UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di laurea in Informatica

Algoritmi efficienti per il Multiple Stacks Double Travelling Salesman Problem

RELATORE
Dott. Alberto Ceselli

TESI DI LAUREA DI
Marco Casazza
Matr. 689878

Anno accademico 2008/2009

Indice

CAPITOLO 1

Introduzione

Le tecniche di ottimizzazione sono sempre più integrate all'interno dei processi produttivi e di distribuzione. Spesso intervenire sulla logistica di un'azienda può garantire un risparmio decisivo. L'introduzione in contesti reali di sistemi evoluti di supporto alle decisioni è dovuta sia alla sempre maggiore diffusione dei sistemi informatici, sia allo sviluppo di nuovi algoritmi in grado di risolvere più velocemente e in modo migliore determinate categorie di problemi.

1.1 Routing e packing

Spesso un'azienda si trova ad affrontare dei problemi di *routing* o *instradamento*, ovvero la scelta del percorso migliore all'interno di una rete telematica o logistica. Nella vita quotidiana applichiamo spesso inconsapevolmente algoritmi che ci portano alla soluzione di questo tipo di problemi ed il mercato offre dispositivi che risolvono per noi quelli più difficili. Si pensi all'ausilio che un semplice navigatore GPS può fornire ad un corriere che deve trovare la strada migliore per effettuare delle consegne all'interno di

una città. Nel settore distribuzione di una azienda questo tipo di problemi non può essere trascurato, poiché i relativi costi incidono sulla competitività complessiva di un prodotto.

Un'altra categoria di problemi che spesso sorgono in logistica è quella che riguarda il *packing* o *inscatolamento*, ovvero la scelta della disposizione ottima di oggetti in un contenitore di dimensioni limitate, come ad esempio degli scatoloni in un furgone. Nonostante sembri facile risolvere questo tipo di problemi, nella realtà non è così. Si consideri la spedizione di merci: se queste non sono disposte nel modo corretto, occupando quindi più spazio del necessario, potrebbero essere necessarie più spedizioni, con un costo maggiore per l'azienda.

Attualmente disponiamo di buoni algoritmi che risolvono molti dei problemi sopracitati quando questi sono presi come attività indipendenti. Spesso però le due categorie di problemi sono strettamente integrate e devono essere risolte simultaneamente.

1.2 Double Travelling Salesman Problem with Multiple Stacks

Uno dei più noti tra i problemi di routing è il TSP[17], ovvero il problema di un commesso che deve visitare dei punti di interesse e tornare alla partenza percorrendo il cammino meno costoso, in termini di tempo, distanza o altro ancora. Più formalmente è richiesto di trovare il ciclo Hamiltoniano[2] minimo della rete presa in considerazione. Il TSP è un problema che si deve spesso affrontare in logistica: si pensi ad esempio alla distribuzione di merci all'interno di una rete, con un veicolo che deve visitare tutti i clienti e ritornare infine al deposito.

Nonostante il TSP sia un problema già di per sé difficile, esistono delle variazioni che rendono ancora più ardua la sua risoluzione.

Il *Double Travelling Salesman Problem with Multiple Stacks* (*DTSPMS*) è un problema reale di raccolta e distribuzione, in cui il problema del TSP è reso ancora più difficile dall'integrazione con un problema di packing.

Il problema del DTSPMS richiede che un singolo veicolo prelevi della merce da alcuni mittenti residenti in una città e la consegni successivamente ai rispettivi destinatari residenti in una seconda città. Dato che le città sono distanti è necessario che la raccolta e la consegna siano effettuate separatamente.

Le merci vengono impacchettate e organizzate all'interno del veicolo su K pile distinte. La loro disposizione è anche chiamata piano di carico. Le pile funzionano in modo LIFO: ogni mittente inserisce la propria merce in cima ad una pila e le consegne devono essere effettuate a destinatari la cui merce è in cima al piano di carico. Ad un trasportatore non è mai possibile riorganizzare il piano di carico poiché questo comporta una maggiore durata della consegna e possibili danneggiamenti della merce.

L'obbiettivo del problema è di minimizzare la distanza percorsa dal veicolo per effettuare tutte le consegne. Viene richiesto quindi di calcolare un TSP su ogni città, ma la particolare regola di costruzione del piano di carico lega le visite tra loro, rendendo il problema molto più difficile da trattare.

1.3 Obbiettivi

L'obbiettivo della tesi è di progettare e realizzare un algoritmo euristico che risolva il problema del DTSPMS in modo efficiente, fornendo comunque buone soluzioni. Infatti, si può dimostrare che alcuni sottoproblemi possono essere risolti in tempo polinomiale ed i relativi metodi possono essere combinati in un unico algoritmo per il DTSPMS.

Il lavoro è svolto in diverse fasi:

rassegna della letteratura: in cui sono stati presi in considerazione diversi metodi di risoluzione per questo ed altri problemi;

analisi proprietà teoriche: in cui sono state prese in esame le caratteristiche del problema;

progettazione del software: in cui sono state progettate le strutture dati e i moduli dell'algoritmo;

implementazione: in cui si è scritto il software vero e proprio integrandolo con librerie esterne;

analisi sperimentale: in cui si sono eseguiti vari test per capire meglio come si comporta l'algoritmo, i suoi punti forti e quelli deboli.

1.4 Organizzazione della tesi

In questo documento sono analizzate le proprietà, le caratteristiche e il lavoro svolto per realizzare un software per la risoluzione del problema del DTSPMS.

Nel capitolo 2 è proposta una rassegna di ciò che è stato realizzato fino ad ora su questo problema, e su altri problemi simili, con alcune considerazioni sui risultati ottenuti.

Nel capitolo 3 è data una spiegazione formale del problema, con l'aggiunta di un modello di programmazione matematica che lo descrive.

Il capitolo 4 è dedicato alle proprietà del problema che ci hanno permesso di realizzare il nostro algoritmo.

Nel capitolo 5 sono descritte le tecniche utilizzate per la realizzazione del risolutore e vengono mostrati gli algoritmi implementati. Verranno inoltre esposte alcune problematiche che possono presentarsi in situazioni reali.

Infine il capitolo 6 è dedicato all'analisi sperimentale effettuata su alcune istanze di test.

CAPITOLO 2

Rassegna della letteratura

Esistono diversi problemi in cui routing e packing sono integrati: in letteratura sono stati studiati metodi per risolvere problemi in cui gli item possono avere forma, volume e peso differenti. Questo approccio è molto simile ai casi reali che le aziende devono affrontare, in quanto le merci non hanno quasi mai le stesse caratteristiche e spesso devono essere raggruppate (in base alla destinazione o alla tipologia) prima di essere disposte sul mezzo di trasporto.

2.1 Routing e packing

Di seguito saranno brevemente descritti alcuni problemi simili a quello da noi studiato. Ognuno di questi si distingue per particolari vincoli che rendono la sua trattazione più complicata.

2.1.1 Pickup e delivery

Nei problemi *Pickup and Delivery Problem* il loading plan è costruito durante la visita dei cutomer, in cui le merci vengono sia caricate che scaricate dal veicolo.

Uno dei problemi più interessanti e simili al DTSPMS è il *Single Vehicle Pickup and Delivery Problem with LIFO constraints*: in questo problema un singolo veicolo deve ritirare e consegnare della merce in un unico viaggio a tutti i clienti. Il vincolo imposto sulla disposizione degli item nel loading plan è che questo sia trattato come una struttura LIFO, in cui l'ultima merce ritirata è anche la prima ad essere consegnata.

Per la risoluzione di questo particolare problema sono stati sperimentati sia algoritmi esatti basati su branch-and-bound sia algoritmi euristici. Tra questi ultimi è da menzionare l'implementazione di un algoritmo *Variable Neighbourhood Search* (*VNS*)[9] in cui vengono usati otto operatori differenti che, applicati iterativamente ad una soluzione di partenza, migliorano gradualmente i risultati. L'algoritmo sfrutta anche un meccanismo di restart riuscendo a raggiungere buoni risultati.

Una recente implementazione di questo algoritmo[4] risolve istanze fino a 750 customer in circa trenta minuti.

2.1.2 Costruzione del loading plan

Tra i problemi in cui routing e packing sono strettamente legati vi sono i *2-dimensional Loading Capacitated Vehicle Routing Problem* (*2L-CVRP*). In questi problemi il loading plan è visto come un piano bidimensionale sul quale devono essere disposte merci di dimensioni diverse. L'obbiettivo è quello di ottenere il loading plan migliore associato ad un tour di consegna. Esistono diverse versioni di questo problema, distinte dalla possibilità di ruotare o spostare le merci caricate. Per le consegne è possibile inoltre usufruire di più veicoli. Gli algoritmi euristici studiati e implementati finora usano due approcci di risoluzione differenti: l'*Ant Colony Optimization* (*ACO*)[7] e il *Guided Tabu Search* (*GTS*)[20].

L'ACO[5] è una metaeuristica in cui una popolazione coopera per risolvere un problema difficile effettuando delle scelte in modo probabilistico. Per risolvere il problema 2L-CVRP si è partiti dall'algoritmo *Saving-Based ACO*[7], studiato per risolvere i problemi *Capacitated Vehicle Routing Problem*. La strategia utilizzata è quella di assegnare inizialmente un veicolo

diverso ad ogni customer e successivamente unire i cammini, riducendo così il numero di veicoli e i costi. Questo metodo è stato adattato per risolvere il problema del 2L-CVRP, per il quale vi sono vincoli aggiuntivi. Ad ogni iterazione l'algoritmo ricalcola un nuovo piano di carico, che aiuterà poi la scelta del cammino da intraprendere. Sono inoltre penalizzate le soluzioni che usufruiscono di un numero maggiore di veicoli.

Il GTS si basa sul metodo *Tabu Search* (*TS*)[10][8]. Il TS è una tecnica metaeuristica di ricerca locale che permette di esplorare nuove soluzioni anche quando si è giunti ad un ottimo locale. L'algoritmo inizia generando una soluzione di partenza che, iterativamente, sarà modificata applicando degli operatori. La particolarità di questa tecnica è che permette di effettuare delle mosse peggioranti e mantiene memoria delle soluzioni già visitate, così da evitarle nelle successive esplorazioni.

Dai test riportati in letteratura, l'ACO risulta essere mediamente migliore del GTS, sia per istanze di piccole dimensioni che per istanze particolarmente difficili. Sono state inoltre considerate istanze di problemi reali di 2L-CVRP, per osservare come si potrebbe comportare un sistema di pianificazione completamente automatizzato. In questi casi l'ACO riesce a risolvere istanze di 76 customer in poco più di un minuto.

2.1.3 Finestre temporali

Oltre che al metodo costruttivo del loading plan, i vincoli possono essere applicati anche ad altre variabili del nostro problema: ad esempio vi sono molti casi reali in cui una merce deve essere recapitata entro un orario prestabilito e quindi il tour di consegna è fortemente influenzato da questa limitazione. Questo è il caso dei *Vehicle Routing with Time Windows and Loading Problem*, problemi in cui conta sia la disposizione delle merci, sia il tempo di consegna al cliente. I mezzi di trasporto hanno un vano di carico limitato nel peso, nel volume e accessibile in modo LIFO, imponendo così che il tour di visita sia l'inverso dell'orine in cui è stato costruito il loading plan. Inoltre ogni cliente può essere visitato solamente da un corriere. L'obbiettivo è quello di minimizzare il numero di mezzi usati e il tragitto percorso.

Un primo approccio proposto in letteratura, chiamato metodo sequenziale (*sequential method*)[14], prevede di risolvere il problema trascurando i vincoli sulla disposizione del carico e permettendo visite multiple per ogni customer. Ad ogni passo viene inserito un nuovo item all'interno del loading plan e se questo è pieno viene utilizzato un nuovo mezzo di trasporto. Inserendo gli item senza rispettare i suddetti vincoli, costringe l'algoritmo a delle scelte: potrebbe essere necessario effettuare degli spostamenti nel piano di carico oppure rivisitare più volte lo stesso customer per poter effettuare le consegne. L'algoritmo deve quindi scegliere tra aumentare il costo del tragitto oppure aumentare il tempo di consegna.

Il metodo gerarchico (*hierarchical method*)[14] invece, non trascura nessun vincolo del problema ma considera il packing come un sottoproblema. L'algoritmo si occupa prima di generare un percorso che visiti tutti i customer e successivamente di generare i loading plan dei vari mezzi.

Per i test dei due approcci sono stati creati appositi dataset di 25 customer. Il metodo gerarchico risulta migliore nei casi generali, mentre quello sequenziale è il più quando la costruzione dei loading plan ha un grosso impatto sulla soluzione finale. Tutti i risultati dei test sono stati ottenuti eseguendo gli algoritmi per meno di un minuto.

2.2 Algoritmi per il DTSPMS

Nonostante il DTSPMS sia un problema studiato solamente di recente, in letteratura esistono già algoritmi per la sua risoluzione, sia esatti che euristici.

2.2.1 Algoritmi esatti

Il metodo meno efficiente per ottenere delle soluzioni ottime al nostro problema, è sicuramente quello di enumerare (implicitamente o esplicitamente) tutte le soluzioni e scegliere infine la migliore. Questo comporta il calcolo di tutte le permutazioni dei due tour e la successiva creazione dei loading plan o, viceversa, la creazione di tutti i possibili loading plan e successivamente i tour ottimi. Dato che un simile approccio non è assolutamente applicabile

a problemi reali, si è cercato di sviluppare algoritmi più efficienti sfruttando differenti tecniche di progettazione.

I primi algoritmi esatti hanno usato un approccio *branch-and-cut*[11] per risolvere il problema formulato in modi diversi.

Un primo modello di risoluzione, chiamato delle precedenze[16], consiste nel rimuovere i vincoli di eliminazione dei sottotour dal modello originale del problema. Ad ogni soluzione trovata vengono ricercati delle violazioni attraverso il calcolo del taglio minimo del grafo: se vi è il passaggio di una sola unità di flusso da una regione del grafo ad un'altra, o non vi è affatto, allora siamo in presenza di sottocicli, che devono essere eliminati inserendo appositi vincoli all'interno del modello.

Un altro metodo consiste nel modellare il problema come un problema di flusso[16]. Anche in questo caso sono i vincoli sui sottotour ad essere rilassati e, trovata una soluzione, vengono ricercate le violazioni in questo modo: partendo da un qualsiasi nodo e seguendo il cammino della soluzione, si dovranno incontrare tutti gli elementi che fanno parte della sua stessa pila in maniera ordinata. Se questo non accade, significa che vi sono dei sottocicli da eliminare. Questo viene fatto aggiungendo vincoli al modello.

Un ultimo approccio è quello di trascurare i vincoli di precedenza e, una volta generati i tour ottimi, inserire dei tagli dove le precedenze non sono rispettate. Questo viene chiamato *TSP with Infeasible Path model* (*TSPIP*)[16].

Tra quelli presentati, il TSPIP è il modello che fornisce i migliori risultati. I primi due approcci invece si equivalgono per istanze di grosse dimensioni, mentre per quelle piccole e medie è il modello a precedenze che fornisce soluzioni di costo inferiore.

Recentemente è stato presentato un nuovo algoritmo esatto per la risoluzione del DTSPMS[13] che usa un approccio totalmente differente da quelli finora presentati: l'algoritmo ricerca i K-migliori tour di pickup e delivery e calcola un loading plan per ogni combinazione fino a che se ne trova uno valido. L'algoritmo garantisce l'ottimalità partendo dalle coppie di tour che hanno il minor costo totale e andando ordinatamente verso le coppie più costose. Purtroppo non è possibile effettuare un confronto tra questo e i pre-

cedenti algoritmi poiché i test effettuati si limitano ad istanze molto piccole, differenti da quelle generalmente usate in letteratura.

2.2.2 Algoritmi euristici

Gli algoritmi euristici sviluppati finora si basano sul paradigma della ricerca locale, ovvero partono da una soluzione ammissibile e iterativamente cercano di migliorarla. Molti hanno inoltre un meccanismo di restart, che permette di rieseguire l'algoritmo considerando soluzioni iniziali differenti.

Prima di introdurre alcuni algoritmi euristici sviluppati, è necessario analizzare alcuni operatori applicabili alle soluzioni del problema, ovvero i tour e il loading plan. Scegliendo differenti elementi si ottengono differenti risultati. L'insieme di tutte le possibili soluzioni ottenibili dall'applicazione di un operatore sulle strutture dati si dice intorno. Di seguito saranno descritti degli operatori già proposti in letteratura per alcuni algoritmi di ricerca locale:

Route Swap (RS): data una soluzione ammissibile, questo operatore inverte l'ordine di visita di due customer in un tour. Se gli item corrispondenti risiedono nella stessa pila allora dovrà essere effettuata l'inversione sia nel piano di carico sia nel secondo tour della soluzione per mantenere l'ammissibilità;

Complete Swap (CS): data una soluzione ammissibile, l'operatore effettua lo scambio di due elementi del loading plan e, per mantenere la compatibilità, effettua degli spostamenti nei tour;

In-stack Swap (IS): questo operatore è simile al CS ma effettua lo scambio solamente tra elementi dello stesso stack;

Reinsertion (R): data una soluzione ammissibile, l'operatore cambia la posizione di visita di un customer sia nel tour di pickup che in quello di delivery, reinserendolo opportunamente in uno stack differente. Per mantenere l'ammissibilità della soluzione le nuove posizioni devono soddisfare le condizioni di precedenza;

r-Route Permutation (r-RP): questo operatore fornisce soluzioni ottenute permutando degli elementi consecutivi di un tour. Viene eseguito solamente su elementi appartenenti a stack diversi, in questo modo viene mantenuta l'ammissibilità;

r-Stack Permutation (r-SP): questo operatore fornisce soluzioni ottenute permutando degli elementi caricati consecutivamente in uno stack della soluzione. Per mantenere questa ammissibile è necessario spostare i customer, associati agli item permutati, all'interno dei tour.

Prima di effettuare operazioni di ricerca locale, è necessario ottenere una soluzione ammissibile di partenza, ad esempio utilizzando lo stesso algoritmo con parametri diversi o usando del codice apposito.

I primi due algoritmi che fanno uso di alcuni degli operatori descritti precedentemente sono il *Tabu Search* (*(*)TS) e il *Simulated Annealing* (*SA*)[15]. Il TS esegue iterativamente gli operatori RS e CS seguendo un pattern predefinito, mentre il SA, ad ogni iterazione, sceglie in modo probabilistico quale dei due operatori applicare. Le implementazioni attuali possono fornire buoni risultati solamente se lasciate in esecuzione per qualche minuto. Mediamente il SA fornisce soluzioni peggiori del 10% rispetto alle migliori soluzioni fornite in letteratura. Ancora più distante è il TS, con risultati peggiori del 20%.

È possibile risolvere il problema anche con algoritmi di *Iterated Local Search* (*ILS*)[15]: partendo da una soluzione ammissibile si ricerca un ottimo locale del problema usando determinate procedure (ad esempio *Steepest Descent*). Raggiunto l'obbiettivo si riparte da un'altra soluzione di base per cercare un altro ottimo locale. In questi algoritmi è molto importante la scelta dei punti di partenza, che possono portare a differenti ottimi locali. Le implementazioni attualmente disponibili forniscono risultati che mediamente sono oltre il 50% peggiori dei migliori risultati con cui si effettua il confronto.

Sono stati realizzati inoltre algoritmi *Large scale Neighbourhood Search*(*LNS*)[15], i quali tolgono un insieme di elementi da una soluzione e li reintroducono successivamente secondo una determinata politica. Le strategie di rimozione e di inserimento determinano la bontà della soluzione ottenuta. I risultati forniti in letteratura per alcune implementazioni sono molto vicini

alle soluzioni di confronto: in soli 10s l'algoritmo riesce a fornire soluzioni peggiori solamente del 4% rispetto ai migliori risultati forniti in letteratura.

Simili all'ILS vi sono gli algoritmi chiamati *Variable Neighborhood Search* (*VNS*) che, partendo da una soluzione di base, arrivano ad un ottimo locale applicando un determinato operatore sulle strutture dati. Quest'ultimo viene successivamente cambiato e l'algoritmo riprende a iterare sulle soluzioni.

Una variante di questa tipologia di algoritmi specifica per il problema del DTSPMS è l'*Hybridized VNS* (*HVNS*)[6]: si parte da un set di soluzioni ammissibili e per ognuna di esse si cerca un ottimo locale utilizzando uno specifico operatore. Se i risultati raggiunti non migliorano le soluzioni già trovate, l'algoritmo riparte modificando quest'ultime e usandole come base. Mediamente l'HVNS è molto vicino ai migliori risultati forniti in letteratura, soprattutto se viene lasciato in esecuzione per qualche minuto.

CAPITOLO 3

Modello formale

È possibile formalizzare il DTSPMS come un problema di ottimizzazione su grafi: siano le città due grafi non orientati completi $G^P(V^P, E^P)$ e $G^D(V^D, E^D)$ chiamati rispettivamente *grafo di pickup* e *grafo di delivery*. Siano i clienti (o *customer*) i nodi dei due grafi, rappresentati come interi all'interno degli insiemi $V^P = \{0 \dots N\}$ e $V^D = \{0 \dots N\}$ in cui 0 è il nodo deposito. Siano $E^P = \{V^P \times V^P\}$ e $E^D = \{V^D \times V^D\}$ gli insiemi degli spigoli che collegano i nodi dei grafi, ovvero le strade che mettono in comunicazione i customer delle città. Siano inoltre c_{ij}^P e c_{ij}^D rispettivamente i costi associati ad ogni spigolo $(i, j) \in E^P$ e $(i, j) \in E^D$, ovvero i costi necessari per percorrere la strada che porta direttamente dal customer i al customer j.

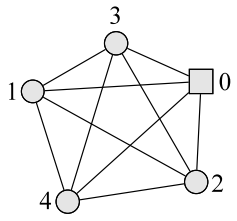


Figura 3.1: Esempio di città per $N = 4$

Sia $I = \{1 \dots N\}$ l'insieme delle merci (o *item*) ognuna rappresentata da un intero. Siano $V_L^P = V^P \setminus \{0\}$ e $V_L^D = V^D \setminus \{0\}$ gli insiemi dei customer delle due città che devono rispettivamente inviare e ricevere le suddette merci. Ad ogni intero dell'insieme dei mittenti V_L^P è associato un intero nell'insieme dei destinatari V_L^D al quale si deve inviare una merce in I rappresentata dal medesimo intero, ovvero ogni customer $i \in V_L^P$ deve inviare la merce $i \in I$ al destinatario $i \in V_L^D$.

Siano il *tour di pickup* e il *tour di delivery* rispettivamente dei cicli Hamiltoniani[2] sul grafo di pickup e sul grafo di delivery, definiti come $P \subseteq E^P$ e $D \subseteq E^D$. Entrambi i tour partono e terminano al deposito. Dati due customer i e j appartenenti ad un tour T, si dice che i precede j nel tour T ($i \prec_T j$) se, partendo dal deposito 0, lo incontriamo prima di j.

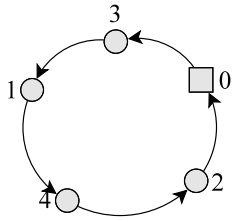


Figura 3.2: Esempio di tour con $N = 4$

Sia L il piano di carico (o *loading plan*) formato da K pile (o *stack*). Ogni stack è una sequenza ordinata che definisce l'ordine in cui sono stati inseriti gli item. Si denota con $Stack(k)$ il k-esimo stack del loading plan, con $k = 1 \dots K$. Si dice che $i \in Stack(k)$ se l'item i è stato inserito nel k-esimo stack. Dati due item i e j appartenenti allo stesso stack k, quindi $i, j \in Stack(k)$, si dice che i precede j all'interno del loading plan L ($i \prec_L j$) quando j compare subito dopo i nello stack.

Una soluzione ammissibile S del problema (mostrata in Figura 3.4) è composta dai due tour P e D e dal loading plan L contenente gli N item disposti in modo che in ogni stack l'ordine di precedenza degli item corrisponda all'ordine di precedenza dei customer all'interno del tour di pickup e sia inverso all'ordine di precedenza dei customer all'interno del tour di delivery.

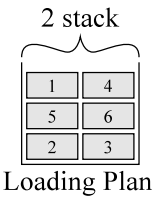


Figura 3.3: Esempio di loading plan con $N = 6$ e $K = 2$

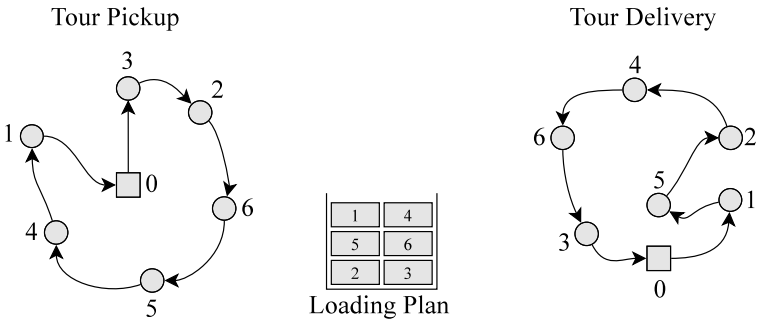


Figura 3.4: Esempio di soluzione ammissibile per $N = 6$ e $K = 2$

L'obbiettivo è quello di ottenere una soluzione ammissibile che minimizzi i costi totali, dati dai costi delle visite ovvero dalla sommatoria di tutti i costi c_{ij}^P e c_{ij}^D degli spigoli $(i, j) \in P$ e $(i, j) \in D$.

È facile dimostrare che il DTSPMS è un problema *NP-HARD*, dato che ha come sottoproblema la risoluzione di un TSP, che è dimostrato essere *NP-HARD*.

Si possono individuare due casi limite del problema:

- quando il loading plan è costruito con $K = 1$ (Figura 3.5);
- quando il loading plan è costruito con $K = N$ (Figura 3.6).

Nel primo caso i tour sono l'uno l'inverso dell'altro e l'ordine di visita dipende esclusivamente da come sono inseriti gli item all'interno dello stack.

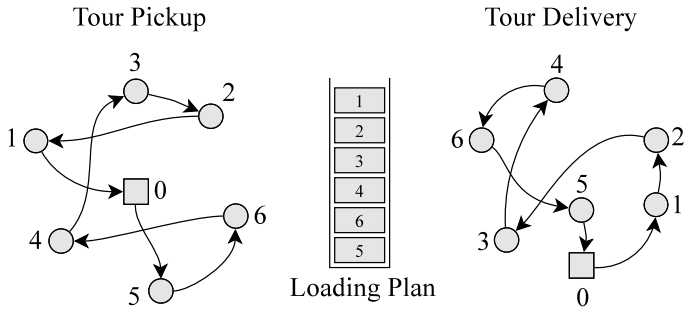


Figura 3.5: Esempio di soluzione con $N = 6$ e $K = 1$

Nel secondo caso invece i tour sono indipendenti, infatti non ci sono vincoli di precedenza da rispettare. Il problema è risolvibile come due TSP distinti su ogni grafo.

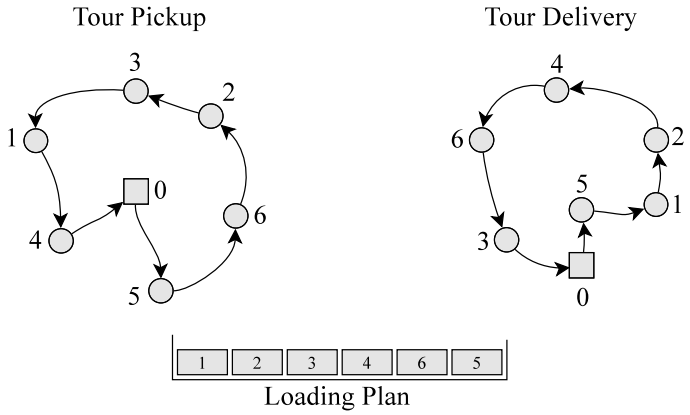


Figura 3.6: Esempio di soluzione con $N = 6$ e $K = N$

3.1 Modello di programmazione matematica

Il problema può essere descritto come problema di programmazione lineare intera[11] nel modo seguente:

Variabili:

$$x_{ij}^T = \begin{cases} 1, \text{ se } (i,j) \in T \\ 0, \text{ altrimenti} \end{cases} \quad ,\forall i,j \in V^T, \forall T \in \{P,D\}$$

$$y_{ij}^T = \begin{cases} 1, \text{ se } i \prec_T j \\ 0, \text{ altrimenti} \end{cases} \quad ,\forall i,j \in V_S^T, \forall T \in \{P,D\}$$

$$z_{ik} = \begin{cases} 1, \text{ se } i \in Stack(k) \\ 0, \text{ altrimenti} \end{cases} \quad ,\forall i \in N, \forall k = 1 \dots K$$

Tutte le variabili del problema sono binarie. Le variabili x indicano se uno spigolo del grafo appartiene alla soluzione. Le variabili y definiscono l'ordine in cui due nodi devono essere visitati nei tour. Le variabili z associano un elemento ad uno stack.

Un modello per il DTSPMS è il seguente:

$$\min \sum_{i,j \in V^P} c_{ij}^P \cdot x_{ij}^P + \sum_{i,j \in V^D} c_{ij}^D \cdot x_{ij}^D$$

Subject to:

$$\sum_{i \in V^T} x_{ij}^T = 1 \quad ,\forall j \in V^T, \forall G \in \{P,D\} \tag{3.1}$$

$$\sum_{j \in V^T} x_{ij}^T = 1 \quad ,\forall i \in V^T, \forall G \in \{P,D\} \tag{3.2}$$

$$y_{ij}^T + y_{ji}^T = 1 \quad ,\forall i,j \in V_S^T, i \neq j, \forall G \in \{P,D\} \tag{3.3}$$

$$y_{ik}^T + y_{kj}^T \leq y_{ij}^T + 1 \quad ,\forall i,j,k \in V_S^T, i \neq j, \forall G \in \{P,D\} \tag{3.4}$$

$$x_{ij}^T \leq y_{ij}^T \quad ,\forall i,j \in V_S^T, \forall G \in \{P,D\} \tag{3.5}$$

$$y_{ij}^P + z_{ik} + z_{jk} \leq 3 - y_{ij}^D \quad ,\forall i,j \in V_S^T, \forall k = 1 \dots K \tag{3.6}$$

$$\sum_{k=1}^{K} z_{ik} = 1 \quad ,\forall i \in I \tag{3.7}$$

$$\sum_{i \in I} z_{ik} \leq C \quad ,\forall k = 1 \dots K \tag{3.8}$$

$$x, y, z \in \mathbb{B} \tag{3.9}$$

L'obbiettivo del problema è quello di ottenere una soluzione con costo minimo, ovvero la somma dei costi degli archi appartenenti alla soluzione deve essere la più bassa possibile.

I vincoli 3.1 e 3.2 sono vincoli di conservazione di flusso, impongono che nella soluzione, per ogni nodo, siano presi un solo arco entrante ed un solo arco uscente.

Il vincolo 3.3 assicura che per ogni coppia di nodi distinti di un grafo deve essere stabilita una precedenza. Il vincolo 3.4 impone che se un nodo i deve precedere k e questo deve precedere j, allora anche i deve precedere j. Il vincolo 3.5 relaziona la varaibile x con la variabile y, imponendo che se un arco appartiene alla soluzione, allora deve esserci una precedenza tra i nodi interessati.

Il vincolo 3.6 rappresenta la regola LIFO di carico degli stack: se due elementi appartengono allo stesso stack non devono essere visitati nello stesso ordine.

Il vincolo 3.7 assicura che un elemento apparterrà ad un solo stack, mentre 3.8 è il vincolo di capacità degli stack, i quali possono contenere solo un numero limitato di elementi.

Il vincolo 3.9 è il vincolo di integrità delle variabili x, y e z.

CAPITOLO 4

Proprietà del problema

Di seguito verranno mostrate alcune proprietà sulle quali si basa il nostro algoritmo di risoluzione. Queste dimostrano che se è data una delle componenti di una soluzione (i tour oppure il loading plan), è possibile risalire alla componente mancante in tempo polinomiale.

4.1 Minimo numero di stack

Problema 1 *Dati il tour di pickup ed il tour di delivery, trovare il loading plan che utilizzi il numero minimo di stack.*

Proprietà 1 *Il problema 1 può essere risolto in tempo polinomiale.*

Dati i due grafi G^P e G^D, in cui i vertici dei customer sono etichettati con interi da 1 a N, è possibile definire il tour di pickup P e il tour di delivery D come permutazioni degli N interi (Figura 4.1). Data la *permutazione identità*[18] $I = (1, \ldots, N)$, è possibile definire $Map()$ come la relazione biunivoca che a ogni intero $i = 1 \ldots N$ di posizione k-esima in P associa l'elemento $j = 1 \ldots N$ di posizione k-esima in I. Come mostrato in Figura

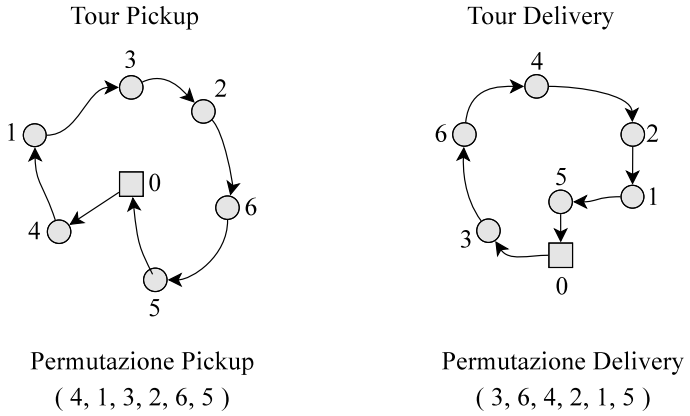


Figura 4.1: Esempio permutazioni con $N = 6$

4.2, è possibile sfruttare questa relazione per ottenere la permutazione M, in cui per ogni elemento $i = 1 \ldots N$ di posizione k-esima in D, vi è un elemento $j = Map(i)$ anch'esso di posizione k-esima.

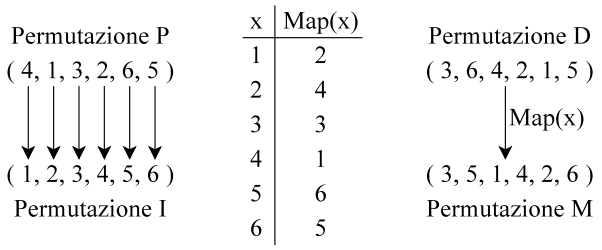


Figura 4.2: Relazioni tra permutazioni

Sia il *grafo dei conflitti* il grafo non orientato costituito da N vertici (etichettati con interi da 1 a N) che possiede uno spigolo per ogni coppia di vertici che non compaiono nello stesso ordine all'interno di I e della permutazione M rovesciata (Figura 4.3). Il problema 1 può essere risolto come la colorazione minima di questo particolare grafo: ogni colore rappresenta uno stack diverso e nodi di colore uguale appartengono allo stesso stack. Ogni spigolo del grafo dei conflitti indica una incompatibilità tra elementi dei tour, le cui merci non possono essere caricate nello stesso stack. Risolvendo la co-

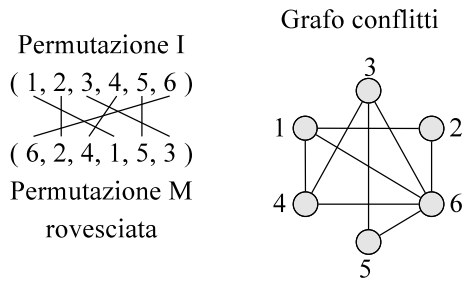


Figura 4.3: Permutazioni e grafo dei conflitti associato

lorazione su questo grafo otterremo che merci incompatibili saranno colorate diversamente e quindi inserite in stack differenti.

Normalmente il problema di colorazione rientra nella categoria dei problemi NP-hard. Per il modo in cui è stato costruito però, il nostro grafo dei conflitti non è un grafo qualsiasi ma bensì un *permutation graph*, ovvero un particolare grafo che rientra nella categoria dei grafi perfetti[2]. In questi casi è stato dimostrato che la colorazione può essere risolta come un problema di flusso massimo in tempo polinomiale[3].

Lo stesso risultato lo si può ottenere in tempo $O(N \log N)$ con un apposito algoritmo[3]: percorrendo la permutazione M al rovescio, per ogni elemento della permutazione, si può inserire l'item corrispondente in uno stack che ha cima un item con etichetta inferiore. Se l'inserimento non può essere effettuato in nessuno stack, ne viene aggiunto uno nuovo. Un esempio di come evolve l'algoritmo è mostrato in Figura 4.4.

Una volta ottenuto un loading plan è necessario ritrasformare le etichette delle merci, che durante l'operazione sono state rinominate. Per fare ciò si può ripercorrere all'indietro la relazione $Map()$.

4.2 Tour ottimi

Problema 2 *Dato un loading plan, trovare un tour di pickup minimo che rispetta la disposizione degli item all'interno del loading plan.*

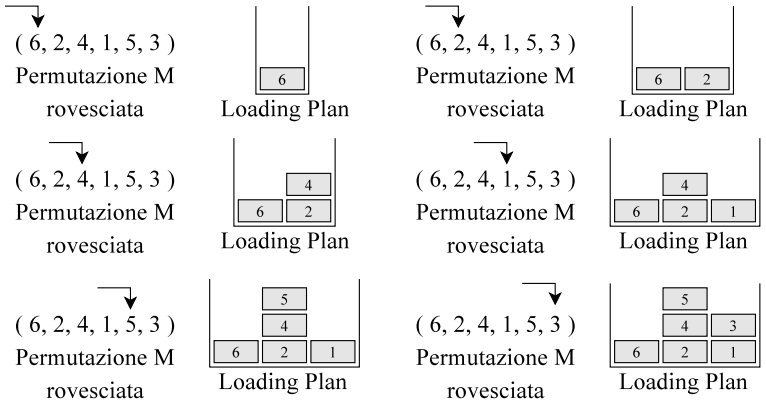
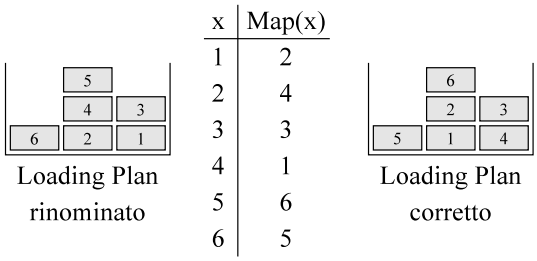


Figura 4.4: Generazione Min Stack Loading Plan



x	Map(x)
1	2
2	4
3	3
4	1
5	6
6	5

Figura 4.5: Etichettatura del loading plan

Problema 3 *Dato un loading plan, trovare un tour di delivery minimo che rispetta la disposizione degli item all'interno del loading plan.*

Proprietà 2 *Entrambi i problemi 2 e 3 possono essere risolti in tempo polinomiale.*

Dato un loading plan, un tour di delivery può essere generato scegliendo iterativamente un item in cima ad uno stack. Per conoscere quale sia il miglior tour è necessario però analizzare tutte le possibili scelte di estrazione.

Denotiamo con $f_c(s_1, s_2, \ldots, s_K, p)$ la funzione che restituisce il costo minimo di un tour parziale in cui sono stati visitati i customer corrispondenti agli item estratti dal loading plan e in cui s_k è il numero di elementi rimasti all'interno dello stack k-esimo con $k = 1 \ldots K$ e p è lo stack da cui prenderemo il prossimo elemento i.

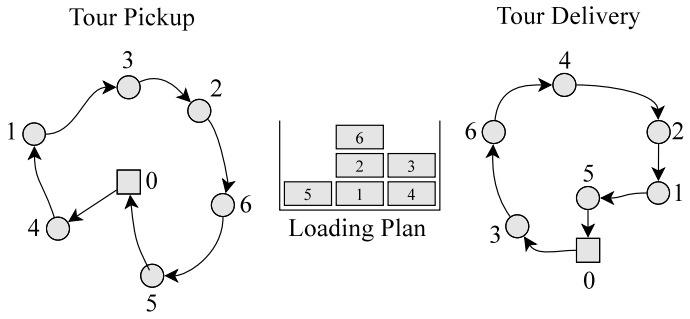


Figura 4.6: Tour e loading plan con il minimo numero di stack

Indicando con $c_{i,j}$ il costo del tragitto percorso per spostarsi da i a j, se i è il primo customer visitato allora $f_c(s_1, \ldots, s_K, p) = c_{0,i}$, in caso contrario il costo per raggiungere i sarà $f_c(s_1, \ldots, s_K, p) = min_{q=1\ldots K}\{f_c(s_1, \ldots, s_q + 1, \ldots, s_K, q) + c_{j,i}\}$, dove j è l'item preso dalla cima dello stack q.

La soluzione ottima può essere trovata in tempo $O(N^{K+1})$, quindi polinomiale rispetto a N e esponenziale solamente rispetto a K, valutando implicitamente tutti i possibili valori di $f_c()$ con un algoritmo di programmazione dinamica. È inoltre possibile ottenere il tour di pickup applicando la stessa procedura e invertendo il tour ottenuto oppure estraendo gli elementi da un loading plan con stack rovesciati.

4.3 Massimo numero di item

Sfruttando la proprietà 1 otteniamo un loading plan che potrebbe non essere valido in caso fossero utilizzati più stack del consentito. Per questo motivo potrebbe essere meglio creare un loading plan parziale che contenga il maggior numero di item in un numero di stack fissato, piuttosto che tutti gli item nel minor numero di stack.

Problema 4 *Dati un tour di pickup e uno di delivery, trovare un loading plan parziale ammissibile che contenga il numero massimo di item.*

Proprietà 3 *Il problema 4 può essere risolto in tempo polinomiale.*

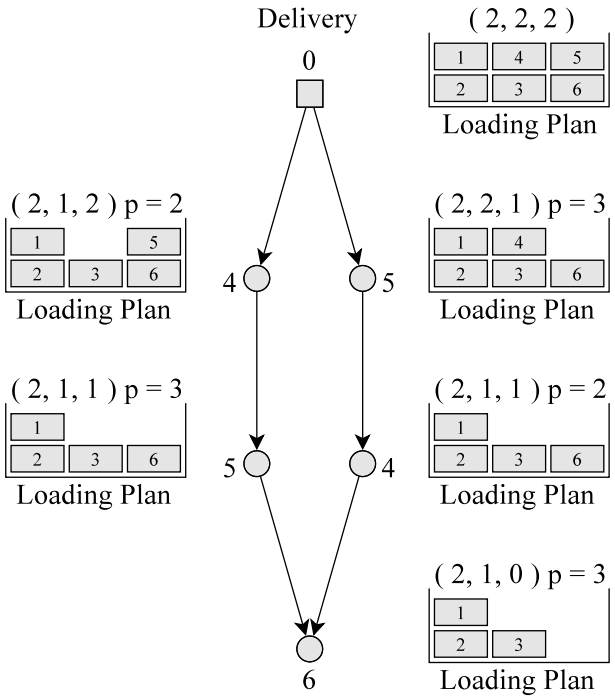


Figura 4.7: Esempio di evoluzione della procedura

Analogamente alla proprietà 1 è possibile, dati due tour, ottenere dalla permutazione di delivery, la permutazione M, in cui gli elementi sono ridefiniti (vedere Figura 4.1 e Figura 4.2).

La permutazione M può essere rappresentata graficamente su un piano cartesiano, in cui sull'asse delle ascisse sono disposti gli elementi della permutazione, mentre su quello delle ordinate le loro posizioni all'interno della permutazione. Ogni punto può essere collegato con una freccia a tutti i punti che hanno ascissa e ordinata maggiori. Il risultato è un grafo orientato in cui, per ogni elemento, si conoscono gli elementi che possono essere inseriti nello stesso stack, ovvero quali elementi sono compatibili e l'ordine in cui possono essere inseriti. Questo grafo è anche chiamato *grafo delle compatibilità*.

Il problema 4 può essere risolto come problema di flusso a costo minimo sul grafo delle compatibilità: innanzitutto è necessario collegare ogni nodo del grafo ad un nodo sorgente S con archi entranti e, con archi uscenti, ad un nodo destinazione D. La capacità massima di tutti gli archi del grafo è

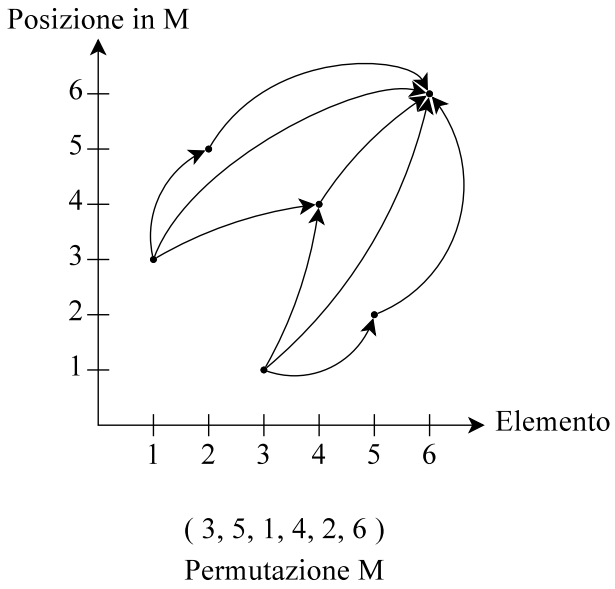


Figura 4.8: Piano cartesiano corrispondente ad una permutazione M

limitata a una unità di flusso e il costo è nullo. Anche per ogni nodo del grafo può transitare una sola unità di flusso a costo -1. L'obbiettivo è di inviare K unità di flusso dalla sorgente alla destinazione visitando il maggior numero di nodi, perché, essendo l'obbiettivo di minimizzazione, il passaggio di flusso attraverso un nodo porterà un guadagno alla soluzione.

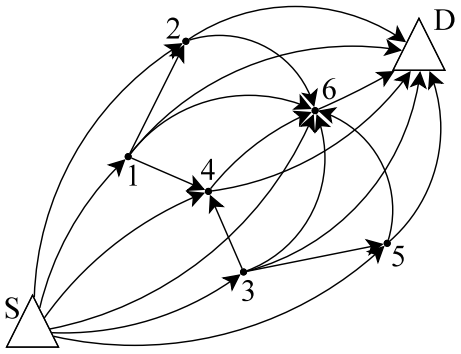


Figura 4.9: Grafo delle compatibilità

I problemi di flusso però non consentono di definire dei costi relativi alla visita dei nodi ma solo sul passaggio di flusso sugli archi. Per questo motivo è necessario operare una trasformazione sul grafo per poter risolvere il problema: ogni nodo del grafo delle compatibilità è sdoppiato in due nodi chiamati testa e coda. Ogni testa sarà collegata con tutti gli archi uscenti del nodo da sdoppiare, mentre ogni coda con tutti gli archi entranti. La coda avrà inoltre un arco uscente di capacità 1 e costo -1 che la collega alla testa.

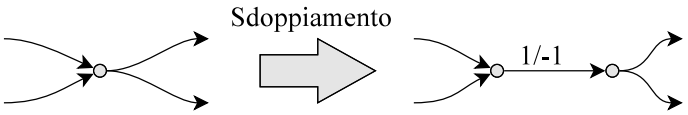


Figura 4.10: Sdoppiamento di un nodo

La soluzione finale avrà K cammini *node-disjoint* che collegano la sorgente alla destinazione. Ogni cammino rappresenta uno stack e l'ordine in cui il flusso passa attraverso i nodi è anche l'ordine in cui gli item saranno inseriti durante il tour di pickup.

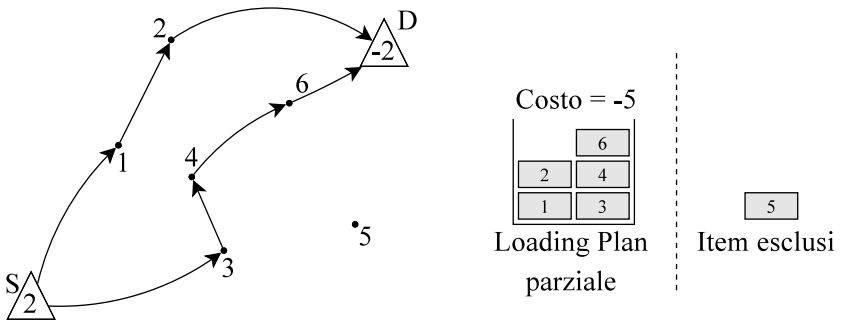


Figura 4.11: Esempio soluzione con $K = 2$

Come per il metodo risolutivo del problema 1 è necessario rinominare le etichette delle merci all'interno del loading plan.

Il problema 4 è anche noto come *Maximum K-Colorable Induced Subgraph*[11], ovvero la ricerca del massimo sottografo indotto colorabile con K

colori. Questo problema è generalmente *NP-HARD* anche su grafi perfetti [19] e sono proprio le caratteristiche del nostro problema che ci permettono di risolvere questo problema in tempo polinomiale.

CAPITOLO 5

Algoritmo

5.1 Schema generale

Il risolutore progettato, chiamato *Alternating Routing-Loading Efficient Algorithm for K-stacks Travelling Salesman Problem* (*ARLEA-KTSP*), unisce le proprietà citate nel capitolo precedente per ottenere un algoritmo euristico polinomiale. L'idea alla base dell'algoritmo è che, partendo da due tour qualsiasi e sfruttando le suddette proprietà, si possano ottenere buone soluzioni in tempi ridotti. Lo schema generale dell'algoritmo riportato nello Pseudocodice 5.1 mostra i passi principali.

Sono denotati come $\bar{L}_k$, $\bar{P}_k$ e $\bar{D}_k$ rispettivamente il loading plan, il tour di pickup e il tour di delivery parziali all'iterazione k dell'algoritmo, mentre $\hat{L}_k$, $\hat{P}_k$ e $\hat{D}_k$ rispettivamente il loading plan, il tour di pickup e il tour di delivery completi che compongono una soluzione ammissibile all'iterazione k.

Si può denotare come $|\bar{L}|$ (cardinalità di $\bar{L}$) l'operatore che restituisce il numero totale di item che sono stati inseriti all'interno di un loading plan parziale.

Sia inoltre la funzione $reverse()$ la funzione che inverte il senso di percorrenza di un tour.

input: $\{P_0, D_0, G^P, G^D\}$
output: $\{L_k, P_k, D_k\}$
  $\bar{L}_0 \leftarrow \emptyset$
  $k \leftarrow 1$
  repeat
    $\bar{L}_k \leftarrow getLoadingPlan(P_{k-1}, D_{k-1})$
    $\bar{P}_k \leftarrow getTour(\bar{L}_k, G^P)$
    $\bar{D}_k \leftarrow reverse(getTour(\bar{L}_k, G^D))$
    $P_k \leftarrow getTourNextIteration(\bar{P}_k, \bar{L}_k, G^P)$
    $D_k \leftarrow getTourNextIteration(\bar{D}_k, \bar{L}_k, G^D)$
    $\hat{L}_k \leftarrow fillLoadingPlan(\bar{P}_k, \bar{D}_k, \bar{L}_k, G^P, G^D)$
    $\hat{P}_k \leftarrow getTour(\hat{L}_k, G^P)$
    $\hat{D}_k \leftarrow reverse(getTour(\hat{L}_k, G^D))$
    $k \leftarrow k + 1$
  until $|\bar{L}_k| \leq |\bar{L}_{k-1}|$

Pseudocodice 5.1: Algoritmo generale

L'algoritmo riceve in ingresso i due tour iniziali P_0 e D_0, e i corrispettivi grafi G^P e G^D, i quali definiscono i costi degli archi.

Successivamente viene calcolato il loading plan parziale $\bar{L}_k$ dell'iterazione k-esima sfruttando la proprietà 3 del problema, mostrata nel capitolo precedente.

Una volta generato un loading plan possiamo risalire a tour compatibili sfruttando programmazione dinamica descritta nella proprietà 2. La proprietà vale sia su loading plan completi che parziali, permettendoci quindi di trovare tour parziali ottimi ($\bar{P}_k$ e $\bar{D}_k$) compatibili con $\bar{L}_k$.

Il passo successivo è quello di creare due tour per l'iterazione successiva, partendo dai tour parziali e inserendo tutti gli elementi esclusi dal loading plan parziale.

La soluzione dell'iterazione k si ottiene completando il loading plan parziale e generando da questo i tour finali tramite programmazione dinamica.

Possiamo definire una condizione di terminazione del ciclo sfruttando la seguente proprietà:

Proprietà 4 *Ogni loading plan parziale $\bar{L}_k$ non contiene mai un numero di item inferiore a quello del loading plan parziale $\bar{L}_{k-1}$ dell'iterazione precedente.*

Dati una coppia di tour di pickup e delivery parziali e un loading plan parziale compatibili, se si aggiungono dei nuovi customer ai tour non si fa altro che aggiungere archi al grafo delle compatibilità. Ciò comporta che, nella migliore delle ipotesi, risolvendo il problema 4 sfruttando la proprietà 3, si possano aggiungere ulteriori item al loading plan parziale. Nel caso invece non venissero aggiunti nuovi archi al grafo delle compatibilità, dalla risoluzione del problema di flusso otterremmo lo stesso loading plan da cui abbiamo generato i tour parziali.

5.2 Inserimento elementi esclusi

L'inserimento degli elementi esclusi da un loading plan all'interno dei tour è un passo chiave sia per ottenere la base per l'iterazione successiva, sia per ottenere una soluzione ammissibile finale.

Prima di parlare dell'algoritmo però è necessario definire alcune regole sintattiche utilizzate:

appartenenza: un item i appartiene ad L ($i \in L$) se compare in una delle K sequenze ordinate del loading plan L. Viceversa i non appartiene ad L ($i \notin L$) se non compare in nessuno stack;

precedenza: ogni coppia di item i e j successivi all'interno del loading plan sarà indicata con (i, j) dove $i \prec_L j$;

inserimento: l'inserimento di un nuovo elemento k tra due item successivi i e j di uno stack, comporta l'aggiunta e la rimozione di alcune precedenze. Usando una notazione insiemistica si può definire l'inserimento come $L \bigcup (i,k),(k,j)\ (i,j)$. In modo simile si può definire l'inserimento di un nuovo customer k tra i customer i e j in un tour T usando la sintassi $T \bigcup (i,k),(k,j)\ (i,j)$, in cui vengono aggiunti gli archi che collegano i a k e k a j e rimosso l'arco che collegava i a j.

La procedura $getTourNextIteration()$ mostrata in 5.2 descrive ciò che avviene durante la creazione dei tour necessari all'iterazione successiva: per ogni elemento escluso k viene ricercata all'interno del tour T la miglior posizione per l'inserimento, ovvero quello meno costoso. Trovata la posizione, si può aggiungere il nodo al tour. Dato che vengono trascurati i vincoli di precedenza, la procedura è indipendente per ogni tour.

input: $\{T, L, G^T\}$
output: $\{T\}$
  for all $k \notin L$ **do**
    $(s,t) \leftarrow \text{argmin}_{(i,j)\in T}\{c^T_{ik} + c^T_{kj} - c^T_{ij}\}$
    $T \leftarrow T \bigcup \{(s,k),(k,t)\} \setminus \{(s,t)\}$
  end for

Pseudocodice 5.2: Procedura $getTourNextIteration()$

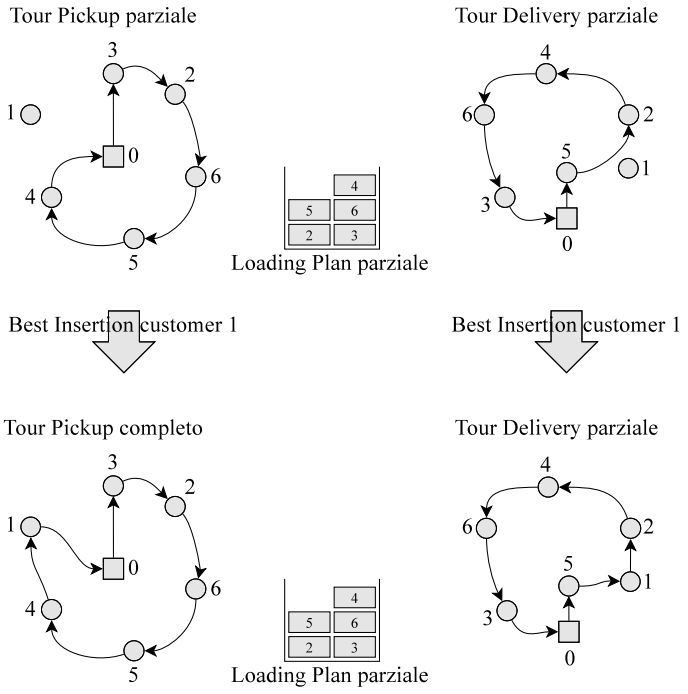


Figura 5.1: Esempio procedura $getTourNextIteration()$

La procedura per completare un loading plan $fillLoadingPlan()$, mostrata in Pseudocodice 5.3, sceglie la miglior posizione all'interno di L in cui

inserire l'item k, ovvero, per ogni tour cerca il miglior punto compreso tra due elementi consecutivi del loading plan. Il costo di ogni elemento è dato dalla somma dei costi dei due tour, che a differenza della procedura 5.2 non possono essere trattati in modo indipendente.

input: $\{P, D, L, G^P, G^D\}$
output: $\{L\}$
  for all $k \notin L$ **do**
    for all $(l, m) \in L$ **do**
      $\hat{c}^P_{klm} \leftarrow \min_{(i,j)\in P | l \prec_P i, j \prec_P m}\{c^P_{ik} + c^P_{kj} - c^P_{ij}\}$
      $\hat{c}^D_{klm} \leftarrow \min_{(i,j)\in D | m \prec_D i, j \prec_D l}\{c^D_{ik} + c^D_{kj} - c^D_{ij}\}$
    end for
    $(l^*, m^*) \leftarrow \text{argmin}_{(l,m)\in L}\{\hat{c}^P_{klm} + \hat{c}^D_{klm}\}$
    $L \leftarrow L \bigcup \{(l^*, k), (k, m^*)\} \setminus \{(l^*, m^*)\}$
  end for

Pseudocodice 5.3: Procedura $fillLoadingPlan()$

5.3 Gestione del vincolo di capacità

Nella realtà non è possibile avere dei mezzi di trasporto con una capacità illimitata, per questo motivo si impone un vincolo che limita la dimensione massima di ogni stack. Nell'algoritmo si è scelto di controllare che il vincolo sia soddisfatto solamente dopo aver generato un loading plan parziale, continuando a sfruttare tutte le altre proprietà teoriche e lasciando inalterata la parte di generazione del loading plan.

Per far si che le soluzioni prodotte rispettino il vincolo di capacità del loading plan, un primo cambiamento deve essere apportato alla procedura $fillLoadingPlan()$ (Pseudocodice 5.3): l'inserimento di un item all'interno del loading plan parziale può avvenire solamente in stack che contengono un numero di elementi inferiore al limite massimo di capacità. Questo ci permette di affermare che se un loading plan parziale rispetta il vincolo della capacità prima di essere completato, lo rispetterà anche dopo l'esecuzione della procedura.

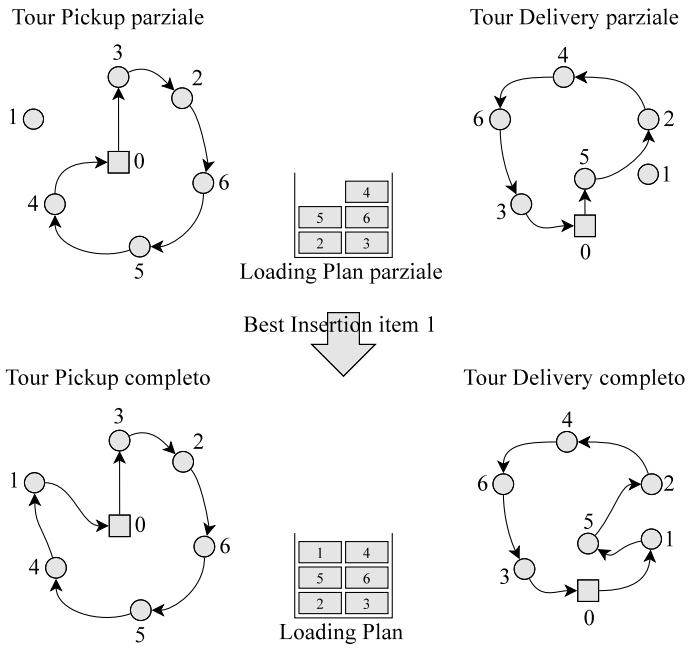


Figura 5.2: Esempio procedura $fillLoadingPlan()$

Purtroppo non sempre si ottiene un loading plan parziale in cui tutti gli stack non violano i vincoli imposti. Per risolvere anche questo problema occorre effettuare un'operazione di *taglio* degli stack (Figura 5.3), eliminando gli elementi in eccesso. Questa operazione può essere fatta in tre momenti diversi:

- dopo aver generato un loading plan parziale;
- dopo aver calcolato i tour di pickup e delivery ottimi;
- al termine delle iterazioni.

Risultati sperimentali preliminari hanno evidenziato che la seconda opzione è la migliore.

Prima di effettuare il taglio del loading plan, può essere utile ricercare eventuali item che possono essere spostati in stack diversi, avvicinandosi così al numero massimo consentito o, nella migliore delle ipotesi, rispettare il vincolo di capacità. Un esempio è mostrato in Figura 5.4.

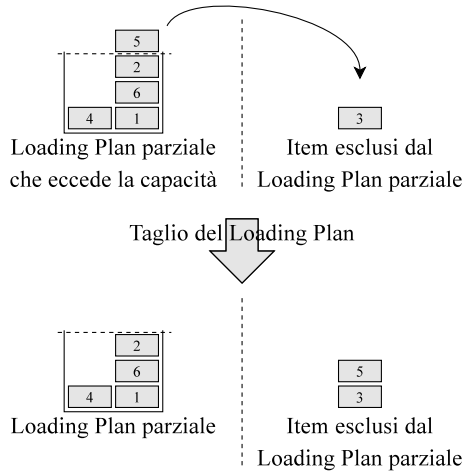


Figura 5.3: Esempio di *taglio* del loading plan

Quando un item e può essere inserito in più stack, è possibile spostarlo senza diminuire il numero complessivo di item presenti nel loading plan e senza aumentare il costo dei tour parziali. Questa operazione, denominata $Move()$ e descritta nello Pseudocodice 5.4, ricerca due item consecutivi i e j, tali che $i \prec_P e$ e $e \prec_P j$ nel tour di pickup P e viceversa $j \prec_D e$ e $e \prec_D i$ nel tour di delivery D, in uno stack k in cui il numero di item già presenti, $|Stack(k)|$, è inferiore alla capacità massima C.

input: $\{P, D, L, e\}$
output: $\{L\}$
  for all $k = 1 \dots K$ **s.t.** $|Stack(k)| < C$ **do**
    for all $(l, m) \in Stack(k)$ **s.t.** $l \prec_P e, e \prec_P m, m \prec_D e, e \prec_D l$ **do**
      let r **s.t.** $r \prec_L e$
      let s **s.t.** $e \prec_L s$
      $L \leftarrow L \setminus \{(r, e), (e, s), (l, m)\} \bigcup \{(r, s), (l, e), (e, m)\}$
    end for
  end for

Pseudocodice 5.4: Procedura $Move()$

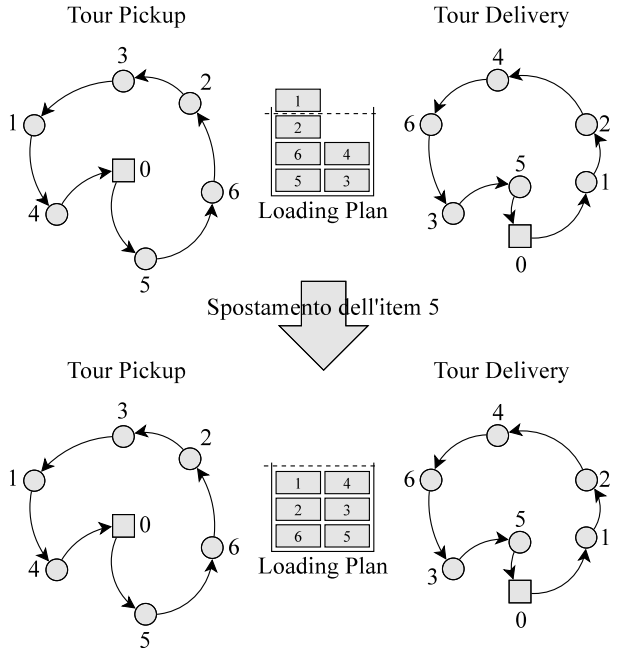


Figura 5.4: Esempio di spostamento di un item

5.4 Punti di restart alternativi

Un modo per ottenere risultati migliori è quello di avere più punti di partenza per l'algoritmo. Questo consente di esplorare uno spazio delle soluzioni maggiore e generalmente di ottenere soluzioni migliori.

Nell'algoritmo vengono implementati due tipi di meccanismi per ottenere dei punti di restart:

- modificando o generando più tour iniziali;
- modificando dei loading plan.

5.4.1 Restart dei tour

Per quanto riguarda i tour è stato implementato un meccanismo che inverte i tour iniziali in modo da poter eseguire l'algoritmo 5.1 su più coppie di ingressi:

- pickup e delivery originali;
- pickup originale e delivery invertito;
- pickup invertito e delivery originale;
- pickup e delivery invertiti.

Ogni coppia di tour produce un loading plan differente poiché differenti sono le precedenze da rispettare.

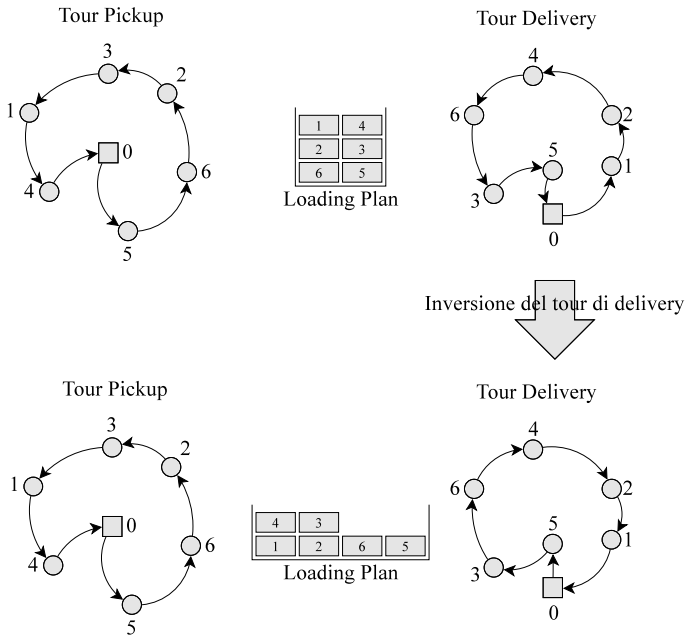


Figura 5.5: Esempio di loading plan per tour invertiti

5.4.2 Restart dei loading plan

Il meccanismo di restart basato sui loading plan modificati è più complesso rispetto a quello dei tour: in un loading plan parziale gli item esclusi sono tali perché incompatibili con altri item presenti nel loading plan. Escludendo questi ultimi si ha la possibilità di inserire i primi nel loading plan. Si può quindi definire l'operatore $Change()$ che, dato un loading plan parziale L e

un item e escluso, genera un loading plan parziale in cui viene inserito e ed esclusi gli item in conflitto (Figura 5.6), ovvero gli item collegati a e da uno spigolo nel grafo dei conflitti G^C. L'operazione però è effettuata solamente su stack in cui è presente una sola incompatibilità, questo per mantenere massimo il numero di item presenti nel loading plan. Dato che e può avere incompatibilità con più item in stack differenti, l'operatore $Change()$ non genererà solamente un loading plan ma un insieme di loading plan $\{L_1, \ldots, L_p\}$ in cui, per ogni elemento dell'insieme, e è inserito in uno stack diverso.

input: $\{P, D, L, G^P, G^D, G^C, e\}$
output: $\{L_1, \ldots, L_p\}$
  $p \leftarrow 1$
  for all $k = 1 \ldots K$ **s.t.** $|\{(e,t) \in G^C | t \in Stack(k)\}| = 1$ **do**
    let $r \in L$ **s.t.** $r \prec_L t$
    let $s \in L$ **s.t.** $t \prec_L s$
    $L_p \leftarrow L \setminus \{(r,t),(t,s)\} \bigcup \{(r,s)\}$
    for all $(l,m) \in Stack(k)$ **do**
      $\hat{c}^P_{eklm} \leftarrow \min_{(i,j) \in P | l \prec_P i, j \prec_P m} \{c^P_{ie} + c^P_{ej} - c^P_{ij}\}$
      $\hat{c}^D_{eklm} \leftarrow \min_{(i,j) \in D | m \prec_D i, j \prec_D l} \{c^D_{ie} + c^D_{ej} - c^D_{ij}\}$
    end for
    $(l^*, m^*) \leftarrow \text{argmin}_{(l,m) \in L} \{\hat{c}^P_{eklm} + \hat{c}^D_{eklm}\}$
    $L_p \leftarrow L_p \bigcup \{(l^*, e), (e, m^*)\} \setminus \{(l^*, m^*)\}$
    $p \leftarrow p + 1$
  end for

Pseudocodice 5.5: Procedura $Change()$

Utilizzando diverse volte l'operatore $Change()$ con differenti item esclusi, si possono ottenere molte combinazioni di loading plan parziali modificati. L'operatore $Change()$ può essere inoltre usato su loading plan parziali già modificati, effettuando così molteplici scambi di item differenti sul medesimo piano di carico.

5.5 Particolari dell'implementazione

Il linguaggio scelto per l'implementazione dell'algoritmo è il C. La decisione è stata presa per l'efficienza del linguaggio e per l'integrazione del nostro

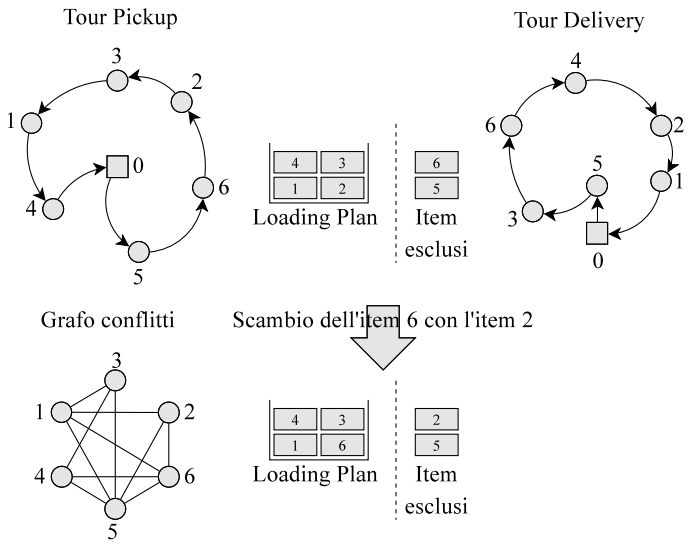


Figura 5.6: Esempio di scambio di un item

programma con librerie preesistenti scritte anch'esse in C.

Per ottenere dei tour iniziali è stata utilizzata la libreria *Concorde*[1], una suite di algoritmi per la risoluzione del problema del TSP. Nell'implementazione è stato utilizzato l'algoritmo *Chained Lin-Kernighan* (*CLK*)[17], un algoritmo di ricerca locale che iterativamente effettua una sequenza di 2-scambi per ottenere soluzioni meno costose.

La generazione di un loading plan parziale con massimo numero di item (Proprietà 3) è realizzata sfruttando la libreria esterna *MCF*[12] per la risoluzione di problemi di flusso. L'algoritmo messo a disposizione è stato sfruttato per risolvere un problema di flusso a costo minimo ed integrato nel nostro algoritmo. Tra gli algoritmi disponibili abbiamo scelto il *primal network simplex*, in quanto algoritmo di default.

Nello pseudocodice 5.1 alla fine di ogni iterazione vengono generato i tour ottimi completi tramite la procedura di programmazione dinamica. Dato che questa procedura è la più complessa dell'algoritmo, è stato invece scelto di aggiungere i customer ai tour di visita durante la procedura $fillLoadingPlan()$, che fornisce soluzioni di poco peggiori ma in tempi ridotti.

CAPITOLO 6

Analisi sperimentale

L'algoritmo proposto è stato implementato in C ed eseguito su un calcolatore con processore da 1833MHz. Le istanze scelte per i test, disponibili al sito internet `http://www.espaciotd.jazztel.es/dtspmsEn.htm`, sono costituite in parte dalle prime istanze generate da Hanne L. Petersen e in parte nuove. Tutte sono in formato *tsp*, compatibile con la libreria Concorde[1]. In queste istanze i customer sono disposti in un piano di dimensioni 100×100 in cui il deposito è posizionato nel punto di coordinate $(50, 50)$. Il costo di uno spostamento tra due customer è calcolato come la distanza euclidea tra i due punti con arrotondamento all'intero più vicino.

6.1 Test versus

Il test chiamato *versus* riguarda il confronto dei risultati ottenuti con il nostro algoritmo e quelli ottenuti con altri risolutori. Nel nostro caso sono stati scelti i risultati forniti dall'algoritmo HVNS e le migliori soluzioni fornite in letteratura, ottenute dall'esecuzione di un algoritmo di LNS per diverse ore.

Per questo test il numero di stack usati per il loading plan è stato fissato a 3, in quanto questo è il limite scelto in letteratura. Per quanto riguarda i parametri del nostro algoritmo è stata scelta una combinazione di fattori che combinasse velocità e risultati. Per questo motivo sono stati scartati i punti di restart sui loading plan, mentre sono stati mantenuti quelli sui tour di partenza. Gli stack delle soluzioni fornite sono inoltre limitati in capacità per uniformarsi ai concorrenti. Ciò che è importante valutare in questo test sono i costi delle soluzioni e i tempi con cui questi sono stati forniti.

I risultati dei test sono riportati nelle tabelle da 6.1 a 6.3:

- in colonna 1 sono indicate le istanze di test;
- in colonna 2 sono riportate le soluzioni ottenute con l'algoritmo LNS, indicate come **Best**;
- in colonna 3 sono riportati i rapporti tra i risultati dell'algoritmo HVNS e i risultati dell'algoritmo LNS, indicati come /**Best**;
- in colonna 4 sono riportati i valori delle soluzioni del nostro algoritmo;
- in colonna 5 sono riportati i rapporti tra i nostri risultati e quelli dell'algoritmo LNS, indicati con /**Best**;
- in colonna 6 sono riportati i rapporti tra le nostre soluzioni e quelle dell'HVNS, indicati con /**HVNS**;
- in colonna 7 sono riportati i tempi di esecuzione del nostro algoritmo.

6.1.1 Istanze di 33 customer

Nella Tabella 6.1 vengono mostrati i risultati ottenuti su istanze di 33 customer. Quelli dell'HVNS sono molto vicini alle migliori soluzioni conosciute, perciò i valori riportati in colonna 5 e 6 sono molto simili. Mediamente la nostra implementazione fornisce risultati peggiori del 10% rispetto alle soluzioni di confronto. La varianza dei risultati è però molto alta: con alcune istanze si riescono ad ottenere valori molto inferiori della media, mentre in altre i risultati rimangono addirittura sopra il 13%.

Il nostro algoritmo termina però in un tempo medio 100 volte inferiore rispetto al tempo d'esecuzione dell'HVNS.

Istanza	**Best**	**HVNS**	**ARLEA-KTSP**			
		/Best	Risultato	/Best	/HVNS	Tempo (s)
R00	1063	1,008	1133	1,066	1,057	0,14
R01	1032	1,008	1132	1,097	1,088	0,07
R02	1065	1,008	1162	1,091	1,082	0,13
R03	1100	1,000	1265	1,150	1,150	0,08
R04	1052	1,005	1130	1,074	1,069	0,10
R05	1008	1,022	1116	1,107	1,083	0,14
R06	1110	1,000	1275	1,149	1,149	0,12
R07	1105	1,004	1264	1,144	1,139	0,08
R08	1109	1,000	1200	1,082	1,082	0,07
R09	1091	1,000	1191	1,092	1,092	0,10
R10	1016	1,000	1139	1,121	1,121	0,08
R11	1001	1,000	1101	1,100	1,100	0,11
R12	1109	1,002	1211	1,092	1,090	0,09
R13	1084	1,000	1219	1,125	1,125	0,09
R14	1034	1,017	1127	1,090	1,072	0,08
R15	1142	1,014	1270	1,112	1,097	0,07
R16	1093	1,000	1173	1,073	1,073	0,08
R17	1073	1,009	1175	1,095	1,085	0,09
R18	1118	1,028	1251	1,119	1,088	0,11
R19	1089	1,006	1201	1,103	1,096	0,11
Media		1,007		1,104	1,097	0,10

Tabella 6.1: Risultati test *versus* su istanze di 33 customer

6.1.2 Istanze di 66 customer

Aumentando il numero di customer delle istanze si può notare che, mediamente, sia i nostri risultati, sia i risultati dell'HVNS peggiorano. Il rapporto medio con quest'ultimo rimane però molto vicino a quello ottenuto con istanze da 33 customer.

Nonostante il tempo medio d'esecuzione sia 6 volte quello delle istanze da 33 customer, è ancora di gran lunga inferiore rispetto ai dieci secondi utilizzati dall'HVNS per fornire le proprie soluzioni.

La Tabella 6.2 riporta nel dettaglio i risultati del test.

Istanza	**Best**	**HVNS**	**ARLEA-KTSP**			
		/Best	Risultato	/Best	/HVNS	Tempo (s)
R00	1594	1,038	1928	1,210	1,165	0,60
R01	1600	1,064	1962	1,226	1,152	0,64
R02	1576	1,077	1900	1,206	1,119	0,62
R03	1631	1,059	2002	1,227	1,159	0,57
R04	1611	1,077	1858	1,153	1,071	0,69
R05	1528	1,069	1860	1,217	1,139	0,49
R06	1651	1,105	1845	1,118	1,011	0,51
R07	1653	1,064	1931	1,168	1,098	0,64
R08	1607	1,111	1875	1,167	1,050	0,75
R09	1598	1,086	1869	1,170	1,077	0,59
R10	1702	1,078	2045	1,202	1,115	0,62
R11	1575	1,067	1967	1,249	1,170	0,59
R12	1652	1,060	1969	1,192	1,124	0,57
R13	1617	1,087	1899	1,174	1,080	0,66
R14	1611	1,066	1962	1,218	1,142	0,72
R15	1608	1,065	1863	1,159	1,088	0,47
R16	1725	1,082	2057	1,192	1,102	0,51
R17	1627	1,075	1936	1,190	1,107	0,52
R18	1671	1,065	2018	1,208	1,134	0,57
R19	1635	1,052	2027	1,240	1,178	0,57
Media		1,072		1,194	1,114	0,60

Tabella 6.2: Risultati test *versus* su istanze di 66 customer

6.1.3 Istanze di 132 customer

La dimensione massima delle istanze prese in considerazione è 132 customer. Come si può notare dalla Tabella 6.3, i rapporti di entrambi gli algoritmo sono peggiorati, similmente a ciò che era accaduto per le istanze minori. Il nostro algoritmo, come in precedenza, sembra però mantenere un distacco dall'HVNS mediamente costante, intorno al 10% .

Purtroppo è da segnalare che il rapporto tra il nostro tempo d'esecuzione e quello dell'HVNS (fissato sempre a dieci secondi) è aumentato di molto, passando da $\frac{1}{100}$ per istanze con 33 customer, ad $\frac{1}{2}$ per quelle con 132.

6.2 Test stack

Il test denominato *stack* si pone come obbiettivo l'analisi dei tempi necessari al nostro algoritmo per fornire una soluzione, al variare del numero di stack del problema.

Istanza	**Best**	**HVNS**		**ARLEA-KTSP**		
		/Best	Risultato	/Best	/HVNS	Tempo (s)
R00	2591	1,157	3173	1,225	1,103	5,40
R01	2645	1,167	3282	1,241	1,118	4,42
R02	2639	1,139	3218	1,219	1,079	4,18
R03	2752	1,124	3315	1,205	1,099	4,95
R04	2603	1,131	3201	1,230	1,131	4,82
R05	2616	1,158	3284	1,255	1,142	5,82
R06	2576	1,160	3155	1,225	1,081	4,66
R07	2615	1,147	3182	1,217	1,101	5,20
R08	2638	1,143	3153	1,195	1,073	5,59
R09	2554	1,136	3027	1,185	1,062	5,60
R10	2646	1,190	3217	1,216	1,069	6,02
R11	2632	1,133	3271	1,243	1,122	4,36
R12	2555	1,185	3234	1,266	1,116	5,06
R13	2659	1,157	3157	1,187	1,101	5,02
R14	2605	1,140	3151	1,210	1,087	5,02
R15	2626	1,185	3206	1,221	1,116	4,70
R16	2534	1,160	3244	1,280	1,131	5,59
R17	2569	1,142	3104	1,208	1,068	4,68
R18	2652	1,151	3179	1,199	1,072	5,36
R19	2644	1,160	3208	1,213	1,129	4,86
Media		1,153		1,222	1,100	5,07

Tabella 6.3: Risultati test *versus* su istanze di 132 customer

Ovviamente i risultati migliorano aumentando il numero di stack. Questa affermazione è valida in generale per tutti gli algoritmi, perché aumentare il numero di stack significa rilassare il problema. Nel caso del nostro algoritmo, aumentare il numero di stack significa però aumentare sensibilmente anche il tempo d'esecuzione: la fase di programmazione dinamica, che permette di ottenere un tour ottimo dal loading plan, ha una complessità $O(N^{K+1})$, dove N è il numero dei customer e K è il numero di stack utilizzati. Quindi il tempo cresce in maniera esponenziale relativamente al numero di stack. Per questo motivo l'analisi dei risultati sarà concentrata sui tempi piuttosto che sui risultati veri e propri.

L'algoritmo è stato eseguito limitando la capacità degli stack e con più tour di partenza come nel test *versus*. Il numero di stack è stato fatto variare tra 2, 3 e 4.

Nelle tabelle da 6.4 a 6.6 sono riportati i risultati:

- in colonna 1 sono indicate le istanze di test;

- in colonna 2 sono riportate le soluzioni dell'algoritmo eseguito con 2 stack;
- in colonna 3 sono riportati i tempi dell'algoritmo eseguito con 2 stack;
- in colonna 4 sono riportate le soluzioni dell'algoritmo eseguito con 3 stack;
- in colonna 5 sono riportati i tempi dell'algoritmo eseguito con 3 stack;
- in colonna 6 sono riportate le soluzioni dell'algoritmo eseguito con 4 stack;
- in colonna 7 sono riportati i tempi dell'algoritmo eseguito con 4 stack.

6.2.1 Istanze di 33 customer

In Tabella 6.4 si può notare come i risultati migliorino sensibilmente aumentando il numero di stack. Come già detto però il tempo di esecuzione tende ad esplodere: già da queste istanze si può notare una crescita sempre più rapida.

6.2.2 Istanze di 66 customer

Come per le istanze da 33 customer, anche per quelle da 66 (Tabella 6.5) otteniamo risultati decisamente migliori e, come in precedenza, anche il tempo cresce notevolmente: mentre per risolvere istanze fino a 3 stack l'algoritmo impiega meno di un secondo, per risolverne una con 4 stack sono necessari oltre dieci secondi.

6.2.3 Istanze di 132 customer

Con i risultati delle istanze di 132 customer (riportati in Tabella 6.6) si può facilmente notare l'esplosione del tempo: l'algoritmo impiega meno di 2 secondi per risolvere le istanze usando due stack. Impiega invece oltre 100 volte tanto per risolvere le istanze con 4 stack.

Istanza	2 stack		3 stack		4 stack	
	Risultato	Tempo (s)	Risultato	Tempo (s)	Risultato	Tempo (s)
R00	1381	0,03	1133	0,14	1025	0,56
R01	1257	0,02	1132	0,07	980	0,57
R02	1249	0,04	1162	0,13	1020	0,66
R03	1366	0,04	1265	0,08	1129	0,49
R04	1326	0,04	1130	0,10	996	0,76
R05	1239	0,05	1116	0,14	948	0,70
R06	1404	0,03	1275	0,12	1082	0,64
R07	1411	0,02	1264	0,08	1104	0,67
R08	1393	0,03	1200	0,07	1097	0,67
R09	1300	0,04	1191	0,10	1069	0,63
R10	1263	0,03	1139	0,08	999	0,71
R11	1176	0,04	1101	0,11	993	0,95
R12	1427	0,03	1211	0,09	1095	0,85
R13	1319	0,03	1219	0,09	1032	0,88
R14	1337	0,03	1127	0,08	1039	0,56
R15	1475	0,03	1270	0,07	1113	0,54
R16	1262	0,03	1173	0,08	1102	0,61
R17	1274	0,04	1175	0,09	1081	0,48
R18	1411	0,05	1251	0,11	1146	0,78
R19	1423	0,04	1201	0,11	1015	0,78
Media		0,03		0,10		0,67

Tabella 6.4: Risultati test *stack* su istanze di 33 customer

Questi risultati fanno capire quanto l'intero risolutore dipenda fortemente dalla fase di programmazione dinamica. Un buon modo per ottenere un algoritmo più rapido è quindi quello di ottimizzare questa fase critica, che rappresenta il collo di bottiglia dell'intero algoritmo.

6.3 Test statistics

I test denominati *statistics*, i cui risultati sono riportati nelle tabelle da 6.7 a 6.9, mettono in evidenza alcune caratteristiche dell'algoritmo analizzando alcuni dati statistici:

- in colonna 1 sono indicate le istanze;
- in colonna 2 è riportato il numero massimo di iterazioni effettuate;
- in colonna 3 è riportato l'iterazione in cui si è trovata la miglior soluzione;

Istanza	2 stack		3 stack		4 stack	
	Risultato	Tempo (s)	Risultato	Tempo (s)	Risultato	Tempo (s)
R00	2153	0,18	1928	0,60	1596	11,76
R01	2276	0,20	1962	0,64	1755	12,55
R02	2170	0,24	1900	0,62	1628	10,06
R03	2233	0,20	2002	0,57	1674	11,44
R04	2130	0,21	1858	0,69	1616	9,59
R05	1985	0,16	1860	0,49	1581	9,78
R06	2139	0,17	1845	0,51	1671	12,49
R07	2229	0,18	1931	0,64	1722	9,20
R08	2142	0,20	1875	0,75	1660	12,78
R09	2034	0,19	1869	0,59	1653	8,81
R10	2367	0,14	2045	0,62	1811	8,01
R11	2199	0,27	1967	0,59	1677	13,63
R12	2244	0,16	1969	0,57	1765	9,95
R13	2273	0,13	1899	0,66	1706	9,28
R14	2222	0,18	1962	0,72	1690	8,98
R15	2206	0,15	1863	0,47	1653	9,17
R16	2334	0,14	2057	0,51	1782	8,44
R17	2333	0,17	1936	0,52	1762	8,95
R18	2304	0,19	2018	0,57	1799	12,41
R19	2278	0,18	2027	0,57	1704	11,48
Media		0,18		0,60		10,44

Tabella 6.5: Risultati test *stack* su istanze di 66 customer

- in colonna 4 è riportato il numero minimo di item esclusi dal loading plan parziale;

- in colonna 5 è riportato il numero di item esclusi dal loading plan parziale nell'iterazione in cui si è trovata la miglior soluzione.

Nelle tabelle successive le istanze saranno sempre indicate in colonna 1.

L'algoritmo è stato eseguito con 4 punti di restart, per questo motivo è stato scelto di riportare solo i valori massimi e minimi delle 4 esecuzioni.

6.3.1 Istanze di 33 customer

I test sulle istanze da 33 customer (in Tabella 6.7) mostrano come l'algoritmo effettui pochissime iterazioni. Mediamente l'algoritmo termina dopo sole 2 iterazioni e molto spesso già alla prima ottiene la soluzione migliore. Generalmente l'algoritmo termina con un loading plan parziale che esclude circa un terzo degli item.

Istanza	2 stack		3 stack		4 stack	
	Risultato	Tempo (s)	Risultato	Tempo (s)	Risultato	Tempo (s)
R00	3640	1,25	3173	5,40	2841	163,51
R01	3715	1,16	3282	4,42	2864	169,89
R02	3747	1,18	3218	4,18	2877	157,24
R03	3870	1,25	3315	4,95	2983	150,48
R04	3595	1,42	3201	4,82	2748	162,06
R05	3596	1,04	3284	5,82	2834	183,60
R06	3555	1,72	3155	4,66	2824	152,56
R07	3585	1,47	3182	5,20	2677	156,00
R08	3706	1,18	3153	5,59	2781	166,86
R09	3422	1,38	3027	5,60	2725	168,40
R10	3760	1,32	3217	6,02	2750	168,80
R11	3664	1,24	3271	4,36	2773	163,37
R12	3604	1,34	3234	5,06	2773	179,11
R13	3659	1,18	3157	5,02	2785	154,92
R14	3542	0,94	3151	5,02	2761	161,05
R15	3768	1,42	3206	4,70	2681	148,74
R16	3686	1,16	3244	5,59	2852	130,72
R17	3612	1,47	3104	4,68	2830	136,24
R18	3698	1,52	3179	5,36	2820	159,47
R19	3752	1,27	3208	4,86	2790	187,93
Media		1,30		5,07		161,05

Tabella 6.6: Risultati test *stack* su istanze di 132 customer

6.3.2 Istanze di 66 customer

In Tabella 6.8 sono riportati i risultati dei test su istanze da 66 customer: rispetto alle istanze con 33 customer non ci sono grandi cambiamenti per ciò che riguarda le iterazioni effettuate, infatti l'algoritmo necessita mediamente di 3 iterazioni e anche in questo caso la migliore soluzione è spesso trovata alla prima. Un cambiamento significativo lo si può però notare per ciò che riguarda il numero degli elementi esclusi: con istanze da 66 customer si riesce di poco a superare la metà degli elementi inseriti nell'ultimo loading plan parziale.

6.3.3 Istanze di 132 customer

Dai test effettuati su istanze con 132 customer (Tabella 6.9) si nota subito come il numero minimo di elementi esclusi in un loading plan parziale sia addirittura sempre sopra la metà degli item, ciò significa che non abbiamo mai un loading plan parziale che include almeno metà degli item. Le iterazioni

Istanza	Massima iterazione	Iterazione Best	Minimo esclusi	Esclusi Best
R00	2	2	11	12
R01	2	2	11	11
R02	2	1	9	15
R03	2	1	10	14
R04	2	1	11	12
R05	3	2	8	8
R06	3	2	11	13
R07	2	1	12	13
R08	3	1	10	11
R09	2	1	9	11
R10	3	1	7	11
R11	2	1	9	10
R12	3	1	8	10
R13	2	2	8	8
R14	2	1	10	14
R15	2	1	10	14
R16	3	2	11	12
R17	2	1	9	11
R18	4	2	11	13
R19	3	1	10	14
Media	2,5	1,4	9,8	11,9

Tabella 6.7: Risultati test *statistics* su istanze da 33 customer

compiute sono sempre poche, segno che se si inseriscono gli item trascurando i vincoli di precedenza è molto difficile ottenere tour compatibili.

6.4 Test lp-restart

Nel test *lp-restart* abbiamo rilevato i cambiamenti delle soluzioni, all'aumentare dei punti di restart basati su modifica del loading plan.

Creare dei punti di restart fa si che generalmente l'algoritmo fornisca soluzioni migliori, ma comporta anche un tempo di calcolo maggiore.

In tutti i test visti finora, l'algoritmo è stato sempre eseguito con 4 punti di partenza iniziali, ottenuti tramite inversione dei tour. Con questo test saranno presi in considerazione i risultati ottenuti dall'esecuzione dell'algoritmo con ulteriori punti di restart, generati dai loading plan iniziali, per un totale di 10 e 50 punti di partenza. I risultati sono riportati nelle tabelle da 6.10 a 6.12:

- in colonna 1 sono indicate le istanze di test;

Istanza	Massima iterazione	Iterazione Best	Minimo esclusi	Esclusi Best
R00	3	1	29	32
R01	4	1	30	35
R02	6	1	30	33
R03	3	1	26	30
R04	4	4	28	28
R05	2	1	28	32
R06	3	1	27	30
R07	3	3	29	32
R08	3	3	29	29
R09	4	2	28	29
R10	3	1	29	31
R11	3	1	29	34
R12	4	2	26	30
R13	4	1	24	33
R14	3	3	25	34
R15	3	1	31	33
R16	3	3	29	29
R17	4	2	32	35
R18	3	2	31	32
R19	2	2	31	31
Media	3,4	1,8	28,6	31,6

Tabella 6.8: Risultati test *statistics* su istanze da 66 customer

- in colonna 2 sono riportate le soluzioni ottenute con l'algoritmo LNS, indicate come **Best**;

- in colonna 3 sono riportati i rapporti tra i risultati ottenuti con 4 punti di restart e i risultati dell'algoritmo LNS, indicati con /**Best**;

- in colonna 4 sono riportati i tempi di esecuzione del nostro algoritmo con 4 punti di restart;

- in colonna 5 sono riportati i rapporti tra i risultati ottenuti con 10 punti di restart e i risultati dell'algoritmo LNS, indicati con /**Best**;

- in colonna 6 sono riportati i tempi di esecuzione del nostro algoritmo con 10 punti di restart;

- in colonna 7 sono riportati i rapporti tra i risultati ottenuti con 50 punti di restart e i risultati dell'algoritmo LNS, indicati con /**Best**;

Istanza	Massima iterazione	Iterazione Best	Minimo esclusi	Esclusi Best
R00	5	4	75	78
R01	5	2	70	79
R02	4	3	74	76
R03	4	3	77	80
R04	4	3	74	77
R05	5	2	71	79
R06	3	3	75	75
R07	3	3	77	77
R08	5	2	70	77
R09	5	4	73	74
R10	4	2	74	75
R11	4	1	75	79
R12	4	4	72	78
R13	5	5	72	72
R14	5	4	73	77
R15	3	1	75	86
R16	5	1	75	82
R17	7	1	70	82
R18	4	3	69	74
R19	5	2	70	75
Media	4,5	2,7	73,1	77,6

Tabella 6.9: Risultati test *statistics* su istanze da 132 customer

- in colonna 8 sono riportati i tempi di esecuzione del nostro algoritmo con 50 punti di restart.

6.4.1 Istanze di 33 customer

In Tabella 6.10 si può notare come l'aggiunta di nuovi punti di partenza migliori i risultati, soprattutto nel passaggio da 4 a 10 punti di restart. Non mancano però casi in cui succede il contrario: questo è dovuto soprattutto alla diversità dei tour iniziali, che porta a differenti soluzioni finali.

Come ci si poteva aspettare l'algoritmo necessita di più tempo per terminare l'esecuzione, rispettando però le previsioni.

6.4.2 Istanze di 66 customer

Anche per le istanze da 66 customer (in Tabella 6.11) si può notare un forte miglioramento tra l'esecuzione normale e con 10 restart. Aumentare ulteriormente il numero di stack invece non provoca miglioramenti sensibili.

Istanza	Best	4 restart		10 restart		50 restart	
		/Best	Tempo (s)	/Best	Tempo (s)	/Best	Tempo (s)
R00	1063	1,066	0,14	1,046	0,20	1,046	0,74
R01	1032	1,097	0,07	1,048	0,13	1,050	0,49
R02	1065	1,091	0,13	1,056	0,33	1,050	0,66
R03	1100	1,150	0,08	1,135	0,13	1,102	0,48
R04	1052	1,074	0,10	1,065	0,23	1,074	0,53
R05	1008	1,107	0,14	1,019	0,30	1,019	0,89
R06	1110	1,149	0,12	1,093	0,23	1,091	0,64
R07	1105	1,144	0,08	1,096	0,13	1,125	0,51
R08	1109	1,082	0,07	1,092	0,15	1,082	0,74
R09	1091	1,092	0,10	1,049	0,20	1,064	0,60
R10	1016	1,121	0,08	1,132	0,15	1,056	0,76
R11	1001	1,100	0,11	1,066	0,19	1,050	0,72
R12	1109	1,092	0,09	1,079	0,21	1,077	0,90
R13	1084	1,125	0,09	1,047	0,18	1,047	0,23
R14	1034	1,090	0,08	1,109	0,19	1,109	0,62
R15	1142	1,112	0,07	1,073	0,18	1,084	0,22
R16	1093	1,073	0,08	1,076	0,23	1,052	0,74
R17	1073	1,095	0,09	1,067	0,14	1,059	0,37
R18	1118	1,119	0,11	1,123	0,20	1,130	0,64
R19	1089	1,103	0,11	1,092	0,18	1,084	0,28
Media		1,104	0,10	1,078	0,19	1,073	0,59

Tabella 6.10: Risultati test *restart* su istanze da 33 customer

Ovviamente anche in questo caso il tempo tende a crescere, ma sempre in maniera lineare.

6.4.3 Istanze di 132 customer

Il test sulle istanze da 132 customer (Tabella 6.12) conferma che aumentare i punti di restart porta a migliori soluzioni finali. Analizzando tutti i test effettuati si può notare che è il passaggio da 4 a 10 punti di restart che da il miglioramento maggiore, mentre questo è inferiore o quasi nullo quando si passa a 50 punti.

6.5 Test tour-restart

Con il test *tour-restart* si è osservato, similmente al test *lp-restart*, come cambiano le soluzioni al variare dei tour di partenza dell'algoritmo.

Istanza	**Best**	**4 restart**		**10 restart**		**50 restart**	
		/Best	Tempo (s)	/Best	Tempo (s)	/Best	Tempo (s)
R00	1594	1,210	0,60	1,169	1,93	1,151	5,01
R01	1600	1,226	0,64	1,229	1,29	1,189	4,85
R02	1576	1,206	0,62	1,150	1,49	1,189	4,83
R03	1631	1,227	0,57	1,139	1,82	1,117	4,40
R04	1611	1,153	0,69	1,135	1,70	1,147	5,18
R05	1528	1,217	0,49	1,143	1,04	1,149	2,27
R06	1651	1,118	0,51	1,085	1,88	1,098	3,74
R07	1653	1,168	0,64	1,105	1,50	1,147	4,28
R08	1607	1,167	0,75	1,099	1,59	1,143	5,09
R09	1598	1,170	0,59	1,136	1,30	1,119	4,00
R10	1702	1,202	0,62	1,185	1,55	1,185	3,55
R11	1575	1,249	0,59	1,153	1,52	1,142	5,11
R12	1652	1,192	0,57	1,129	1,98	1,081	5,53
R13	1617	1,174	0,66	1,187	2,10	1,153	3,25
R14	1611	1,218	0,72	1,192	1,38	1,175	3,67
R15	1608	1,159	0,47	1,149	0,90	1,157	2,52
R16	1725	1,192	0,51	1,151	1,27	1,165	4,52
R17	1627	1,190	0,52	1,127	1,50	1,152	2,15
R18	1671	1,208	0,57	1,173	1,24	1,172	3,91
R19	1635	1,240	0,57	1,171	1,18	1,143	3,97
Media		1,194	0,60	1,150	1,51	1,149	4,09

Tabella 6.11: Risultati test *restart* su istanze da 66 customer

Sono stati eseguiti dei test prendendo solamente una coppia di tour tra quelle fornite dall'euristica CLK, prendendone 4 generate tramite inversione delle precedenti e prendendo invece 50 tour generati casualmente. I risultati e i rispettivi tempi, sono riportati nelle tabelle da 6.13 a 6.15:

- in colonna 1 sono indicate le istanze di test;
- in colonna 2 sono riportate le soluzioni ottenute con l'algoritmo LNS, indicate come **Best**;
- in colonna 3 sono riportati i rapporti tra i risultati ottenuti con 1 tour di partenza e i risultati dell'algoritmo LNS, indicati con /**Best**;
- in colonna 4 sono riportati i tempi di esecuzione del nostro algoritmo con 1 tour di partenza;
- in colonna 5 sono riportati i rapporti tra i risultati ottenuti con 4 tour di partenza e i risultati dell'algoritmo LNS, indicati con /**Best**;

Istanza	**Best**	**4 restart**		**10 restart**		**50 restart**	
		/Best	Tempo (s)	/Best	Tempo (s)	/Best	Tempo (s)
R00	2591	1,225	5,40	1,204	11,88	1,149	34,18
R01	2645	1,241	4,42	1,212	13,36	1,201	37,38
R02	2639	1,219	4,18	1,213	8,28	1,186	24,52
R03	2752	1,205	4,95	1,183	11,27	1,142	31,02
R04	2603	1,230	4,82	1,209	9,85	1,195	32,17
R05	2616	1,255	5,82	1,212	11,75	1,164	39,94
R06	2576	1,225	4,66	1,193	9,64	1,208	34,04
R07	2615	1,217	5,20	1,171	9,26	1,156	31,10
R08	2638	1,195	5,59	1,186	13,63	1,171	41,80
R09	2554	1,185	5,60	1,140	11,59	1,157	34,20
R10	2646	1,216	6,02	1,149	13,55	1,155	20,46
R11	2632	1,243	4,36	1,206	10,20	1,189	37,74
R12	2555	1,266	5,06	1,216	13,94	1,182	39,76
R13	2659	1,187	5,02	1,172	13,44	1,184	34,55
R14	2605	1,210	5,02	1,175	14,05	1,167	37,28
R15	2626	1,221	4,70	1,217	12,41	1,180	33,64
R16	2534	1,280	5,59	1,219	11,29	1,204	32,07
R17	2569	1,208	4,68	1,219	11,22	1,181	30,47
R18	2652	1,199	5,36	1,139	13,43	1,156	31,34
R19	2644	1,213	4,86	1,194	11,75	1,153	50,78
Media		1,222	5,07	1,191	11,79	1,174	34,42

Tabella 6.12: Risultati test *restart* su istanze da 132 customer

- in colonna 6 sono riportati i tempi di esecuzione del nostro algoritmo con 4 tour di partenza;
- in colonna 7 sono riportati i rapporti tra i risultati ottenuti con 50 tour di partenza e i risultati dell'algoritmo LNS, indicati con /**Best**;
- in colonna 8 sono riportati i tempi di esecuzione del nostro algoritmo con 50 tour di partenza.

6.5.1 Istanze di 33 customer

Dai risultati riportati in Tabella 6.13 si può vedere come risolvendo il problema utilizzando come punto di partenza i 4 tour invertiti, si ottengano risultati migliori rispetto a quelli ottenuti con le altre condizioni.

Nonostante con 4 punti di partenza si quadruplichi il lavoro da compiere, i tempi rimangono molto bassi. Questo è dovuto al fatto che parte del tempo è riservato all'allocazione iniziale delle zone di memoria che saranno usate durante la computazione.

Istanza	Best	1 tour		4 tour		50 random	
		/Best	Tempo (s)	/Best	Tempo (s)	/Best	Tempo (s)
R00	1063	1,066	0,05	1,066	0,14	1,295	1,55
R01	1032	1,115	0,04	1,097	0,07	1,305	1,62
R02	1065	1,072	0,06	1,091	0,13	1,214	1,72
R03	1100	1,188	0,06	1,150	0,08	1,293	1,59
R04	1052	1,177	0,06	1,074	0,10	1,211	1,72
R05	1008	1,157	0,08	1,107	0,14	1,346	1,65
R06	1110	1,168	0,06	1,149	0,12	1,256	1,75
R07	1105	1,204	0,04	1,144	0,08	1,307	1,82
R08	1109	1,127	0,04	1,082	0,07	1,230	1,58
R09	1091	1,170	0,06	1,092	0,10	1,275	1,76
R10	1016	1,315	0,05	1,121	0,08	1,325	1,64
R11	1001	1,099	0,06	1,100	0,11	1,208	1,61
R12	1109	1,150	0,08	1,092	0,09	1,300	1,52
R13	1084	1,190	0,06	1,125	0,09	1,292	1,64
R14	1034	1,162	0,06	1,090	0,08	1,480	1,59
R15	1142	1,088	0,06	1,112	0,07	1,245	1,54
R16	1093	1,179	0,05	1,073	0,08	1,294	1,54
R17	1073	1,205	0,05	1,095	0,09	1,238	1,74
R18	1118	1,166	0,06	1,119	0,11	1,216	1,64
R19	1089	1,131	0,05	1,103	0,11	1,255	1,62
Media		1,156	0,06	1,104	0,10	1,279	1,64

Tabella 6.13: Risultati test *tour-restart* su istanze di 33 customer

6.5.2 Istanze di 66 customer

Per quanto riguarda i test con 4 punti di partenza, i risultati sulle istanze da 66 customer (Tabella 6.14) confermano un sensibile miglioramento rispetto ai test con un solo tour, a discapito di tempi quasi raddoppiati. Si possono però notare varie eccezioni: si deve ricordare che i tour sono stati solamente rovesciati e non cambiati, per cui può sempre capitare che una sola coppia di tour molto buoni fornisca soluzioni finali migliori di 4 coppie meno valide.

I risultati dei test con 50 tour casuali rimangono invece molto distanti dalle altre soluzioni, nonostante impieghino molto più tempo per terminare l'esecuzione.

6.5.3 Istanze di 132 customer

Anche i test con istanze da 132 costumer (Tabella 6.15) mostrano come vi sia un buon miglioramento impiegando più tour come punto di partenza dell'algoritmo.

Istanza	Best	1 tour		4 tour		50 random	
		/Best	Tempo (s)	/Best	Tempo (s)	/Best	Tempo (s)
R00	1594	1,205	0,37	1,210	0,60	1,400	2,79
R01	1600	1,356	0,33	1,226	0,64	1,379	2,12
R02	1576	1,218	0,36	1,206	0,62	1,343	2,83
R03	1631	1,305	0,36	1,227	0,57	1,356	3,10
R04	1611	1,223	0,40	1,153	0,69	1,405	2,60
R05	1528	1,259	0,32	1,217	0,49	1,412	2,32
R06	1651	1,225	0,36	1,118	0,51	1,351	2,53
R07	1653	1,179	0,36	1,168	0,64	1,336	2,29
R08	1607	1,142	0,41	1,167	0,75	1,309	2,87
R09	1598	1,220	0,36	1,170	0,59	1,342	2,31
R10	1702	1,256	0,32	1,202	0,62	1,347	2,66
R11	1575	1,223	0,40	1,249	0,59	1,400	2,80
R12	1652	1,153	0,34	1,192	0,57	1,293	2,56
R13	1617	1,222	0,35	1,174	0,66	1,375	2,89
R14	1611	1,265	0,38	1,218	0,72	1,382	2,37
R15	1608	1,218	0,37	1,159	0,47	1,368	2,45
R16	1725	1,191	0,31	1,192	0,51	1,314	3,16
R17	1627	1,307	0,38	1,190	0,52	1,387	2,20
R18	1671	1,282	0,37	1,208	0,57	1,333	2,38
R19	1635	1,300	0,36	1,240	0,57	1,322	2,70
Media		1,237	0,36	1,194	0,60	1,358	2,60

Tabella 6.14: Risultati test *tour-restart* su istanze di 66 customer

Tralasciando il test con 50 tour casuali, possiamo dire che con tutte le istanze i tempi d'esecuzione sono rimasti molto contenuti, portando però a risultati sensibilmente migliori quando sono stati utilizzati un numero maggiore di tour. Per questo motivo la configurazione con 4 coppie di tour di partenza è stata scelta come la migliore dal punto di vista del rapporto risultati/prestazioni e usata per il test *versus*.

Il test con 50 tour casuali ha dimostrato come la scelta dei tour iniziali influisca notevolmente sulla qualità delle soluzioni finali.

6.6 Test capacity

Il test chiamato *capacity* mette in relazione i risultati ottenuti dall'algoritmo vincolando o meno il numero massimo di item presenti negli stack. L'algoritmo può funzionare in entrambi i modi, ma in letteratura il problema del DTSPMS è sempre presentato con questa limitazione del loading plan.

Istanza	**Best**	**1 tour**		**4 tour**		**50 random**	
		/Best	Tempo (s)	/Best	Tempo (s)	/Best	Tempo (s)
R00	2591	1,249	3,37	1,225	5,40	1,344	94,10
R01	2645	1,279	3,08	1,241	4,42	1,310	85,56
R02	2639	1,275	3,10	1,219	4,18	1,354	95,12
R03	2752	1,203	3,90	1,205	4,95	1,339	103,57
R04	2603	1,242	3,34	1,230	4,82	1,291	96,54
R05	2616	1,242	4,08	1,255	5,82	1,377	89,02
R06	2576	1,252	3,30	1,225	4,66	1,371	103,10
R07	2615	1,248	3,82	1,217	5,20	1,341	91,78
R08	2638	1,172	4,17	1,195	5,59	1,332	97,36
R09	2554	1,162	3,62	1,185	5,60	1,282	105,06
R10	2646	1,234	3,94	1,216	6,02	1,341	93,47
R11	2632	1,260	3,36	1,243	4,36	1,333	94,49
R12	2555	1,301	3,66	1,266	5,06	1,384	95,74
R13	2659	1,247	3,43	1,187	5,02	1,354	89,68
R14	2605	1,264	3,84	1,210	5,02	1,357	104,56
R15	2626	1,272	3,68	1,221	4,70	1,387	95,03
R16	2534	1,301	3,26	1,280	5,59	1,388	100,55
R17	2569	1,319	3,15	1,208	4,68	1,355	93,30
R18	2652	1,199	3,48	1,199	5,36	1,309	88,81
R19	2644	1,281	3,20	1,213	4,86	1,382	91,67
Media		1,250	3,54	1,222	5,07	1,347	95,43

Tabella 6.15: Risultati test *tour-restart* su istanze di 132 customer

L'algoritmo è stato eseguito usando 3 stack e i punti di restart generati tramite inversione dei tour iniziali.

L'analisi dei risultati sarà incentrata sui costi delle soluzioni, poiché i tempi saranno molto simili, dato che il vincolo di capacità comporta solamente un overhead minimo.

I risultati sono riportati nelle tabelle da 6.16 a 6.18:

- in colonna 1 sono indicate le istanze di test;
- in colonna 2 sono riportate le soluzioni ottenute con l'algoritmo LNS, indicate come **Best**;
- in colonna 3 sono riportati i rapporti tra i risultati ottenuti con stack illimitati e i risultati dell'algoritmo LNS, indicati con /**Best**;
- in colonna 4 sono riportati i tempi di esecuzione del nostro algoritmo con con stack illimitati;

- in colonna 5 sono riportati i rapporti tra i risultati ottenuti con stack limitati e i risultati dell'algoritmo LNS, indicati con /**Best**;
- in colonna 6 sono riportati i tempi di esecuzione del nostro algoritmo con con stack limitati.

6.6.1 Istanze di 33 customer

I risultati riportati in Tabella 6.16 mostrano come il problema risolto senza vincoli di capacità fornisca soluzioni con costi inferiori. I pochi casi in cui capita il contrario sono dovuti, molto probabilmente, a dei tour iniziali differenti.

Istanza	**Best**	**Stack illimitati**		**Stack limitati**	
		/Best	Tempo (s)	/Best	Tempo (s)
R00	1063	1,046	0,07	1,066	0,14
R01	1032	1,070	0,07	1,097	0,07
R02	1065	1,056	0,12	1,091	0,13
R03	1100	1,160	0,08	1,150	0,08
R04	1052	1,074	0,10	1,074	0,10
R05	1008	1,019	0,13	1,107	0,14
R06	1110	1,093	0,09	1,149	0,12
R07	1105	1,133	0,06	1,144	0,08
R08	1109	1,082	0,09	1,082	0,07
R09	1091	1,065	0,10	1,092	0,10
R10	1016	1,132	0,07	1,121	0,08
R11	1001	1,066	0,09	1,100	0,11
R12	1109	1,079	0,09	1,092	0,09
R13	1084	1,047	0,09	1,125	0,09
R14	1034	1,094	0,08	1,090	0,08
R15	1142	1,084	0,07	1,112	0,07
R16	1093	1,113	0,08	1,073	0,08
R17	1073	1,067	0,08	1,095	0,09
R18	1118	1,109	0,10	1,119	0,11
R19	1089	1,044	0,11	1,103	0,11
Media		1,082	0,09	1,104	0,10

Tabella 6.16: Risultati test *capacity* su istanze di 33 customer

6.6.2 Istanze di 66 customer

Anche per le istanze da 66 customer (risultati in Tabella 6.17) la situazione non cambia: i costi sono mediamente inferiori rispetto alla controparte

più vincolata. Non mancano le eccezioni, ma come detto precedentemente la colpa è da ricercarsi nella diversità dei tour iniziali. Anche il tempo di esecuzione è pressoché identico.

Istanza	**Best**	**Stack illimitati**		**Stack limitati**	
		/Best	Tempo (s)	/Best	Tempo (s)
R00	1594	1,183	0,71	1,210	0,60
R01	1600	1,231	0,72	1,226	0,64
R02	1576	1,142	0,67	1,206	0,62
R03	1631	1,180	0,55	1,227	0,57
R04	1611	1,164	0,71	1,153	0,69
R05	1528	1,143	0,44	1,217	0,49
R06	1651	1,137	0,72	1,118	0,51
R07	1653	1,105	0,65	1,168	0,64
R08	1607	1,104	0,72	1,167	0,75
R09	1598	1,136	0,66	1,170	0,59
R10	1702	1,162	0,56	1,202	0,62
R11	1575	1,187	0,50	1,249	0,59
R12	1652	1,148	0,61	1,192	0,57
R13	1617	1,200	0,48	1,174	0,66
R14	1611	1,175	0,56	1,218	0,72
R15	1608	1,140	0,48	1,159	0,47
R16	1725	1,166	0,53	1,192	0,51
R17	1627	1,148	0,52	1,190	0,52
R18	1671	1,188	0,52	1,208	0,57
R19	1635	1,167	0,53	1,240	0,57
Media		1,160	0,59	1,194	0,60

Tabella 6.17: Risultati test *capacity* su istanze di 66 customer

6.6.3 Istanze di 132 customer

Le istanze con 132 customer non subiscono un andamento diverso dalle precedenti: i risultati, seppur di poco, sono migliori nella versione con stack illimitati che mediamente impiega anche nove centesimi di secondo in meno per effettuare l'elaborazione.

Nonostante in quasi tutte le istanze si abbia un miglioramento, questo rimane minimo e spesso si ottiene un risultato migliore intervenendo sui tour di origine invece che sul rilassamento dei vincoli.

Istanza	**Best**	**Stack illimitati**		**Stack limitati**	
		/Best	Tempo (s)	/Best	Tempo (s)
R00	2591	1,232	5,50	1,225	5,40
R01	2645	1,181	4,57	1,241	4,42
R02	2639	1,205	4,55	1,219	4,18
R03	2752	1,190	4,39	1,205	4,95
R04	2603	1,167	4,96	1,230	4,82
R05	2616	1,188	5,11	1,255	5,82
R06	2576	1,203	4,52	1,225	4,66
R07	2615	1,195	5,58	1,217	5,20
R08	2638	1,177	4,65	1,195	5,59
R09	2554	1,192	4,73	1,185	5,60
R10	2646	1,194	4,48	1,216	6,02
R11	2632	1,211	4,85	1,243	4,36
R12	2555	1,232	5,36	1,266	5,06
R13	2659	1,235	4,98	1,187	5,02
R14	2605	1,167	4,98	1,210	5,02
R15	2626	1,218	4,87	1,221	4,70
R16	2534	1,253	5,95	1,280	5,59
R17	2569	1,226	5,06	1,208	4,68
R18	2652	1,166	4,75	1,199	5,36
R19	2644	1,158	5,73	1,213	4,86
Media		1,200	4,98	1,222	5,07

Tabella 6.18: Risultati test *capacity* su istanze di 132 customer

CAPITOLO 7

Conclusioni

In questa tesi si è mostrato come è possibile implementare un algoritmo efficiente per la risoluzione del problema del DTSPMS.

Il lavoro però non è ancora concluso, infatti è possibile intervenire ancora su alcuni aspetti, in modo da migliorare ciò che è stato fatto finora:

- migliorare le performance modificando le strutture dati e ottimizzando le sezioni di codice critiche, come ad esempio le procedure di programmazione dinamica, vero collo di bottiglia dell'algoritmo;
- sfruttare le architetture multi-processor parallelizzando l'algoritmo, in modo da ottenere soluzioni in tempi più brevi;
- introdurre la generazione di differenti punti di restart basati sui tour che, come si è visto nei test, possono migliorare sensibilmente i risultati;
- migliorare le procedure finora implementate in modo che riescano ad offrire soluzioni migliori, ad esempio modificando la strategia di inserimento degli elementi esclusi da un loading plan parziale;
- implementare procedure di ricerca locale;

- modificare l'algoritmo per risolvere varianti del DTSPMS, in cui, ad esempio, la relazione tra i customer nel grafo di pickup e i customer nel grafo di delivery è di molti-a-molti o in cui gli item hanno dimensioni diverse.

Il risultato del nostro lavoro è da considerarsi anche come un punto di partenza per ulteriori sviluppi. Ciò che è stato ottenuto finora è un risolutore veloce, ma che offre soluzioni ancora distanti dai concorrenti. Per questo motivo uno dei possibili impieghi potrebbe essere come generatore di punti di restart per algoritmi di ricerca locale.

Attualmente il lavoro si sta concentrando su una versione dell'algoritmo sviluppata in C++, linguaggio che offre un numero maggiore di feature e che ci permette di gestire in modo migliore il codice rispetto all'attuale implementazione in C.

Bibliografia

[1] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde documentation. Website http://www.tsp.gatech.edu/concorde/index.html.

[2] B. Bollobás. *Modern Graph theory*, volume 184 of *Graduate Texts in Mathematics*. New York: Springer, 1998.

[3] A. Brandstädt. On improved time bounds for permutation graph problems. In *Graph-Theoretic Concepts in Computer Science*, volume 653 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin / Heidelberg, 1992.

[4] F. Carrabs, J. Cordeau, and G. Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, 19(4):618–632, 2007.

[5] M. Dorigo and T. Stützle. *Ant colony optimization*. New York: Springer, 2004.

[6] Á. Felipe and G. Tirado M. T. Ortuño. Neighborhood structures to solve the double tsp with multiple stacks using local search. Technical report, Department of Statistics and Operations Research, Universidad Complutense de Madrid, 2008.

[7] G. Fuellerer, K. F. Doerner, R. F. Hartl, and M. Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers and Operations Research*, 36(3):655–673, 2009.

[8] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[9] B. Golden and S. Raghavan. *The Vehicle Routing Problem: Latest Advances and New Challenges*. New York: Springer, 2008.

[10] A. Hertz. Anniversary focused issue of computer and operations research on tabu search. *Computers and Operations Research*, 33(9):2447–2448, 2006.

[11] F. S. Hillier and G. J. Lieberman. *Introduction to operations research*. Boston : McGraw-Hill, 2001.

[12] A. Löbel. Mcf documentation. Website http://www.zib.de/Optimization/Software/Mcf.

[13] R. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. An exact method for the double tsp with multiple stacks. Technical report, Department of Management Engineering, Technical University of Denmark and Department of Engineering Science, The University of Auckland, 2009.

[14] A. Moura and J. F. Oliveira. An integrated approach to the vehicle routing and container loading problems. *OR Spectrum*, 31(4):775–800, 2009.

[15] H. L. Petersen. Heuristic solution approaches to the double TSP with multiple stacks. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2006.

[16] H. L. Petersen, C. Archetti, and M. G. Speranza. Exact solutions to the double travelling salesman problem with multiple stacks. Technical report, DTU Transport, Technical University of Denmark and Department of Quantitative Methods, University of Brescia, 2008.

[17] C. Rego and F. Glover. *The Traveling Salesman Problem and Its Variations*. New York: Springer, 2002.

[18] H. Schneider and G. P. Barker. *Matrices and linear algebra*. New York: Dover Publications, 1989.

[19] M. Yannakakis and F. Gavril. The maximum k-colorable subgraph problem for chordal graphs. *Information Processing Letters*, 24(2):133–137, 1987.

[20] E. E. Zachariadis, C D. Tarantilis, and C. T. Kiranoudis. A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 195(3):729–743, June 2009.