'euristico di Gillett e Miller (1974) richiede le coordinate geografiche di ogni nodo. I clienti di S_C sono ordinati in ordine crescente dell'angolo polare che

formano con un raggio arbitrario centrato presso il deposito. Un percorso risultante p inizia con il primo cliente libero di S_C (quello con l'angolo polare più piccolo) e inserisce i clienti consecutivi di S_C per cui

$$L_p \leq Q_p$$

Ogni inserimento è seguito da una procedura di Local Search 2-opt, la quale sarà descritta nel capitolo 5.

4.1.4 *Euristico di Mole e Jameson (MOL)*

L'euristico di Mole e Jameson (1976) usa un criterio di risparmio ponderato con due parametri positivi, μ e φ. Il percorso risultante inizia con il cliente libero più lontano. Ad ogni iterazione è calcolato un costo di inserimento ponderato

$$A(i,k,j) = d_{ip} + d_{pj} - \mu d_{ij}$$

per ogni cliente libero p e ogni coppia *(i,j)* di clienti adiacenti nel percorso. Sia (i_p, j_p) la miglior posizione di inserimento e $A^*(i_p, p, j_p)$ il corrispondente costo minimo. Allora il miglior cliente p^*da inserire nel percorso massimizza

$$B = \varphi d_{1p} - A^*(i_p, p, j_p)$$

su tutti i client non instradati p che possono essere inseriti. Questo è inserito tra i_{p^*} e j_{p^*}. Il percorso è ottimizzato con una procedura di 2-opt dopo ogni iterazione.

4.1.5 *Euristico Due Fasi di Christofides, Mingozzi e Toth (CHR)*

Nella prima fase dell'euristico *Due Fasi* di Christofides et al. (1979), sono rapidamente costruiti percorsi per avere un insieme *L* di seeds (clienti iniziali) per i percorsi definitivi costruiti nella seconda fase. Ogni percorso risultante è inizializzato come un loop $(1, a, 1)$ nel cliente libero più lontano a, dove 1 è l'identificativo del nodo che indica il deposito. Questo seed a è memorizzato in *L*. Ogni iterazione considera ciascun cliente p libero e ammissibile per calcolare il punteggio

$$S = d_{1p} + \varphi d_{pa} \quad , \varphi \geq 1$$

ed inserisce il cliente che minimizza S, alla posizione che minimizza l'aumento della durata del percorso. Il percorso è poi immediatamente ottimizzato con una procedura di 2-opt. Questo processo di inserimento continua finché non vi sono altri clienti che possono entrare nel percorso. In altre parole, questa fase tende ad inserire, nel percorso inizializzato ad a, i clienti che sono vicini al segmento che unisce 1 ad a. Alla fine della fase 1, i percorsi provvisori vengono cancellati ed L contiene un insieme di seeds ben distribuiti.

Nella fase 2, il costo di inserimento di un cliente p per un seed a in L è definito come

$$g(p,a) = d_{1p} + \mu d_{pa} - d_{1a}, \quad \mu \geq 1$$

Ogni seed a viene quindi eliminato da L per iniziare un nuovo percorso. Si calcola l'insieme X di clienti raggiungendo i loro costi più piccoli per questo seed e, per ogni cliente p di X, un tipo di *rammarico*:

$$R_p = Min\{\, g\,(p,r\,)|r \in L\,\} - g\,(p,a\,)$$

Questo rammarico misura l'aumento del costo nel caso in cui p venga inserito nel suo secondo miglior percorso, invece del suo percorso migliore basato su a. Viene inserito quindi il cliente ammissibile di X di massimo rammarico, alla locazione che minimizza l'aumento della durata del percorso. Si ottimizza infine il percorso risultante con la procedura 2-opt. Se alcuni clienti sono ancora liberi dopo la costruzione del percorso per l'ultimo seed di L, si applicano ricorsivamente le fasi 1 e 2 a questo insieme, finché tutti i clienti non sono raggiunti.

4.2 Local Search

Le soluzioni di tutti i precedenti euristici possono essere migliorate con algoritmi di ricerca locale. Nel corso di questa tesi saranno trattati nel dettaglio due algoritmi di local search: 2-opt e 3-opt. In generale, la local search parte da una soluzione iniziale ammissibile e cerca di migliorarla esplorando il suo intorno. L'intorno deve essere propriamente definito in base al problema di ottimizzazione combinatoria affrontato. Generalmente i tempi computazionali degli algoritmi di local search sono maggiori di quelli degli algoritmi costruttivi. Un algoritmo di local search opera nel seguente modo:

- *Step 0:* Sia x_c una soluzione iniziale e sia Z_c il suo costo
- *Step 1*: Sia $x_b = x_c$ e $Z_b = Z_c$
- *Step 2:* per ogni soluzione $x \in I(x_c)$, se $Z_x < Z_c$, allora $x_c = x$ e $Z_c = Z_x$
- *Step 3:* Se $(Z_c < Z_b)$ torna allo *Step 1*

dove $I(x)$ è l'intorno della soluzione x, ossia l'insieme delle soluzioni che possono essere generate modificando x. La maggior parte delle Local Search applicate ai problemi di Vehicle Routing e schedulazione, sono algoritmi *edge-exchange*, ossia implementano degli scambi di archi. Le soluzioni vicine per una singola route sono un insieme di percorsi ottenuti rimpiazzando un insieme di k archi con un altro insieme di k archi. Questo meccanismo viene chiamato *k-exchange*, e un percorso che non può più essere migliorato dopo un k-exchange è detto *k-ottimale*. Verificare un k-ottimale richiede un tempo $O(nk)$.

Nel caso di un Multi-Trip, la Local Search ha bisogno di essere adattata per preservare l'ammissibilità dell'assegnamento *Trip-To-Truck* ottenuto con la procedura *PackTrips* al termine di un euristico: le modifiche in cui non è rispettato il working time dei veicoli devono ora essere scartate. Come già esposto al punto *2.1.1*, eseguire PackTrips ad ogni tentativo di modifica è possibile ma costoso in termini di tempo. Inoltre, esperimenti computazionali mostrano che un packing immediato ad ogni mossa può *strangolare* procedure di miglioramento assegnando percorsi ai veicoli fino al loro tempo massimo di lavoro: così, la maggior parte delle mosse diventano impossibili. Questo è il motivo per cui, nel Multi-Trip, è consigliabile applicare PackTrips solo alla fine.

4.3 Tabu Search

La *Tabu Search* (ricerca tabu) è una tecnica metaeuristca utilizzata per la soluzione di numerosi problemi di ottimizzazione, tra cui problemi di scheduling e routing, problemi su grafi e programmazione intera. Per tecnica metaeuristca si intende un metodo euristico per la soluzione di una classe molto ampia di problemi computazionali combinando diverse procedure a loro volta euristiche, con l'obiettivo di ottenere una procedura più robusta ed efficiente. La tecnica Tabu Search viene proposta da Glover (1989) come metodo per continuare la ricerca oltre i minimi locali. La Tabu Search rappresenta una evoluzione della Local Search che,

come detto nel paragrafo precedente, è utilizzata per trovare il minimo di una funzione f su un insieme S, dove S è lo spazio delle soluzioni. Essa parte da una soluzione iniziale ed esegue una serie di *mosse* che portano ad una nuova soluzione all'interno del *vicinato* (o insieme di adiacenza) della soluzione corrente, nel quale la funzione obiettivo f assume un valore minore del valore attuale. Il difetto della Local Search sta nel fatto che, se nell'insieme di adiacenza non esistono soluzioni migliori di quella corrente, la ricerca si arresta. La soluzione ottima individuata dalla ricerca locale risulta quindi associata ad un minimo locale dello spazio delle soluzioni, il quale è spesso ben lontano dalla soluzione ottima globale.

Per prima cosa, al fine di poter sfuggire alla *trappola* dei minimi locali, la Tabu Search consente mosse *peggioranti*. A questo punto occorre impedire le mosse eseguite recentemente, per evitare di ricadere subito dopo nel minimo locale. Il metodo Tabu Search consiste fondamentalmente nel rendere *proibite* (tabu) le ultime mosse eseguite nel cammino di ricerca, in modo che l'algoritmo non ricada nel minimo locale. Occorre pertanto tener traccia di alcune informazioni relative all'itinerario percorso: esse saranno impiegate per guidare la mossa dalla soluzione corrente a quella successiva, scelta all'interno dell'insieme di adiacenza.

4.4 Metodo di Taillard per il MTVRP

Il VRP con utilizzo multiplo di veicoli (MTVRP) è una variante dello standard VRP in cui gli stessi veicoli devono essere assegnati a diversi percorsi durante un dato periodo di pianificazione. Nell'articolo di Taillard et al. (1996) è descritto un efficiente Tabu Search per questo tipo di problema. Nonostante progettare percorsi con uso multiplo di veicoli sia piuttosto importante nella pratica, ad oggi il MTVRP ha ricevuto poca attenzione nell'ambito della Ricerca Operativa: solo Fleischmann (1990) ha affrontato in maniera diretta questo problema. Nel suo articolo, egli propone un euristico per il MTVRP basato sui savings e lo illustra con esempi in cui il numero dei clienti varia tra 68 e 361.

Un semplice euristico per il MTVRP consiste nell'abbassare artificialmente il valore del working time dei veicoli L ad un valore L', producendo così diversi percorsi corti per poi combinarli mediante un algoritmo di *bin-packing*. Tuttavia, scegliere il valore di L' non è banale; inoltre, in generale, la scelta di un valore di L' molto basso porta ad una bassa probabilità di trovare buone soluzioni. Nell'articolo

sopracitato, Taillard et al. (1996) propongono un nuovo euristico per questo problema.

Negli ultimi anni, sono stati proposti diversi efficienti algoritmi per il VRP (Taillard (1993), Gendreau et al. (1994), Rochat e Taillard (1995)). Generalmente, producono soluzioni molto buone e a volte ottime. Tuttavia, si presentano spesso casi in cui la ricerca rimane *intrappolata* in un minimo locale e le tecniche standard di diversificazione non sono abbastanza efficienti per risolvere tale situazione.

L'algoritmo di Rochat e Taillard (1995) permette una diversificazione del processo di ricerca, generando e combinando soluzioni, in modo simile a quanto eseguito da algoritmi genetici. Più precisamente, la procedura di generazione del percorso produce inizialmente diverse buone soluzioni VRP utilizzando un algoritmo di Tabu Search. Poi estrae singoli percorsi da questa *popolazione* di soluzioni, e combina alcuni di questi percorsi per definire una soluzione parziale di partenza per una nuova applicazione di Tabu Search. Questo processo è ripetuto un certo numero di volte e alcuni dei percorsi generati sono selezionati come candidati per la soluzione finale VRP. È importante notare che ogni applicazione di Tabu Search ha l'effetto di produrre una soluzione VRP di partenza completa da un insieme limitato di percorsi e può inoltre modificare tali percorsi attraverso il processo di ricerca locale.

L'algoritmo proposto da Taillard et al. (1996) è basato sul principio di *Rochat-Taillard*. Esso si compone fondamentalmente di tre parti: per prima cosa viene generato un vasto insieme di percorsi buoni, soddisfacendo i vincoli del VRP; viene poi generata una soluzione di un sottoinsieme di questi percorsi usando un algoritmo enumerativo; infine, i percorsi selezionati vengono assemblati (purchè non sia violato il vincolo di working time dei veicoli) mediante diverse applicazioni di un euristico *bin-packing*.

L'idea della prima generazione di percorsi individuali e della successiva combinazione di essi in una soluzione globale era già stata implementata da altri autori, quali Foster e Ryan (1976), Ryan et al. (1993), Rochat e Taillard (1995), Renaud et al. (1996). Rispetto ai precedenti lavori, l'algoritmo parallelo di Rochat e Taillard produce un insieme molto più ampio di percorsi e la qualità media di tali percorsi è anche maggiore. Inoltre esso può gestire facilmente una varante del

VRPM in cui sono considerate *sanzioni* per il lavoro straordinario, ossia il lavoro che non rientra nei tempi durante i quali il deposito è aperto, cosa che nella pratica accade molto spesso. L'algoritmo di Taillard è illustrato con maggior dettaglio nei paragrafi seguenti.

4.4.1 ***Parte 1: generazione di un percorso***

La prima parte dell'algoritmo consiste nella generazione di soluzioni VRP iniziali. Più precisamente, vengono generate, mediante l'algoritmo Tabu Search di Taillard, h soluzioni VRP con un numero non specificato di veicoli, dove h è un parametro dato in input. I singoli percorsi sono poi inseriti all'interno di una lista e ad ognuno di essi viene associata un'etichetta che riporta il valore della soluzione VRP.

Una volta generate le soluzioni VRP iniziali, vengono eseguite le seguenti operazioni p volte, dove p è un parametro dato in input:

1) Selezionare casualmente un percorso dalla lista, in base a un criterio che dia maggiore peso ai percorsi che sono generati spesso o che appartengono alle migliori soluzioni VRP. Più nello specifico, siano i percorsi nella lista etichettati con $R_1, \dots, R_\sigma$ e sia z_i il valore della soluzione VRP da cui R_i è stato estratto: si ordinino i percorsi in modo che $z_1 \leq z_2 \leq \cdots \leq z_\sigma$ e si assegni al percorso R_i una probabilità di selezione
$$P_i = 2(\sigma - i + 1) / \sigma(\sigma + 1).$$
2) Ignorare tutti i percorsi che hanno vertici comuni ai percorsi già selezionati; se rimangono altri percosi, tornare al punto 1.
3) Usando i percorsi selezionati al punto 1. come punto di partenza; applicare una Tabu Search per generare una nuova soluzione VRP e aggiungere alla lista i percorsi individuali, etichettati come indicato in precedenza. Eliminare i percorsi *dominati* R_i domina R_j se entrambi i percorsi hanno lo stesso insieme di vertici, ma R_j s è almeno lungo quanto R_i). Se un percorso è duplicato, mantenerne solo una copia, ma registrare la frequenza di tali percorsi poichè influenzerà la probabilità di scelta degli stessi.

4.4.2 *Parte 2: generazione di nuove soluzioni VRP*

Nella seconda parte dell'algoritmo sono selezionati al massimo q percorsi, dove q è un parametro dato in input e $q \gg m$. Tipicamente, il numero di percorsi selezionati è q, ma ve ne possono essere di meno se la dimensione della lista è minore di q. I percorsi sono selezionati in ordine non decrescente delle loro etichette e inseriti in un insieme J. Questa regola di selezione garantisce l'esistenza di una soluzione VRP ammissibile. Poi, all'interno dell'albero di ricerca, sono generate tutte le soluzioni ammissibili VRP che possono essere costruite attraverso una combinazione di percorsi di J. Al fine di controllare la crescita dell'albero di ricerca, la priorità è data ai percorsi che contengono il maggior numero di clienti. Questo processo termina con un insieme K di soluzioni VRP ammissibili.

4.4.3 *Parte 3: generazione di soluzioni per il MTVRP*

Nell'ultima parte dell'algoritmo, viene effettuato un tentativo di ottenere una soluzione ammissibile al MTVRP, risolvendo un problema di *packing* per ogni soluzione VRP di K, e viene selezionata la soluzione migliore di tutte. Per ogni soluzione VRP k di K, sia $f_{k,l}$ la durata dell'l-esimo percorso, dove $l = 1, \dots, m_k$, dove m_k è il numero di percorsi nella soluzione k. Poi, per ogni k, è identificata una soluzione MTVRP ogni volta che esiste una soluzione bin-packing ammissibile con m *contenitori* identici di dimensione M e m_k *oggetti* di grandezza $f_{k,1}, f_{k,2}, \dots, f_{k,m_k}$. Per identificare tali soluzioni bin-packing, tutti gli oggetti sono prima ordinati per valori non crescenti di grandezza e gradualmente assegnati al contenitore che possiede almeno quella grandezza.

In contrapposizione ad un algoritmo esatto per il problema del bin-packing, il tempo computazionale di un euristico è più stabile e prevedibile e, nella pratica, la perdita di qualità che emerge dall'utilizzo di un euristico è solitamente trascurabile. In caso contrario, può essere effettuato un tentativo di ottenere una soluzione ammissibile scambiando ripetutamente oggetti che appartengono ai diversi contenitori. L'ammissibilità, in questo caso, non è garantita. Quando è consentito il lavoro fuori orario, qualsiasi ora di lavoro oltre il tempo M contribuisce ad una penalità di un fattore θ (quando $\theta = 0.5$, il lavoro straordinario è pagato il 50% in più rispetto al prezzo standard). L'algoritmo di Taillard, nello step degli scambi,

tenta di minimizzare il lavoro straordinario totale. In questo caso, esiste sempre una soluzione MTVRP ammissibile.

L'algoritmo descritto contiene tre parametri: h, p e q. Sia nell'implementazione di Rochat e Taillard, sia in quella sopra illustrata, il valore di h è stato impostato a 20; il valore di p dovrebbe essere proporzionale alla dimensione del problema; il valore di q è tipicamente qualche centinaia.

4.5 Merge Heuristic di Prins (MER)

Contrariamente agli euristici sequenziali, il Merge Heuristic di Prins (2002) inizia con un insieme completo di $nt = n$ percorsi triviali, ciascuno dei quali contiene uno ed un solo cliente. Ogni successiva iterazione di MER concatena due percorsi, diminuendo la durata totale, purchè le concatenazioni siano possibili. La soluzione iniziale è certamente molto costosa, ma ammissibile, dal momento che:

- qualsiasi richiesta può essere trasportata da qualsiasi veicolo;
- un veicolo può raggiungere qualsiasi cliente e tornare al deposito in un tempo non superiore a T_{MAX};
- n veicoli sono disponibili grazie alla somma dei veicoli fittizi ad S_V.

Per concatenare coppie di veicoli, viene usato il concetto di saving: il saving s_{ij} di un arco (i,j) è il guadagno ottenuto nel caso in cui i clienti i e j vengano serviti all'interno dello stesso percorso, invece che in due percorsi separati:

$$s_{ij} = d_{1i} - d_{ij} + d_{j1}$$

MER costruisce una lista H di archi, ordinati per valori decrescenti di saving. Il processo di concatenazione testa in questo ordine ogni arco (i,j) in H. Siano u e v i percorsi che contengono, rispettivamente, i clienti i e j. Quando le concatenazioni avranno creato percorsi con almeno tre clienti, i e j non saranno necessariamente alle estremità di questi percorsi. Quindi, u e v dovranno essere concatenati solo se i continua ad essere un estremo del percorso u e j continua ad essere un estremo del percorso v. Naturalmente, questa operazione deve produrre un percorso ammissibile.

In presenza di flotte eterogenee, ossia composte da veicoli di differenti dimensioni e quindi con diverse capacità di carico, nel caso in cui i percorsi vengano assegnati sempre allo stesso veicolo, il processo di concatenamento può terminare presto, per insufficiente capacità nei veicoli. Il rimedio consiste nel controllare l'esistenza di un nuovo assegnamento ammissibile ogni qualvolta si provi ad unire due percorsi u e v con un arco (i, j). Questo può essere implementato mediante la semplice procedura *TryToAssign*, che può essere descritta come segue:

- costruzione di una lista R contenente una coppia (Lp, p) per ogni percorso corrente $p \neq u, v$, più una per il nuovo percorso u che si vuole far risultare dall'unione di u e v;
- ordinamento di R per valori decrescenti di carico;
- scansionamento delle coppie di R e assegnamento di ciascuna di esse al primo veicolo conveniente (si ricordi che l'insieme dei veicoli S_V è ordinato per valori decrescenti di capacità di trasporto);
- se l'assegnamento è stato completato con successo, è memorizzato in un vettore A, con $A_i = j$ se il percorso i è stato assegnato al veicolo j.

La complessità di tale procedura è dominata dalla complessità $O(n \log n)$ della fase di ordinamento. *TryToAssign* è solo un tentativo di trovare un nuovo assegnamento, calcolato con l'ausilio di un array A al fine di preservare l'assegnamento corrente.

Se l'assegnamento è stato trovato, i percorsi u e v possono essere uniti. Il nuovo assegnamento può essere confermato mediante la semplice procedura *Assign*, che sostituisce il nuovo assegnamento a quello corrente.

L'algoritmo MER proposto da Prins (2002) può essere descritto brevemente come segue:

- *Step 0*: Costruzione di una soluzione iniziale;
- *Step 1*: Costruzione di un albero H i cui nodi rappresentano i savings di ciascuna coppia di clienti;
- *Step 2*: Scansione dei nodi per individuare merges fattibili:
    a. Viene preso il nodo (i, j) col massimo saving s_{ij};

b. Siano u e v i percorsi che contengono, rispettivamente, i clienti i e j. Se i e j appartengono ancora all'insieme degli estremi, rispettivamente, del percorso u e del percorso v, e la durata complessiva dell'eventuale percorso risultato dall'unione di u e v non supera il working time dei veicoli T_{MAX}, allora viene eseguita la procedura *TryToAssign*. Se l'assegnamento è completato con successo, viene effettuato il *merge* dei percorsi u e v mediante la procedura *Assign* e il percorso v viene eliminato. Se vi sono ancora nodi nell'albero H, tornare allo *Step 3*.

Nei paragrafi seguenti saranno affrontati nel dettaglio i passi di MER appena illustrati.

4.5.1 ***Costruzione di una soluzione iniziale***

La soluzione di partenza di MER consiste nella creazione di $nt = n$ percorsi, ossia di tanti percorsi quanti sono i clienti da servire; l'i-esimo percorso rappresenta un viaggio di andata e ritorno dal deposito all'i-esimo cliente. Questa soluzione iniziale, che si presenta come i petali di un fiore, è solitamente chiamata *Daisy* (margherita). L'inizializzazione di un generico percorso p può essere descritta come segue:

$$E_p = (p) \quad (4.10)$$
$$L_p = q_p \quad (4.11)$$
$$T_p = 2 \times d_{1p} \quad (4.12)$$
$$V_p = p \quad (4.13)$$
$$\tau_p = p \quad (4.14)$$
$$td = td + T_p \quad (4.15)$$

ossia:

- L'insieme E_p dei clienti del p-esimo percorso contiene il p-esimo cliente (4.10);
- Il carico L_p del p-esimo percorso è uguale alla quantità richiesta dal p-esimo cliente q_p (4.11);

- La durata del p-esimo percorso è data dalla distanza d_{1p} dal deposito (nodo 1) al p-esimo cliente, moltiplicata per due, dal momento che si tratta di un viaggio di andata e ritorno (4.12);
- Il p-esimo veicolo serve il p-esimo cliente (4.13);
- τ è un vettore in cui è memorizzato, per ogni cliente, l'identificativo del percorso da cui è servito; la soluzione di partenza prevede che il p-esimo cliente sia servito dal percorso p (4.14);
- la durata totale è data dalla somma delle durate degli $n = nt$ percorsi (4.15).

4.5.2 *Costruzione dell'albero dei savings*

Il *saving sij* è il risparmio che si otterrebbe nel caso in cui i clienti i e j, attualmente serviti rispettivamente nell'i-esimo e nello j-esimo percorso, venissero serviti nello stesso viaggio. Come già detto nel paragrafo precedente, il costo, in termini di distanza percorsa, del percorso i è dato da: $2 \times d_{1i}$; analogamente, il costo del percorso *j* è dato da: $2 \times d_{1j}$. Il costo complessivo di questi due percorsi è quindi dato da:

$$T_i + T_j = (2 \times d_{1i}) + (2 \times d_{1j})$$

Sia ora k il percorso che, partendo dal deposito, giunga al cliente *i* e, prima di tornare all'origine, vada a servire il cliente *j*; è intuitivo notare che il costo di *k* sarà:

$$T_k = d_{1i} + d_{ij} + d_{j1}$$

Ossia:

$$T_k = T_i + T_j - d_{1i} - d_{j1} + d_{ij}$$

Il saving s_{ij} è dato dalla differenza $(T_i + T_j) - T_k$, ossia:

$$s_{ij} = d_{1i} - d_{ij} + d_{j1}$$

L'albero dei savings contiene n^2 nodi che rappresentano tutte le possibili coppie di clienti con i relativi savings.

4.5.3 ***Ricerca di merges fattibili***

Una volta costruito l'albero dei savings, MER procede con l'estrazione del nodo (i, j) che ha il massimo risparmio e valuta la possibilità di servire i clienti i e j attraverso un unico viaggio. Siano $u = \tau_i$e $v = \tau_j$ i percorsi contenenti, rispettivamente, i clienti i e j. Si noti che l'unione di due viaggi viene effettuata sulla base della coppia di clienti (i, j). Affinchè u e v possano essere uniti in un unico percorso, è necessario che i sia un estremo di u e j un estremo di v. Se così non fosse, si perderebbe la sequenza di clienti serviti da uno dei due singoli percorsi. Supponiamo, ad esempio, che il percorso u serva, nell'ordine, i clienti: 23, 36, 18 e 20 e che il percorso v serva i clienti: 40, 62, 58 e 70; supponiamo inoltre che il massimo saving venga rilevato in corrispondenza dei clienti 20 e 62. Si noti che, mentre il cliente 20 è un estremo o, più precisamente, l'ultimo cliente servito da u, il cliente 62 non è un estremo di v. In tal caso, effettuando il merge di u e v sulla base dei clienti 20 e 62, si otterrebbe un percorso che serve, nell'ordine, i clienti: 23, 36, 18, 20, 62, 58 e 70, ignorando il cliente 40. Pertanto, il cliente i deve essere il primo o l'ultimo cliente del percorso u e, analogamente, j deve essere il primo o l'ultimo cliente servito dal percorso v. Una volta soddisfatti questi vincoli, prima di effetturare il merge di u e v, occorre controllare che la distanza complessiva percorsa dal viaggio risultante rispetti il vincolo di working time, ossia:

$$T_u + T_v - s_{ij} \leq T_{MAX}$$

Inoltre, la somma delle quantità di prodotto richieste da tutti i clienti serviti da u e da v non deve superare la capacità Q_k del veicolo k che effettua tale percorso, ossia:

$$L_u + L_v \leq Q_k$$

Quest'ultimo vincolo viene verificato dalla procedura *TryToAssign* descritta nel paragrafo *4.4*. Se questi vincoli sono rispettati, può essere effettuato il merge dei percorsi u e v.

4.5.4 ***Unione di due percorsi***

Il merge dei percorsi u e v sarà sovrascritto nel percorso u, mentre il percorso v sarà eliminato. Esso può essere descritto come segue:

$$E_u = E_u \,||(i,j)||\, E_v \quad (4.7)$$
$$L_u = L_u + L_v \quad (4.8)$$
$$T_u = T_u + T_v - s_{ij} \quad (4.9)$$
Cancellazione del percorso v (4.10)
Chiamata della procedura $Assign$ (4.11)

ossia:

- Alla sequenza E_u dei clienti serviti dal percorso u, viene concatenata la sequenza E_v dei clienti serviti dal percorso v (4.7);
- al carico L_u trasportato dal percorso u viene sommato il carico L_v trasportato dal percorso v(4.8);
- la distanza percorsa da u è ora uguale alla somma delle distanze percorse da u e da v, meno il risparmio ottenuto unendo i nodi i e j (4.9);
- il percorso v viene cancellato (4.10);
- la procedura *Assign* sostituisce l'assegnamento A_p, risultato dalla procedura *TryToAssign*, del percorso p all'assegnamento corrente V_p (4.11).

Dopo aver effettuato il merge dei percorsi u e v, l'algoritmo procede con una nuova ricerca di merges fattibili nell'albero dei savings. Il processo termina quando non vi sono più nodi nell'albero.

4.5.5 ***Assunzioni di MER***

Si noti che, se la più grande richiesta dei clienti è maggiore della capacità del veicolo avente la minima capacità di carico, un euristico sequenziale può trascurare un cliente con una grande richiesta., e andare in *deadlock* (stallo) quando tutti i veicoli adatti a ciascun cliente sono già stati usati. Un vantaggio di MER è dato dal fatto che può essere fermato ad ogni iterazione, durante la generazione di una soluzione ammissibile. Fa eccezione il caso in cui una soluzione ammissibile sia ottenuta mediante l'utilizzo di veicoli fittizi; in questo caso, il problema può essere risolto rendendo tali veicoli reali, il che, nella pratica, consiste nell'affittare un numero $n - m$ di veicoli, dove, come indicato nel paragrafo 2.1, $n - m$ è il numero di veicoli fittizi utilizzati per garantire l'ammissibilità della soluzione anche nel caso pessimo. L'algoritmo MER evita i deadlock in quanto unisce i carichi di due percorsi solo se la procedura *TryToAssign* riesce nell'assegnamento di un nuovo insieme di percorsi ad un veicolo. Tuttavia, è comunque richiesta

un'assunzione più debole: la flotta S_v di veicoli, eventuamente completata dagli $n - m$ veicoli fittizi, deve poter gestire il daisy iniziale. Ad esempio, non esiste una soluzione in presenza di due veicoli aventi entrambi la massima capacità Q_1, due veicoli aventi la seconda massima capacità Q_2 e tre clienti con una richiesta $Q_2 < q_i \leq Q_1$, dove q_i indica la richiesta dell'i-esimo cliente e $i = 1, 2, 3$.

4.5.6 ***Costo computazionale di MER***

La lista H di archi è organizzata come un *heap* (mucchio, ossia *una grande quantità di qualcosa*). Nell'allocazione di memoria basata su heap, la memoria è allocata da un grande blocco di memoria inutilizzata. La dimensione della memoria da allocare può essere determinata a runtime e la durata di vita dell'allocazione stessa non dipende dalla procedura o dallo stack frame correnti. Si accede per via indiretta alla regione di memoria allocata, in genere attraverso un riferimento. La procedura standard *HeapExtract* permette di estrarre in $O(log\ k)$ l'elemento più piccolo di un insieme di cardinalità k, preservando la struttura dell'heap. L'altro operatore di heap usato in questo algoritmo è *HeapInsert*, che effettua l'inserimento in un heap in $O(log\ k)$. Nel caso di MER, gli elementi in H consistono di $k = n\ (n-1)/2$ archi non orientati tra due clienti: così, ogni operazione di heap costa $O(log\ n)$.

La soluzione *daisy* iniziale è costruita in $O(n)$. L'heap può essere costruito in $O(n2\ log\ n)$ in iterazioni successive oppure, unendo ricorsivamente i *sub-heaps*, in $O(n2)$. Il loop principale scansiona $O(n2)$ archi; per ogni arco, l'estrazione da H impiega $O(log\ n)$. Le altre operazioni per gli archi sono effettuate solo per merges completati con successo; si noti che il numero massimo di merges è $n - 1$. Il costo di una chiamata alla procedura *TryToAssign* è di $O(n\ log\ n)$, mentre una chiamata alla procedura *Assign* costa $O(n)$. Ne emerge che l'euristico descritto ha un costo computazionale di $O(n2\ log\ n)$.

5. Implementazione di un euristico per il MTVRP

In questo capitolo sarà proposta l'implementazione di un euristico per il Multi-Trip VRP (MTVRP), basato sul modello di Prins (2002), illustrato nel paragrafo 4.5. Nei paragrafi seguenti verrà presentato l'algoritmo; saranno analizzate le scelte implementative e verranno descritte nel dettaglio due procedure di Local Search già citate nel paragrafo 4.2: 2-opt e 3-opt. Tali metodi sono stati implementati all'interno dell'algoritmo al fine di migliorare la soluzione ottenuta. Il costo della soluzione è inteso in termini di distanza percorsa. L'algoritmo è stato sviluppato in linguaggio C su Visual Studio 2010 64bit. Sono state realizzate tre diverse implementazioni: una versione basilare, una con l'ottimizzazione parziale mediante l'aggiunta della procedura 2-opt ed una versione finale che prevede l'utilizzo sia di 2-opt sia di 3-opt.

5.1 Concetti fondamentali

In questo paragrafo sono esposti i concetti base che saranno utili al lettore per una più facile comprensione dell'algoritmo.

5.1.1 *Distanza*

Sulla base delle coordinate geografiche del deposito e dei clienti, l'algoritmo crea una matrice D di dimensione $(n+1) \times (n+1)$ contenente le distanze tra ogni possibile coppia di clienti e le distanze tra il deposito e ciascun cliente. Il valore in corrispondenza di D_{ij} rappresenta la distanza euclidea tra i clienti i e j, ed è calcolata nel seguente modo:

$$D_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (5.1)$$

5.1.2 *Saving Massimo*

Il saving sij di un arco (i,j) è il guadagno ottenuto nel caso in cui i clienti i e j vengano considerati all'interno dello stesso percorso, invece che in due percorsi separati. Esso è così calcolato:

$$s_{ij} = d_{1i} - d_{ij} + d_{j1} \quad (5.2)$$

Il merge di due percorsi è effettuato sulla base dei clienti i e j, ancora in attesa di essere serviti, a cui corrisponde il saving massimo.

5.1.3 ***Creazione ed estensione di Routes***

La costruzione di un nuovo percorso ha inizio con la ricerca della coppia (i, j) di clienti che corrisponde al valore massimo nella matrice S dei savings. Siano u e v i percorsi a cui appartengono, rispettivamente, i clienti i e j. Essi possono essere concatenati, attraverso i e j, se sono soddisfatti i vincoli

$$i \text{ e } j \text{ sono entrambi } \textit{non-serviti} \quad (5.3)$$
$$L_u + L_v \leq cv \quad (5.4)$$
$$d_u + d_v - S_{ij} \leq wt \quad (5.5)$$

ossia:

- la somma delle quantità di prodotto L_u ed L_v trasportate dai percorsi u e v non deve superare la capacità cv dei veicoli (5.4);
- la somma delle distanze percorse dai viaggi u e v, meno il risparmio S_{ij} ottenuto dall'unione attraverso i clienti i e j, non deve superare il massimo working-time wt dei veicoli (5.5).

Se il merge è effettuato con successo, i clienti i e j vengono dichiarati *serviti* e la matrice R dei Routes viene aggiornata, ossia, in corrispondenza della u-esima riga, viene aggiunta la sequenza di clienti presenti nella v-esima riga.

L'algoritmo procede aggiungendo clienti al percorso u, fino a che non esistono più percorsi v in grado di soddisfare i vincoli (5.3), (5.4) e (5.5). Più precisamente, si cerca il saving massimo a partire dai nodi estremi del percorso verso tutti gli altri nodi *non-serviti*. Se sono soddisfatti i vincoli (5.4) e (5.5), viene aggiunto alla route il cliente corrispondente a tale saving massimo. Se tutti i clienti sono *serviti*, la fase di creazione ed estensione di percorsi termina. Se non esistono clienti in grado di soddisfare i vincoli (5.4) e (5.5), il percorso che si sta considerando non può più essere esteso: si procede, in tal caso, con la creazione di un nuovo percorso. Altrimenti, si cerca nuovamente il massimo saving a partire dai nodi estremi del percorso verso tutti gli altri nodi *non-serviti*, per aggiungere un nuovo cliente alla route in questione.

Il meccanismo di estensione di un percorso appena illustrato può essere descritto come un loop che opera nel seguente modo:

1) Siano s e t gli identificativi, rispettivamente, del primo e dell'ultimo cliente attualmente serviti dal percorso che si sta considerando; viene preso il valore massimo nella matrice S dei savings, in corrispondenza delle righe s e t; sia $u = s$ o $u = t$, a seconda della riga a cui appartiene il valore massimo appena trovato S_{si} o S_{tj}, e sia $v = i$ o $v = j$, a seconda della colonna a cui appartiene il valore massimo appena trovato;
2) Se sono soddisfatti i vincoli (5.4) e (5.5), i percorsi u e v sono dichiarati *serviti* e la matrice R dei Routes viene aggiornata;
3) Se tutti i clienti sono *serviti*, la fase di estensione dei percorsi termina;
4) Se non esistono clienti in grado di soddisfare i vincoli (5.4) e (5.5), il percorso che si sta considerando non può essere esteso; si procede quindi con la ricerca di un nuovo percorso, come illustrato nel precedente paragrafo;
5) Si torna quindi al punto 1 per aggiungere altri clienti al percorso corrente.

5.1.4 ***Assegnamento Trip-to-Trucks***

La parte finale dell'algoritmo prevede l'assegnamento *Trip-To-Truck* dei percorsi ottenuti ai vincoli a disposizione, con l'obiettivo di minimizzare il numero di veicoli utilizzati.

Sia nt il numero di viaggi risultanti dalle precedenti fasi di creazione ed estensione degli itinerari. Operando in modalità Single-Trip, l'algoritmo terminerebbe qui: il numero V di veicoli da utilizzare sarebbe infatti uguale al numero nt di percorsi, in quanto tale modalità prevede che ciascun veicolo possa effettuare al massimo un percorso.

L'algoritmo è stato pensato, invece, per risolvere istanze di Multi-Trip VRP, ossia prevede la possibilità, per ciascun veicolo, di effettuare più percorsi, purchè non si superi il massimo working time wt dei veicoli. L'adattamento di questo euristico al caso Multi-Trip è ottenuto con una implementazione della procedura *PackTrips*. Definiamo *multitrip* l'insieme degli itinerari assegnati ad un veicolo. PackTrips opera nel seguente modo:

1) Sia mt il numero di multitrips; si ha $mt = nt$;

2) Se tutti i multitrips sono *completi*, la procedura termina; sia p il multitrip ancora *non-completo* che percorre la distanza maggiore;

3) Se esiste un multitrip j ancora *non-completo* tale che:

$$d_p \leq wt - d_j$$

Allora il multitrip j è concatenato al multitrip p e j viene dichiarato *servito*; altrimenti, p viene dichiarato *completo*; si torna al punto 2.

5.2 La procedura 2-Opt

2-opt è un metodo di Local Search ben noto in letteratura, il quale fu proposto da Croes (1958). Gli algoritmi di Local Search hanno bisogno di una soluzione iniziale da cui partire, un meccanismo che consente di modificare la soluzione attraverso delle *mosse*, un criterio di accettazione della soluzione e un criterio di arresto.

La soluzione iniziale può essere generata attraverso un *percorso generato a caso* oppure da una qualsiasi altra euristica. Il meccanismo che modifica la soluzione corrente è di tipo *edge-exchange*, ossia basato su uno scambio di archi, e il criterio di accettazione è di tipo *first-improvement*, ovvero, la prima mossa migliorativa viene subito effettuata, modificando la soluzione corrente. La condizione di arresto si ha dopo aver effettuato una visita completa all'interno della soluzione corrente, senza trovare miglioramenti. Come già detto nel paragrafo 4.3, il minimo locale individuato nella Local Search può essere anche molto lontano dalla soluzione ottima, in quanto dipende fortemente dalla soluzione di partenza. Il funzionamento della procedura 2-opt può essere descritto nel seguente modo:

- Step 1. Sia p il percorso a cui è applicata tale procedura;

- Step 2: Se esistono due archi (i, j) e (k, l) non consecutivi di p, ossia tali che $i \neq k, l,\ j \neq k, l,$ dove $j = i + 1,\ l = k + 1,$ che soddisfano il seguente vincolo:

$$D_{ik} + D_{jl} < D_{ij} + D_{kl}$$

Allora gli archi (i, j), (k, l) vengono sostituiti con gli archi (i, k), (j, l).

Torna allo Step 1.

- Step 3. Il percorso p è 2-ottimale; la procedura termina.

Una conseguenza particolare di un'applicazione della procedura 2-opt si può avere quando essa porta all'eliminazione del primo o dell'ultimo arco. In questo caso, si avrà un cambiamento del valore dei nodi estremi s e t del percorso; dovendo poi valutare la possibilità di inserimento di un nuovo cliente, occorrerà cercare il massimo saving a partire dai nodi estremi appena modificati. Ne concludiamo che un euristico che effettua una procedura di questo tipo può creare non solo itinerari più convenienti di quelli che si otterrebbero normalmente, ma anche una soluzione diversa in termini di clienti serviti da un dato percorso. Tuttavia, notiamo che nelle iterazioni successive a una procedura 2-opt con la modifica di almeno uno dei nodi estremi del percorso non si avrà necessariamente una soluzione migliore di quella che si otterrebbe senza di essa, a causa dell'inserimento nel percorso di clienti diversi. Inoltre, come già detto nel paragrafo 4.2, i tempi computazionali degli algoritmi di local search sono generalmente maggiori di quelli degli algoritmi costruttivi. Per questo motivo, l'algoritmo sviluppato prevede l'esecuzione di 2-opt solo una volta ogni due iterazioni.

5.3 La procedura 3-Opt

3-opt è, come 2-opt, un algoritmo di Local Search di tipo *edge-exchange*. Il meccanismo di scambio considera, questa volta, tre archi e li combina in diversi modi. Come in 2-opt, il criterio di accettazione è di tipo *first-improvement*. La procedura termina dopo aver effettuato una visita completa delle soluzioni vicine senza trovare miglioramenti.

Il meccanismo di 3-opt può essere descritto nel seguente modo:

- Step 1. Sia p il percorso a cui è applicata tale procedura;
- Step 2. Siano (i,j), (k,l), (r,q) tre archi non consecutivi di p, ossia tali che $i \neq k,l,r,q$; $j \neq k,l,r,q$; $k \neq r$, q; $l \neq r,q$, dove j = i + 1, l = k + 1, q = r + 1; consideriamo tutti i modi in cui è possibile sostituire questi tre archi con altri tre:
  - (i,k), (j,r), (l,q);
  - (i,r), (l,j), (k,q);
  - (i,l), (r,j), (j,q);

- (i, l), (r, j), (k, q).

Siano:

- $C_1 = D_{ik} + D_{jr} + D_{lq}$;
- $C_2 = D_{ir} + D_{lj} + D_{kq}$;
- $C_3 = D_{il} + D_{rk} + D_{jq}$;
- $C_4 = D_{il} + D_{rj} + D_{kq}$.

i costi della terna di archi relativi, rispettivamente, alla prima, alla seconda, alla terza e alla quarta combinazione; sia $C_{min} = min\{C_1, C_2, C_3, C_4\}$ e sia $C_0 = D_{ij} + D_{kl} + D_{rq}$. Se $C_{min} < C_0$, gli archi (i, j), (k, l), (r, q) vengono sostituiti con la terna di archi di costo C_{min}. Torna allo Step 1.

- Step 3. Il percorso p è 3-ottimale; la procedura termina.

Come nel 2-opt, anche nel 3-opt, in caso di eliminazione del primo o dell'ultimo arco, si avrà un cambiamento del valore dei nodi estremi s e t del percorso; in tal caso si avrà, nelle successive iterazioni, un percorso diverso e non necessariamente migliore di quello che si otterrebbe senza applicare la 3-opt.

Nell'algoritmo sviluppato e illustrato in questo capitolo, la procedura 3-opt è applicata solo ai percorsi *completo*, ossia non più estendibili mediante l'inserimento di nuovi clienti. Operando in questo modo, le successive iterazioni non saranno influenzate, in quanto l'algoritmo procederà con la costruzione di un nuovo percorso e l'itinerario appena ottimizzato non sarà più modificato.

5.4 L'algoritmo

5.4.1 ***Fase 0: Inizializzazione***

Step 1: Costruzione della matrice delle distanze D_{ij} per ogni coppia (i, j) di clienti. Il valore di D_{ij} viene calcolato come indicato nella (5.1).

Step 2: Costruzione della matrice dei savings S_{ij} per ogni coppia (i, j) di clienti. Il valore di S_{ij} è calcolato come indicato nella (5.2).

Step 3: Creazione di una matrice R dei *Routes* (percorsi), di dimensione $(n \times n)$, in cui, in corrsipondenza della i-esima riga è memorizzata la sequenza E_i dei clienti serviti dall'i-esimo percorso. La soluzione iniziale

prevede l'esistenza di tanti percorsi quanti sono i clienti, in modo tale che l'i-esimo percorso serva l'i-esimo cliente.

5.4.2 *Fase 1: Generazione di Routes*

Step 4: Se $\nexists$ una nuova possibile route, vai al punto 10; altrimenti, crea una nuova route p;

Step 5: Se $\nexists$ un cliente per espandere la route p, vai al punto 9; altrimenti, espandi la route; $count := 0$;

Step 6: $count := count + 1$;

Step 7: Se $(count \ \% \ 2) = 0$, vai al punto 8; altrimenti, vai al punto 5;

Step 8: Esegui 2-opt sul percorso p;

Step 9: Esegui 3-opt sul percorso p;

5.4.3 *Fase 2: Trip-to-Truck*

Step 10: Minimizzazione del numero di veicoli da utilizzare mediante l'esecuzione della procedura *PackTrips*, il cui funzionamento è descritto nel paragrafo 5.1.4.

Step 11: Termina, restituendo la miglior soluzione trovata X^* e il relativo costo Z^*.

5.5 Risultati

Nella tabella 5.1 sono riportati i risultati dei test effettuati su 21 istanze proposte da Christofides et al. (1979). Per ogni istanza viene indicato il numero di clienti coinvolti (colonna *Clienti*), il wotking time dei veicoli (colonna *Working Time*), il valore ottimo (colonna *Optimal Value*), la soluzione trovata dall'algoritmo sopra descritto (colonna *Soluzione trovata*) e la distanza, espressa in percentuale, di tale soluzione dall'Optimal Value. Tale distanza è calcolata nel seguente modo:

$$(Soluzione\ trovata - Optimal\ Value) \times 100 \ \div Optimal\ Value$$

e fornisce un indice di quanto la soluzione trovata dall'algoritmo differisce da quella ottima. Per soluzione trovata intendiamo la soluzione restituita dalla versione finale dell'algoritmo che, come già detto, prevede un'esecuzione di 2-Opt ogni due iterazioni ed un'esecuzione di 3-Opt alla fine della costruzione di ogni route. In

tutte queste istanze, sono considerati veicoli identici, dotati di una capacità di carico pari a 200.

Istanza	*Clienti*	*Working Time*	*Optimal Value*	*Soluzione trovata*	*Distanza in %*
CMT-12-m1-t861	101	861	820	939	15
CMT-12-m1-t902	101	902	820	939	15
CMT-12-m2-t430	101	430	820	939	15
CMT-12-m2-t451	101	451	820	939	15
CMT-12-m3-t287	101	287	820	939	15
CMT-12-m3-t301	101	301	820	939	15
CMT-12-m4-t215	101	215	820	939	15
CMT-12-m4-t225	101	225	820	939	15
CMT-12-m5-t172	101	172	820	941	15
CMT-12-m5-t180	101	180	820	992	21
CMT-12-m6-t150	101	150	820	876	7
CMT-11-m1-t1094	121	1094	1034	1241	20
CMT-11-m1-t1146	121	1146	1034	1241	20
CMT-11-m2-t547	121	547	1034	1241	20
CMT-11-m2-t573	121	573	1034	1241	20
CMT-11-m3-t365	121	365	1034	1241	20
CMT-11-m3-t382	121	382	1034	1241	20
CMT-11-m4-t274	121	274	1034	1175	14
CMT-11-m4-t287	121	287	1034	1175	14
CMT-11-m5-t219	121	219	1034	1087	5
CMT-11-m5-t229	121	229	1034	1107	7

Tabella 5.1: risultati dei test sulle istanze di Christofides et al. (1979)

Bibliografia

Christofides, N., Mingozzi, A., Toth, P., a cura di John Wiley & Sons, *The vehicle routing problem*. In *Combinatorial Optimization*, Chichester, UK

Croes, G. A., *A method for solving traveling salesman problems*, 1958. Operations Research, **6** 791-812.

Dantzig, G. B. e Ramser, J. H., *The Truck Dispatching Problem*, 1959. Management Science, **6**(1) 80-91.

Fleischmann, B., *The vehicle routing problem with multiple use of vehicles*, 1990. Working paper, Fachbereich Wirst-chaftswissenschaften, Universität Hamburg.

Foster, B. A. e Ryan, D. M., *An integer programming approach to the vehicle scheduling problem*, 1976. Operational Research Quarterly, **27** 367-384.

Gendreau, M., Hertz, A., Laporte, G., *A tabu search heuristic for the vehicle routing problem*, 1994. Management Science, **40** 1276-1290.

Gillett, B. E. e Miller, L. R., *A heuristic algorithm for the vehicle dispatch problem*, 1974. Operations Research, **22**(2) 340-349.

Glover, F., *Tabu Search - Parti I*, 1989. ORSA Journal on Computing, **1**(3) 190-206.

Mingozzi, A., Roberti, R., Toth, P., *An Exact Algorithm for the Multitrip Vehicle Routing Problem*, 2013. Journal on Computing, **25**(2) 193-207.

Mole, R. H. e Jameson, S. R., *A sequential route-building algorithm employing a generalized saving criterion*, 1976. Operational Research Quarterly, **27**(2) 503-511.

Prins, C., *Efficient Heuristic for the Heterogeneous Fleet Multitrip VRP with Application to a Large-Scale Real Case*, 2002. Journal of Mathematical Modelling and Algorithms, **1** 135-150.

Renaud, J., Boctor, F. F., Laporte, G., *An improved petal heuristic for the vehicle routeing problem*, 1996. Journal of Operational Research Society, **47** 329-336.

Rochat, Y. e Taillard, E. D., *Probabilistic intensification and diversification in local search for vehicle routing*, 1995. Heuristics, **1**(1) 147-167.

Rochat, Y. e Taillard, É. D., *Probabilistic intensification and diversification in local search for vehicle routing*, 1995. Heuristics, **1**(1) 147-167.

Ryan, D. M., Hjorring, C., Glover, F., *Extension of the petal method for vehicle routeing*, 1993. Journal of Operational Research Society, **44** 289-296.

Taillard, É. D., Laporte, G., Gendreau, M., *Vehicle Routeing whit Multiple Use of Vehicles,* 1996. Journal of the Operational Research Society, **47**(8) 1065-1070.

Taillard, É., *Parallel iterative search methods for vehicle routing problems*, 1993. Networks, **23** 661-676.

Ringraziamenti

A Uno: prego.

A Maria Chiara, per essere stata sempre presente; per il sostegno nei momenti difficili, per i bei momenti trascorsi insieme. Per aver fatto sempre il massimo, per avermi compreso, aiutato e incoraggiato. Per essere stata indispensabile.

A Mec, la persona più generosa mai conosciuta, l'amico ideale, il coinquilino perfetto; per la compagnia, l'infinita sopportazione, il sostegno morale e la disponibilità immediata, in ogni momento e in ogni caso.

Al Dott. Tusco Reggi, compagno di studio e di battute tristissime, per la sintonia nel conciliare momenti di serietà e studio intenso con momenti di totale cazzeggio. Per avermi insegnato tanto, per i consigli su ogni cosa, per il sostegno nello studio e fuori.

Ad Alfredo Potassio, poeta, inventore e libero pensatore, per le mille avventure vissute insieme, tra arte, poesia, radio, musica e sedie.

A Silvio e Oliver, per essere stupidi almeno quanto me; per il divertimento assicurato anche nelle serate apparentemente monotone e le intere giornate trascorse in facoltà ad alternare studio e demenza.

Ad Amilcare, Elvis, Arduino e Mare, per aver reso piacevole anche lo studio più pesante, per l'aiuto reciproco e le tante risate.

Ai miei genitori, per aver reso possibile tutto questo. Per aver creduto in me anche quando io stesso avevo smesso di farlo; per essere stati pazienti; per essermi stati vicini e lontani contemporaneamente; per aver sempre provato (riuscendoci) a capirmi; per avermi dato tutto, aiutandomi a diventare quello che sono.