Università degli Studi di Milano

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di laurea magistrale in Informatica

Algoritmo genetico per il Multiple Stacks Double Travelling Salesman Problem

Relazione del progetto di Sistemi Intelligenti

Casazza Marco

Matricola: 771605

Indice

1 Introduzione **1**

2 Modello formale **3**

2.1 Casi particolari 6

2.2 Modello di programmazione matematica 6

3 Letteratura **9**

3.1 Algoritmi esatti 9

3.2 Algoritmi euristici 10

3.3 Meta-euristiche per problemi simili 14

4 Algoritmo genetico per il DTSPMS **17**

4.1 Rappresentazione 18

4.2 Generazione della popolazione 19

4.3 Ricombinazione 20

4.4 Mutazione 22

4.5 Selezione naturale 25

4.6 Gestione della popolazione 26

4.7 Ricerca locale 26

5 Analisi sperimentale **29**

6 Conclusioni **35**

Bibliografia **37**

CAPITOLO 1

Introduzione

Il *Double Travelling Salesman Problem with Multiple Stacks (DTSPMS)* è un problema reale di raccolta e distribuzione, in cui sono integrati problemi di *routing* e di *packing*.

Il DTSPMS richiede che un singolo veicolo prelevi della merce da alcuni mittenti residenti in una città (o in generale una regione) e la consegni successivamente ai rispettivi destinatari residenti in una seconda città. L'associazione tra mittenti e destinatari è di tipo uno-a-uno: ogni mittente e destinatario può rispettivamente inviare e ricevere una sola unità di merce.

Per vari motivi, come ad esempio la distanza tra le due città, è necessario che le fasi di raccolta e di consegna delle merci siano effettuate separatamente, visitando prima tutti i mittenti e consegnando in seguito le merci ai destinatari.

Le merci sono disposte su pallet della medesima dimensione e organizzate all'interno del veicolo in un numero arbitrario di file distinte. La loro disposizione, che assume l'aspetto di una griglia, è anche chiamata piano di carico.

Il caricamento e la consegna delle merci sono effettuati in modo LIFO: ogni mittente inserisce la propria merce in testa ad una fila e le consegne devono essere effettuate a destinatari la cui merce è in testa ad una delle file del piano di carico.

Si suppone inoltre che il veicolo viaggi sempre pieno e che ad un trasportatore non sia mai possibile riorganizzare il piano di carico.

L'obbiettivo del problema è di minimizzare il costo necessario ad effettuare tutte le consegne, che può essere misurato come la distanza percorsa dal veicolo, il tempo impiegato, il carburante consumato e così via. Viene richiesto quindi di risolvere un *Travelling Salesman Problem (TSP)*[14] 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.

Il mio obbiettivo è di presentare un algoritmo genetico che risolva questo problema e confrontare i risultati ottenuti con quelli di altre euristiche già presentate in letteratura.

Questo documento è organizzato come segue: nel Capitolo 3 vengono descritti i metodi per risolvere il DTSPMS già presentati in letteratura e alcune meta-euristiche per un problema simile. Nel Capitolo 2 viene descritto un modello formale del problema. Nel Capitolo 4 sono descritte le varie tecniche sperimentate durante l'implementazione dell'algoritmo genetico, i cui risultati saranno analizzati nel Capitolo 5. Il Capitolo 6 conclude il documento con alcune considerazioni sul lavoro svolto.

CAPITOLO 2

Modello formale

È possibile formalizzare il DTSPMS come un problema di ottimizzazione su grafi: rappresentiamo le città come i grafi completi $G^P = (V^P, E^P)$ e $G^D = (V^D, E^D)$, chiamati rispettivamente *grafo di pickup* e *grafo di delivery*. Gli insiemi dei vertici $V^P = \{0, \ldots, N\}$ e $V^D = \{0, \ldots, N\}$ rappresentano rispettivamente gli N mittenti e destinatari (chiamati in generale *customer*) e il deposito di ogni città (identificato con etichetta 0). Ad ogni spigolo del grafo è associato un valore che rappresenta il costo necessario a spostare il veicolo da un customer i ad un customer j: ogni spigolo $[i, j] \in E^P$ è associato ad un costo c^P_{ij}, ogni spigolo $[i, j] \in E^D$ è associato ad un costo c^D_{ij}.

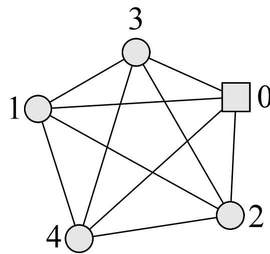

Figura 2.1: Esempio di città per $N = 4$

Sia $I = \{1 \ldots N\}$ l'insieme delle merci (o *item*) e siano $V^P_L = V^P \setminus \{0\}$ e

$V_L^D = V^D \setminus \{0\}$ gli insiemi dei customer che devono rispettivamente inviare e ricevere le suddette merci. Ogni customer $i \in V_L^P$ deve inviare la merce $i \in I$ al destinatario $i \in V_L^D$, vi è quindi una relazione di tipo 1-a-1 (figura 2.2).

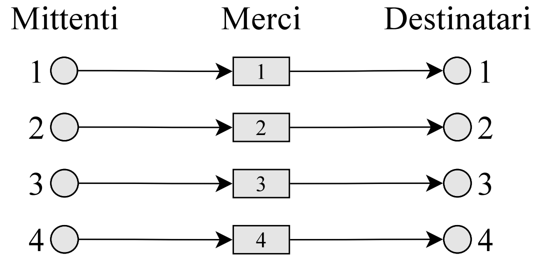

Figura 2.2: Associazione tra mittenti e destinatari

Siano il *tour di pickup* P e il *tour di delivery* D rispettivamente dei *cicli Hamiltoniani*[1] costruiti sul grafo di pickup e sul grafo di delivery che partono e terminano nei rispettivi depositi. Dati due customer i e j appartenenti ad un tour T e stabilito un senso di visita per T, si dice che i precede j nel tour T ($i \prec_T j$) se nella visita lo incontriamo prima di j. Vale inoltre la proprietà transitiva: se $i \prec_T k$ e $k \prec_T j$ allora $i \prec_T j$.

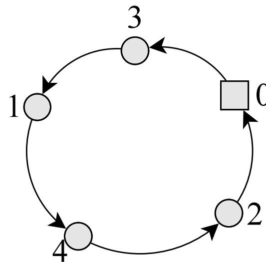

Figura 2.3: Esempio di tour con $N = 4$

Sia L il piano di carico (o *loading plan*) formato da K pile o *stack*. Ogni stack è una sequenza ordinata di interi che definisce l'ordine di caricamento degli item. Si denota con $Stack(k)$ il k-esimo stack del loading plan, con $k = 1 \ldots K$. Si dice che $i \in Stack(k)$ se l'item i è stato inserito nel k-esimo stack. Ogni stack può contenere al massimo C item.

Dati due item $i, j \in Stack(k)$, si dice che i precede j all'interno del loading plan L ($i \prec_L j$) quando j è impilato sopra i nello stack.

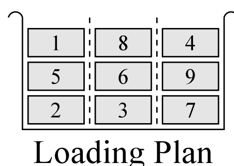

Figura 2.4: Esempio di loading plan con $N = 9$ e $K = 3$

Una soluzione ammissibile S del problema (mostrata in figura 2.5) 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, ovvero per ogni coppia di item i, j se $i \prec_L j$ allora $i \prec_P j$ e $j \prec_D i$.

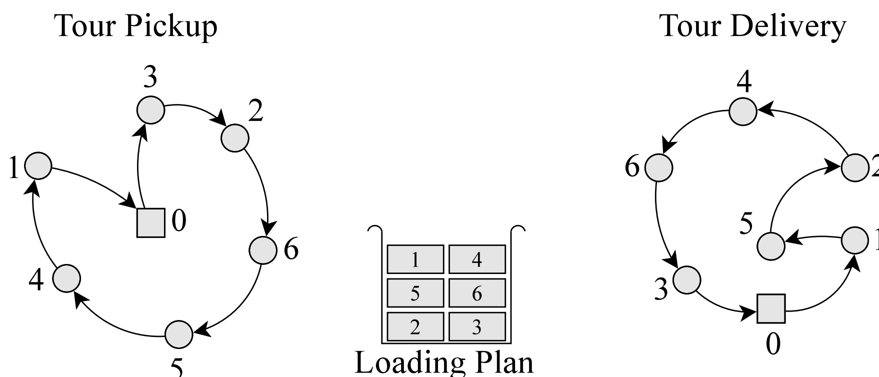

Figura 2.5: Esempio di soluzione ammissibile per $N = 6$ e $K = 2$

L'obbiettivo è quello di ottenere una soluzione ammissibile che minimizzi il costo totale delle visite, dato dalla somma 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 sotto-problema la risoluzione di un TSP, che è noto essere un problema *NP-HARD*.

2.1 Casi particolari

In base al numero di stack disponibili, si possono individuare due casi particolari del problema:

- quando il loading plan è costruito con $K = 1$ (figura 2.6);
- quando il loading plan è costruito con $K \geq N$ (figura 2.7).

Nel primo caso, essendo presente un solo stack, i tour sono l'uno l'inverso dell'altro e l'ordine di visita dipende esclusivamente dall'ordine in cui sono inseriti gli item all'interno dello stack. Il problema può essere risolto come un singolo TSP su un grafo in cui il costo di ogni arco è dato dalla somma dei costi degli archi corrispondenti nel grafo di pickup e di delivery.

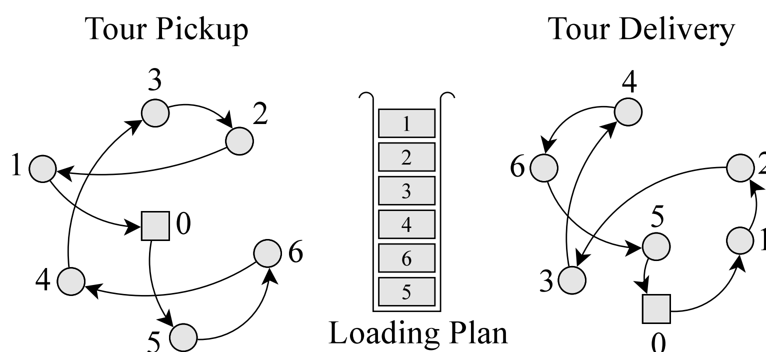

Figura 2.6: Esempio di soluzione con $N = 6$ e $K = 1$

Nel secondo caso invece i tour sono indipendenti tra loro: essendo il numero di stack almeno uguale al numero degli item, ognuno di essi può essere caricato in uno stack differente, eliminando così i sono vincoli di precedenza da rispettare durante la costruzione delle visite. Il problema è risolvibile quindi come due TSP distinti, uno per ogni grafo.

2.2 Modello di programmazione matematica

Il problema può essere descritto come problema di programmazione lineare intera[10] nel modo seguente:

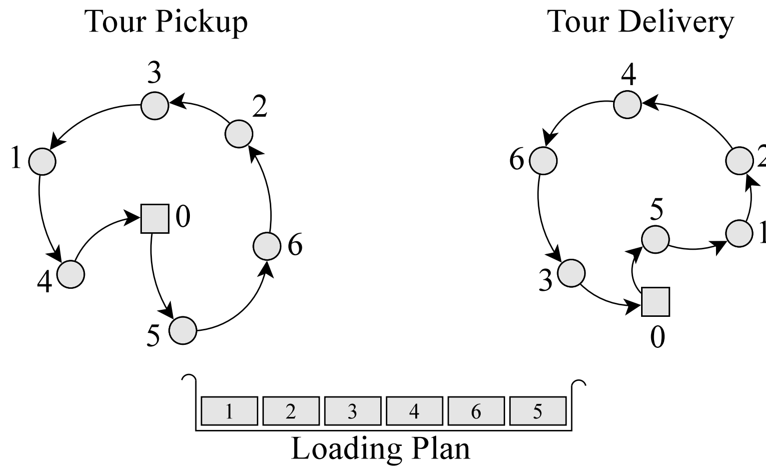

Figura 2.7: Esempio di soluzione con $N = 6$ e $K = 6$

Variabili

$$x_{ij}^T = \begin{cases} 1, & \text{se } [i,j] \in T \\ 0, & \text{altrimenti} \end{cases} \qquad \forall i,j \in V^T, \forall T \in \{P, D\}$$

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

$$z_{ik} = \begin{cases} 1, & \text{se } i \in Stack(k) \\ 0, & \text{altrimenti} \end{cases} \qquad \forall i \in I, \forall k = 1 \ldots K$$

Tutte le variabili del problema sono variabili binarie:

- le variabili x indicano se uno spigolo del grafo appartiene alla soluzione;
- le variabili y definiscono l'ordine in cui due customer devono essere visitati;
- le variabili z associano ogni item allo stack in cui è inserito.

Funzione obbiettivo

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

La funzione obbiettivo richiede di minimizzare la somma dei costi degli spigoli del grafo usati per costruire i tour di visita.

Subject to

$$\sum_{i \in V^T} x_{ij}^T = 1 \qquad \forall j \in V^T, \forall T \in \{P, D\} \tag{2.1}$$

$$\sum_{j \in V^T} x_{ij}^T = 1 \qquad \forall i \in V^T, \forall T \in \{P, D\} \tag{2.2}$$

$$y_{ij}^T + y_{ji}^T = 1 \qquad \forall i, j \in V_L^T, i \neq j, \forall T \in \{P, D\} \tag{2.3}$$

$$y_{ik}^T + y_{kj}^T \leq y_{ij}^T + 1 \qquad \forall i, j, k \in V_L^T, \forall T \in \{P, D\} \tag{2.4}$$

$$x_{ij}^T \leq y_{ij}^T \qquad \forall i, j \in V_L^T, \forall T \in \{P, D\} \tag{2.5}$$

$$y_{ij}^P + z_{ik} + z_{jk} \leq 3 - y_{ij}^D \qquad \forall i, j \in V_L^T, \forall k = 1 \dots K \tag{2.6}$$

$$\sum_{k=1}^{K} z_{ik} = 1 \qquad \forall i \in I \tag{2.7}$$

$$\sum_{i \in I} z_{ik} \leq C \qquad \forall k = 1 \dots K \tag{2.8}$$

$$x, y, z \in \mathbb{B} \tag{2.9}$$

I vincoli 2.1 e 2.2 sono vincoli di conservazione del flusso e impongono che una sola unità di flusso entri ed esca da ogni vertice.

Il vincolo 2.3 assicura che per ogni coppia di vertici distinti di ogni grafo sia stabilita una precedenza. Il vincolo 2.4 definisce la proprietà transitiva della relazione di precedenza tra customer: se un i precedere k e k precede j allora i precede j. Il vincolo 2.5 relaziona la variabile x con la variabile y, imponendo che se uno spigolo appartiene alla soluzione, allora deve esserci una precedenza tra i vertici adiacenti. La presenza della variabile y e dei vincoli descritti fanno si che in questo modello non siano necessari vincoli di eliminazione dei subtour.

Il vincolo 2.6 impedisce i customer associati a item appartenenti allo stesso stack siano visitati nello stesso ordine in entrambi i tour di visita.

Il vincolo 2.7 assicura che un elemento appartenga ad un solo stack, mentre 2.8 è il vincolo di capacità degli stack, i quali possono contenere solo un numero limitato di item.

Il vincolo 2.9 è il vincolo di integrità delle variabili x, y e z.

CAPITOLO 3

Letteratura

Nonostante il DTSPMS sia molto recente, in letteratura sono già stati proposti algoritmi che consentono di risolvere questo problema. Di seguito saranno analizzati i suddetti algoritmi, divisi tra algoritmi che garantiscono l'ottimalità della soluzione e algoritmi basati su euristiche o meta-euristiche

3.1 Algoritmi esatti

La maggior parte degli algoritmi esatti finora proposti utilizzano un approccio *branch-and-cut*[10] su differenti formulazioni del problema.

Il modello delle precedenze (*precedenze model*)[12] consiste nel rimuovere dalla formulazione classica del problema i vincoli di precedenza. Il problema rilassato viene quindi risolto e si controlla l'ammissibilità della soluzione. Se vi sono delle violazioni si aggiungono nel modello i vincoli che le eliminano e si risolve nuovamente il problema, altrimenti la soluzione trovata è ottima.

Una variante del modello delle precedenze usa delle variabili di precedenza relative solamente ad item che appartengono allo stesso stack (*row precedence model*)[12]. L'algoritmo branch-and-cut per questo modello rilassa i vincoli di precedenza e di eliminazione dei subtour. Ad ogni iterazione si risolve il rilassamento del problema e si ricercano prima le inammissibilità relative ai subtour e, in caso non ve ne siano, quelle relative alla precedenza. Per rilevare la presenza di subtour si cercano i tagli di cardinalità minima

che separano il deposito da ogni altro nodo del grafo. Se vi sono dei tagli di cardinalità inferiore a 2 significa che vi è un subtour.

Un secondo metodo sempre basato sul branch-and-cut consiste nel modellare il problema come un problema di flusso (*flow model*)[12]. Anche in questo caso il problema viene rilassato eliminando i vincoli di eliminazione dei subtour e, in aggiunta, vengono eliminati i vincoli che stabiliscono l'ordine di caricamento/consegna delle merci. Per rilevare eventuali violazioni dei primi si utilizza lo stesso procedimento descritto per il *row precedenze model*, mentre per i secondi si percorrono entrambi i tour controllando che i nodi le cui merci appartengono allo stesso stack siano stati visitati nell'ordine corretto.

Un ultimo modello proposto (*TSP with infeasible paths model*)[12] rilassa tutti i vincoli che riguardano la costruzione del loading plan, come ad esempio le precedenze imposte per item appartenenti allo stesso stack, focalizzandosi sulla costruzione dei due tour. Ad ogni iterazione si controlla se i due tour sono ammissibili e se è possibile costruire un loading plan. In caso contrario sono aggiunti tagli per eliminare eventuali subtour e correggere i vincoli non rispettati.

Questi modelli basati su branch-and-cut sono in grado di risolvere istanze di grandezza fino a 18 customer e 3 stack in circa un'ora. L'approccio basato su *TSP with infeasible paths* si è inoltre rivelato essere il migliore tra le alternative.

Recentemente è stato inoltre presentato un nuovo algoritmo esatto[11] che a differenza degli altri non utilizza un approccio branch-and-cut: l'algoritmo ricerca i k-migliori tour di pickup e di delivery e per ogni loro combinazione calcola un loading plan. Le soluzioni ammissibili vengono ordinate in base al costo complessivo e la soluzione di costo inferiore è la soluzione ottima del problema. Nel caso invece non vi siano soluzioni ammissibili si ricercano nuovi K tour da combinare. Le istanze utilizzate per i test e risolte con questo algoritmo si limitano ad una grandezza massima di 12 customer e 3 stack.

3.2 Algoritmi euristici

Le euristiche finora proposte sono molte ma la maggior parte di queste è basata sul paradigma della ricerca locale e fa uso di particolari operatori[13,

6] per esplorare lo spazio delle soluzioni.

Per completezza è necessario descrivere questi operatori prima di analizzare le varie euristiche:

Route Swap (RS): l'operatore scambia la posizione di due customer all'interno di un tour e modifica il secondo nel caso si perda l'ammissibilità. L'appartenenza degli item agli stack non viene modificata, ma se le merci dei customer scelti risiedono nello stesso stack devono essere scambiate a loro volta;

Complete Swap (CS): l'operatore scambia la posizione di due item che appartengono a stack differenti. Per mantenere l'ammissibilità della soluzione devono essere scambiati anche i customer associati nei tour di visita;

In-Stack Swap (ISS): l'operatore è simile all'operatore CS ma effettua lo scambio solamente tra item dello stesso stack;

Reinsertion (R): l'operatore sposta un item all'interno del loading plan, spostando anche i relativi customer all'interno dei tour, rispettando però i vincoli di precedenza;

r-Route Permutation (r-RP): l'operatore inverte l'ordine di visita di r customer consecutivi in entrambi i tour di visita. Gli elementi presi in considerazione devono appartenere tutti a stack differenti, in questo modo si mantiene l'ammissibilità della soluzione ottenuta;

r-Stack Permutation (r-SP): l'operatore fornisce soluzioni ottenute permutando r elementi consecutivi di uno stack. Per mantenere l'ammissibilità è necessario modificare la posizione dei customer associati.

3.2.1 Ricerca locale

Le prime meta-euristiche presentate in [13] sono basate su *Tabu Search (TS)* e *Simulated Annealing (SA)* e utilizzano gli operatori RS e CS per esplorare lo spazio delle soluzioni.

Il *Tabu Search*[9] è una tecnica euristica basata sul paradigma della ricerca locale che evita di rimanere bloccati in un ottimo locale, accettando anche mosse peggioranti. In questo tipo di algoritmi è molto importante mantenere una lista delle soluzioni da evitare (*tabu list*), che evita di cadere

in un loop infinito. L'algoritmo TS proposto in [13] applica iterativamente i due operatori seguendo un pattern predefinito e mantenendo una tabu list di lunghezza costante.

Il *Simulated Annealing*[9] è una tecnica di ricerca locale che, come il TS, cerca di sfuggire ad ottimi locali accettando mosse peggioranti ma, a differenza del TS, le accetta solamente con una certa probabilità, che diminuisce con il passare del tempo. L'algoritmo SA proposto in [13] utilizza gli stessi operatori del TS, ma la scelta è effettuata in modo probabilistico.

In [13] è stata sperimentata la tecnica *Iterated Local Search (ILS)*[9]: questa tecnica utilizza una procedura di ricerca locale che raggiunge velocemente un ottimo locale. La soluzione trovata subisce quindi una perturbazione, fornendo il punto di partenza per l'iterazione successiva. Per il DTSPMS è stata utilizzata una procedura *steepest descent* (o *gradient descent*) e le soluzioni iniziali sono state generate casualmente.

Sempre in [13] è stato implementato un algoritmo *Large scale Neighborhood Search (LNS)*, che iterativamente rimuove un grossa quantità di elementi dalla soluzione e li reinserisce per ottenere una soluzione differente. La qualità delle strategie di rimozione e di inserimento determina la bontà della soluzione ottenuta.

Infine sono stati sviluppati diversi algoritmi che appartengono alla categoria dei *Variable Neighborhood Search (VNS)*[9]: questi algoritmi si basano sul fatto che un ottimo locale raggiunto con un operatore, non è necessariamente un ottimo locale per tutti gli operatori, mentre un ottimo globale lo è per tutti. L'algoritmo raggiunge quindi un ottimo locale con un operatore, dopodiché applica un operatore differente e prosegue la ricerca.

Le varianti implementate per risolvere il DTSPMS sono diverse:

- *Variable Neighborhood Descent (VND)*[9];
- *General Variable Neighborhood Search (GVNS)*[9];
- *Hybridized Variable Neighborhood Search (HVNS)*[6].

L'algoritmo VND applica iterativamente operatori differenti fino ad arrivare ad un ottimo locale comune per tutti gli operatori. Questo algoritmo è il più semplice e rapido ma resta facilmente intrappolato in ottimi locali comuni che però non sono ottimi globali.

L'algoritmo GVNS utilizza l'algoritmo VND come sotto-procedura per arrivare ad un ottimo locale, dopodiché modifica la soluzione e ripete l'algoritmo VND. Gli operatori vengono suddivisi in due set: uno utilizzato con l'algoritmo VND e uno usato per modificare la soluzione trovata in un ottimo locale.

Infine l'algoritmo HVNS sfrutta a sua volta l'algoritmo GVNS come sotto-procedura, ma scegliendo l'operatore da utilizzare in modo casuale. L'HVNS inoltre genera più soluzioni di partenza, e applica a tutte alcuni passi dell'algoritmo. Tra le soluzioni ottenute sceglie la migliore per intensificare la ricerca. L'algoritmo mantiene inoltre una tabu list per evitare di tornare a soluzioni già visitate.

I risultati delle implementazioni proposte in letteratura sono stati confrontati con i risultati ottenuti dopo aver eseguito per diverse ore l'algoritmo LNS (ai quali si farà riferimento come i migliori risultati disponibili in letteratura) su istanze di dimensioni molto maggiori rispetto a quelle risolvibili con un algoritmo esatto. Limitando il tempo d'esecuzione a tre minuti, solamente gli algoritmi VNS e LNS si avvicinano alle migliori soluzioni conosciute, mentre le altre meta-euristiche ottengono risultati peggiori almeno del 10%.

3.2.2 Euristiche costruttive

Fino ad ora in letteratura è stato proposto un solo algoritmo costruttivo: l'*Alternating Routing-Loading algorithm (ARL)*[2]. L'algoritmo cerca di ottenere l'ammissibilità della soluzione partendo da tour buoni ma costruiti in modo indipendente l'uno dall'altro. Ad ogni iterazione l'algoritmo crea dai tour di visita un loading plan che include il numero massimo di item. Se il loading plan è completo allora i tour rispettano tutti i vincoli di precedenza e l'algoritmo può terminare, altrimenti il loading plan si dice parziale e l'algoritmo costruite due soluzioni: una ammissibile e una non ammissibile. La prima è ottenuta rispettando i vincoli di precedenza mentre la seconda non rispetta tali vincoli e fornisce in generale tour di costo inferiore. Tali tour forniscono la base dell'iterazione successiva. L'algoritmo termina quando in due iterazioni successive si ottiene un loading plan parziale che include lo stesso numero di item.

Questo algoritmo fornisce in pochi secondi soluzioni mediamente migliori delle soluzioni ottenute con altre meta-euristiche come il SA e il TS.

3.3 Meta-euristiche per problemi simili

Il DTSPMS non è l'unico problema in cui routing e packing sono integrati. In letteratura sono già stati affrontati problemi simili, come ad esempio il *2-dimensional Loading Capacitated Vehicle Routing Problem (2L-CVRP)*: in questo problema un insieme di K veicoli, il cui piano di carico è visto come un piano bidimensionale, devono consegnare delle merci di dimensioni differenti ai rispettivi destinatari. L'obbiettivo è quello di ottenere il tour di consegna e il piano di carico migliori per ogni veicolo in modo da minimizzare i costi di trasporto delle merci.

Esistono diverse versioni di questo problema, che si distinguono:

- per la possibilità di ruotare le merci (*non-oriented loading*) o di non poterlo fare, dovendo rispettare un determinato orientamento (*oriented loading*);
- per la possibilità di spostare le merci (*unrestricted loading*) o di non poterlo fare, costringendo alla visita dei destinatari le cui merci possono essere scaricate senza spostare merci di altri destinatari (*sequential loading*).

Sono state sviluppate alcune meta-euristiche basate su *Ant Colony Optimization (ACO)*[9] e *Guided Tabu Search (GTS)*.

L'*Ant Colony Optimization* è una meta-euristica probabilistica che trae ispirazione dalla biologia, in particolare dal mondo delle formiche. Questa tecnica si basa sulla comunicazione indiretta di una colonia di agenti, chiamati anche *ants* (formiche), tramite una traccia artificiale di feromoni che costituisce l'informazione usata dagli agenti per costruire la soluzione.

In [7] è stato implementato un algoritmo basato sull'algoritmo *Saving-Based ACO*[16, 15] per il *Vehicle Routing Problem (VRP)*. Questo algoritmo consiste in tre passi fondamentali:

1. generazione delle soluzioni ottenute con una popolazione di agenti;
2. applicazione di un algoritmo di ricerca locale per migliorare le soluzioni;
3. aggiornamento delle tracce di feromone.

Partendo da questa base è stato implementato un algoritmo per il 2L-CVRP che ad ogni iterazione aggiunge il calcolo di loading plan ammissibili, i quali

guidano gli agenti nelle decisioni e nella costruzione delle soluzioni. Inoltre, per limitare il numero dei veicoli utilizzati, vengono penalizzate le soluzioni che utilizzano più di K veicoli.

Il *Guided Tabu Search* è invece una meta-euristica basata sul *Tabu Search* in cui la funzione obbiettivo è alterata in modo da diversificare lo spazio delle soluzioni visitate e penalizzare determinate scelte come nella meta-euristica *Guided Local Search*[9]. L'algoritmo proposto in [19] riesce a risolvere il problema sia in condizione *sequential loading* sia *unrestricted loading*. Le merci devono avere però un orientamento stabilito (*oriented loading*).

L'algoritmo ottiene una soluzione di partenza con un'apposita euristica e la migliora iterativamente con operatori generici per il VRP e TSP definiti in [18] e [3]. Una soluzione viene però accettata solo se permette di costruire un loading plan ammissibile e non è presente nella *tabu list*. Inoltre ad ogni iterazione gli archi più costosi della soluzione vengono penalizzati in modo da evitare di sceglierli successivamente, conducendo l'algoritmo a valutare soluzioni più promettenti.

Da test effettuati su entrambi gli algoritmi[4], l'ACO risulta essere mediamente migliore del GTS, sia per istanze di piccole dimensioni che per istanze particolarmente difficili. Sono state inoltre considerate istanze 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 fino a 76 customer in poco più di un minuto.

CAPITOLO 4

Algoritmo genetico per il DTSPMS

Un algoritmo genetico è un metodo euristico di ricerca e ottimizzazione che si ispira alla teoria evoluzionistica. L'algoritmo simula il processo evolutivo di una popolazione facendo ricombinare tra loro i vari individui, applicando una mutazione casuale per alcuni di essi ed eliminando quelli che non posseggono le caratteristiche necessarie per la sopravvivenza. Questo processo lo si può applicare a problemi di ottimizzazione in cui ogni individuo rappresenta una soluzione del problema.

Per ogni algoritmo genetico è necessario scegliere:

- il modo in cui saranno rappresentati gli individui;
- il metodo con cui viene generata la popolazione iniziale;
- come effettuare la ricombinazione;
- come effettuare la mutazione;
- una politica per la selezione degli individui da eliminare;
- una politica con cui gestire la numerosità della popolazione.

Inoltre è possibile aggiungere procedure di ricerca locale per migliorare la qualità degli individui della popolazione.

Le tecniche utilizzate possono essere distinte tra tecniche generiche, applicabili ad un'ampia gamma di problemi, e tecniche specifiche per un particolare problema.

Di seguito sono riportate le tecniche studiate e sviluppate per la realizzazione di un algoritmo genetico per la risoluzione del DTSPMS.

4.1 Rappresentazione

Per il DTSPMS ho deciso che ogni individuo della popolazione rappresenti il loading plan di una possibile soluzione del problema. Il cromosoma che identifica un individuo si può rappresentare come una stringa permutazione degli interi da 1 a N, ovvero una permutazione di tutti gli item del loading plan.

È possibile effettuare il mapping tra loading plan e cromosoma in diversi modi. Nel mio caso ho studiato due possibili tecniche:

mapping per stack: il cromosoma è dato dalla concatenazione degli stack del loading plan (figura 4.1);

mapping per righe: il cromosoma viene costruito concatenando le righe del loading plan (figura 4.2).

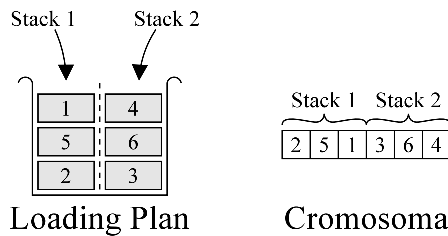

Figura 4.1: Mapping per stack del loading plan nel cromosoma

È inoltre sempre possibile risalire al loading plan dato il cromosoma:

mapping per stack: si divide il cromosoma in K blocchi di C elementi, ogni blocco è uno stack del loading plan;

mapping per righe: si divide il cromosoma in C blocchi di K elementi, ogni blocco è una riga del loading plan.

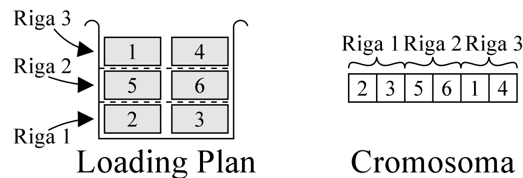

Figura 4.2: Mapping per righe del loading plan nel cromosoma

4.1.1 Funzione di fitness

La *fitness* di un individuo indica il suo livello di qualità all'interno della popolazione, ovvero quanto l'individuo è migliore rispetto agli altri. In generale gli individui con fitness migliore corrispondo a soluzioni migliori.

Nel mio caso la fitness di un individuo corrisponde al costo di una soluzione, quindi gli individui migliori avranno una fitness bassa. Il costo della soluzione non è però direttamente riconducibile al cromosoma o al loading plan ma dipende dal modo in cui sono costruiti i tour. Sono quindi necessarie apposite procedure per calcolarli.

In [2] è già stato mostrato come sia possibile risolvere questo problema in modo ottimo con algoritmi di programmazione dinamica, ma è anche possibile implementare apposite euristiche in modo da rendere l'operazione più rapida. L'algoritmo utilizzato nell'implementazione è derivato dall'algoritmo *Random Insertion*[8] per il TSP: l'algoritmo costruisce iterativamente un tour inserendo ad ogni iterazione un nodo scelto in modo casuale nella posizione che minimizza il costo aggiunto. L'algoritmo è stato modificato in modo che sia sempre scelto un item in cima allo stack e che i customer corrispondenti siano inseriti in modo corretto all'interno dei tour, rispettando i vincoli di precedenza.

4.2 Generazione della popolazione

È possibile generare gli individui della popolazione iniziale con due metodi differenti:

- in modo casuale;
- risolvendo il problema con un'euristica.

Generare individui in modo casuale è molto semplice in quanto l'operazione consiste nel creare stringhe di N interi senza ripetizioni (ovvero il cromosoma) e da queste ottenere una soluzione con i metodi già descritti.

La generazione con euristica è invece più complessa: nonostante esistano già delle buone euristiche, ho preferito implementare un algoritmo specifico per contenere i tempi d'esecuzione. L'algoritmo progettato permette di ottenere un loading plan iniziale risolvendo un *Capacitated Vehicle Routing Problem (CVRP)*[17] con K veicoli di capacità C su un grafo $\hat{G} = (\hat{V}, \hat{E})$ dove l'insieme dei vertici è definito come $\hat{V} = \{0, \dots, N\}$, dove 0 è il deposito, e l'insieme degli spigoli come $\hat{E} = \{[i,j] \mid [i,j] \in E^P \wedge [i,j] \in E^D\}$. Per ogni spigolo di $\hat{G}$ il costo è dato dalla somma dei costi del medesimo spigolo nei grafi di pickup e delivery, ovvero $\hat{c}_{i,j} = c^P_{i,j} + c^D_{i,j}$.

L'algoritmo sceglie casualmente ad ogni iterazione un nodo da visitare e lo associa ad un veicolo in modo da minimizzare il costo di visita, senza però eccedere il limite di capacità C. In questo modo al termine dell'algoritmo si ottengono K sequenze di interi senza ripetizione (ad eccezione del deposito) che possono essere utilizzate come stack del loading plan. Test preliminari hanno evidenziato che i tour costruiti in questo modo sono mediamente migliori dei tour costruiti in modo casuale.

4.3 Ricombinazione

La ricombinazione o *crossover*, è il processo che permette di generare nuovi individui (figli) da coppie di individui che fanno parte della popolazione (genitori). I figli hanno alcune caratteristiche in comune con i genitori e, in generale, questa operazione consente di migliorare la qualità degli individui della popolazione.

Nella mia implementazione ho utilizzato due tecniche differenti:

- *order crossover*[5];
- *single point crossover* con euristica per risolvere i conflitti.

L'*order crossover* è una tecnica di ricombinazione utilizzata generalmente quando il cromosoma è la permutazione di un insieme di elementi. Questa tecnica ricopia la parte compresa tra due punti di taglio da un genitore e ottiene dall'altro la parte mancante evitando di inserire due volte lo steso gene.

L'algoritmo funziona in questo modo:

1. vengono scelti casualmente due punti di taglio t_1 e t_2, con t_1 che precede t_2. La parte del genitore G_1 compresa tra t_1 e t_2 è ricopiata nella stessa posizione all'interno del cromosoma figlio;
2. i geni rimanenti vengono copiati dal genitore G_2 partendo dal taglio t_2 e seguendo l'ordine in cui appaiono. I geni già presenti vengono saltati. Raggiunta la fine del cromosoma si ricomincia dal primo gene, come se il cromosoma fosse una struttura circolare;
3. si genera un secondo figlio invertendo i genitori.

Un esempio dell'algoritmo è riportato nelle figure 4.3 e 4.4.

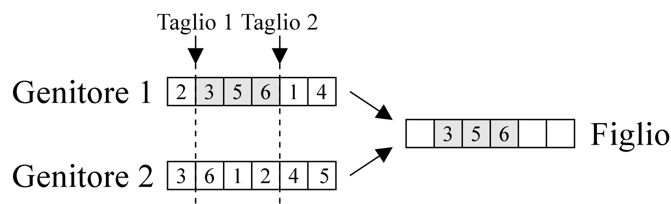

Figura 4.3: Passo 1 dell'*order crossover*: copia della parte di G_1 compresa tra t_1 e t_2

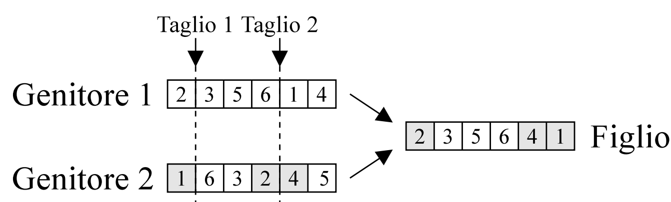

Figura 4.4: Passo 2 dell'*order crossover*: inserimento geni mancanti partendo da t_2

Il secondo tipo di ricombinazione implementata si basa sulla classica ricombinazione con singolo punto di taglio:

1. è scelto casualmente un punto di taglio t;

2. il primo figlio viene composto con la parte che precede t del genitore G_1 e con la parte che segue t del genitore G_2;
3. la stessa operazione ma con genitori invertiti viene ripetuta per il secondo figlio.

Questo però non è sufficiente a garantire che un cromosoma sia una permutazione di tutti i geni: alcuni geni potrebbero essere ripetuti 2 volte, mentre altri potrebbero essere assenti.

Per risolvere questo problema è necessario:

- rimuovere dal cromosoma i geni ripetuti;
- costruire dal cromosoma incompleto un loading plan parziale;
- costruire i tour di visita parziali validi per il loading plan parziale;
- completare la soluzione parziale con gli item mancanti inseriti uno dopo l'altro nella posizione del loading plan che permette di minimizzare il costo di visita nei tour (figura 4.5).

4.3.1 Scelta dei genitori

La scelta degli individui per la fase di ricombinazione è effettuata in modo casuale, privilegiando però gli individui con una migliore funzione di fitness:

1. si estrae dalla popolazione un individuo in modo casuale;
2. si genera un numero casuale compreso tra i valori delle fitness del miglior individuo e del peggiore;
3. se il valore della funzione di fitness dell'individuo selezionato è inferiore al numero generato allora l'individuo può partecipare alla ricombinazione.

4.4 Mutazione

La mutazione è un processo importante affinché gli individui maturino verso una nuova forma. In questo progetto sono stati sviluppati 4 differenti tipi di mutazione:

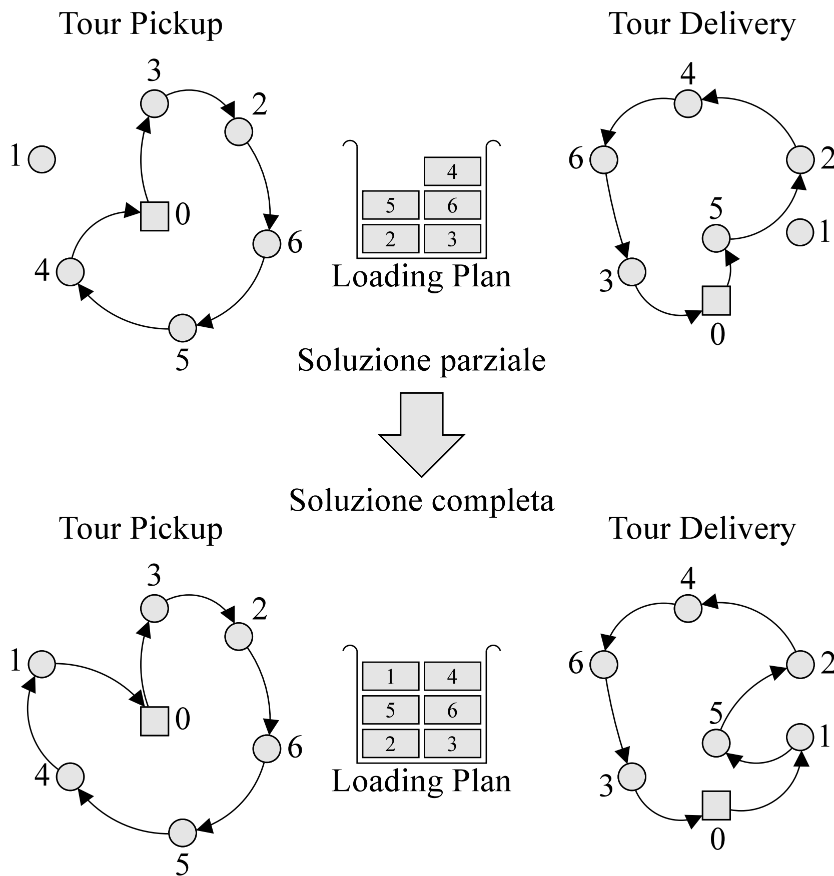

Figura 4.5: Completamento della soluzione con l'inserimento degli item mancanti

- *swap-mutation*[5];
- *insert-mutation*[5];
- *inversion-mutation*[5];
- con operatore *Total-Swap (TS)*.

Le prime tre tecniche operano sul cromosoma dell'individuo e, dopo aver effettuato la mutazione, ritrasformano il cromosoma in un loading plan ricavando anche dei nuovi tour di visita.

La tecnica *swap-mutation* (figura 4.6) scambia la posizione di due geni del cromosoma scelti in modo casuale.

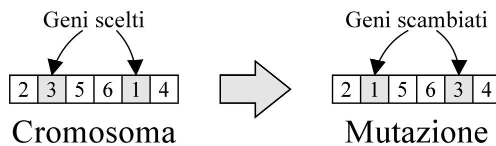

Figura 4.6: Esempio di *swap-mutation* dei geni 1 e 3

L'*insert-mutation* (figura 4.7) sposta un gene in una posizione diversa scelta in modo casuale, spostando (a destra o a sinistra) i geni compresi tra la vecchia e la nuova posizione.

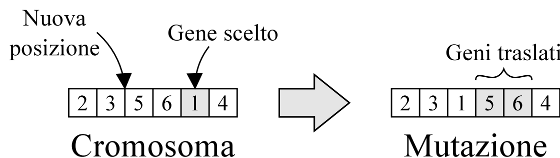

Figura 4.7: Esempio di *inserti-mutation* del gene 1 in posizione 2

L'*inversion-mutation* (figura 4.8) sceglie invece un insieme di r geni contigui e inverte le loro posizioni.

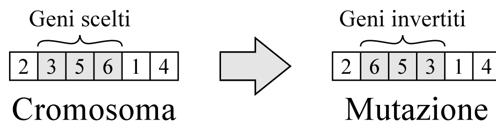

Figura 4.8: Esempio di *inversion-mutation* con $r = 3$

L'ultima tecnica è una mutazione specifica per il problema ed utilizza un'operatore chiamato *Total-Swap (TS)*, che può essere definito come l'unione degli insiemi delle soluzioni ottenibili applicando gli operatori *Complete-Swap (CS)* e *In-Stack-Swap (ISS)*, già descritti nella sezione 3.2. Con l'operatore TS si ottiene quindi l'insieme delle soluzioni raggiungibili scambiando due item qualsiasi del loading plan (figura 4.9). La soluzione meno costosa tra quelle ottenute è scelta come risultato della mutazione solo se consente di migliorare la fitness dell'individuo. In caso contrario l'individuo non subisce la mutazione e ciò significa che ci si trova in un minimo locale per l'operatore TS.

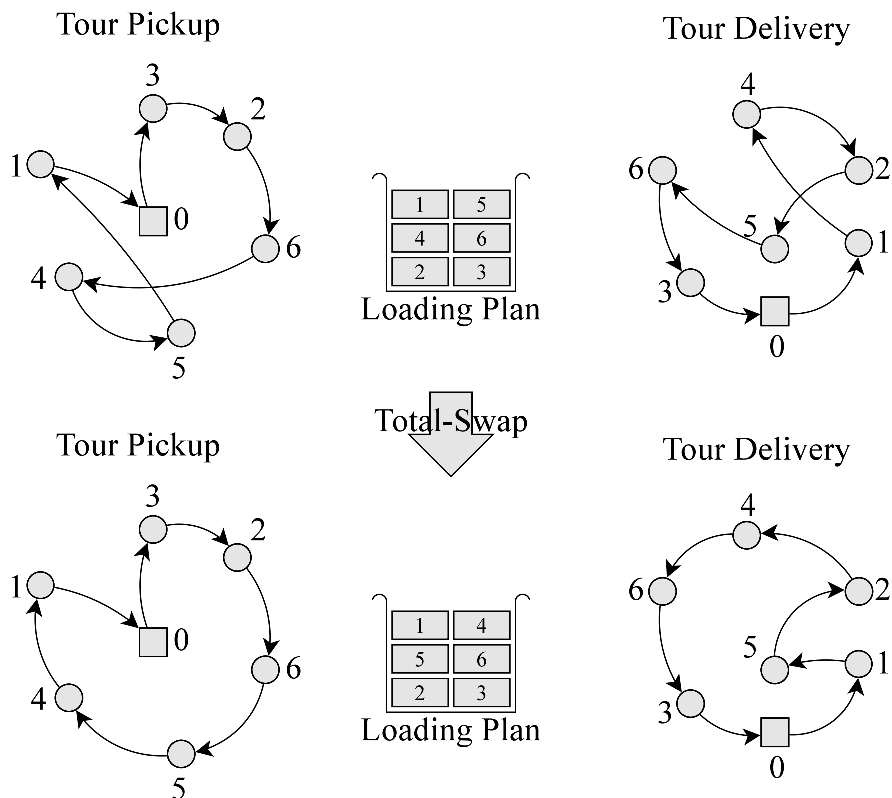

Figura 4.9: Esempio di mutazione con operatore TS: scambio degli item e customer 4 e 5

È da notare che sebbene quest'ultimo operatore assomigli molto alla mutazione *swap-mutation*, le soluzioni generate possono essere molto diverse: entrambi hanno l'effetto di scambiare due item nel loading plan, ma mentre con la tecnica *swap-mutation* lo scambio è causale, con l'operatore TS è invece finalizzato a migliorare la soluzione. Inoltre con la prima tecnica si ricalcolano entrambi i tour di visita, cosa che non avviene per l'operatore TS.

4.5 Selezione naturale

La selezione naturale identifica all'interno della popolazione gli individui meno promettenti e li elimina, lasciando così solo quelli con maggiori qualità.

La politica di selezione implementata è molto simile alla politica di selezione dei genitori per la ricombinazione:

1. per ogni individuo si genera un numero casuale compreso tra i valori di fitness del miglior individuo e del peggiore;
2. se il valore della funzione di fitness dell'individuo è inferiore al numero generato allora sopravvive, altrimenti viene eliminato dalla popolazione.

Questa procedura fa si che gli individui con fitness migliore abbiano più probabilità di sopravvivere.

4.6 Gestione della popolazione

La popolazione non è mantenuta costante durante le ere ma varia in base all'effetto ottenuto dalla ricombinazione e dalla selezione naturale: la prima genera sempre un numero fisso di figli che non sostituiscono però i genitori, mentre la seconda elimina un numero casuale di individui.

Questo può portare a diversi problemi:

- aumento spropositato della popolazione dovuto alla scarsa varianza degli individui;
- riduzione della popolazione a pochi individui nel caso la diversità tra i migliori individui e il resto della popolazione sia elevata.

Per limitare questi effetti indesiderati, la ricombinazione viene sospesa quando la popolazione diventa troppo numerosa (lasciando che la mutazione aumenti la varianza degli individui) mentre la selezione è sospesa quando la popolazione è molto ridotta.

4.7 Ricerca locale

All'algoritmo è stata aggiunta anche una procedura di ricerca locale ottenuta modificando l'operatore *Reinsertion (R)* di cui si è parlato nella sezione 3.2. L'operatore estrae un item qualsiasi dal loading plan e dai tour di visita, reinserendolo nella posizione che minimizza il costo. Sorgono però problemi quando un item è inserito in uno stack che raggiunge la capacità massima ed

è per questo motivo che, per mantenere la procedura efficiente, è stato imposto il reinserimento all'interno del medesimo stack da cui un item è stato estratto.Test preliminari hanno evidenziato come questa procedura migliori sensibilmente la qualità delle soluzioni fornite, anche se la sua potenza è inferiore a quella dell'operatore originale.

CAPITOLO 5

Analisi sperimentale

Le tecniche descritte nel capitolo precedente sono state implementate in un programma scritto in C++. Questo è stato successivamente testato su un calcolatore con processore di frequenza 1833MHz.

Le istanze utilizzate per i test sono istanze disponibili online e usate dagli altri algoritmi proposti in letteratura. Queste istanze rappresentano le città indicando la posizione dei customer all'interno di un piano di dimensione 100×100 in cui il deposito si trova in posizione $(50; 50)$. Le istanze si differenziano per la numerosità dei nodi: per i test sono state usate istanze con 33, 66 e 132 customer. Il numero di stack utilizzati per costruire il loading plan è stato fissato a 3.

I test sono stati effettuati con la seguente configurazione dell'algoritmo:

- gli individui sono stati rappresentati con il mapping per righe in quanto questa rappresentazione permette di sfruttare meglio la procedura di ricerca locale;

- il numero degli individui della popolazione iniziale è stato fissato a 100: il 25% di questi è stato generato con l'euristica presentata, mentre i restanti sono stati generati in modo casuale;

- la tecnica scelta per la ricombinazione è la *single point crossover* con euristica per risolvere i conflitti;

- ogni fase di ricombinazione aumenta del 50% la numerosità della popolazione e almeno il 25% dei nuovi individui sono figli del miglior individuo della popolazione;
- ad ogni era ogni individuo ha una probabilità del 50% di mutare;
- la mutazione è effettuata con l'operatore TS. Se questo non migliora la soluzione viene scelto casualmente una tecnica tra *swap-mutation*, *insert-mutation* e *inversion-mutation*;
- la soglia massima della popolazione è di 500 individui, quella minima di 50;
- ad ogni era viene eseguita la procedura di ricerca locale per ogni individuo che sia stato modificato nell'era precedente (tramite mutazione o ricerca locale);

Per poter confrontare i risultati con quelli ottenuti da altri algoritmi vengono considerate due tipi di soluzioni: la migliore soluzione ottenuta dopo $10s$ e la migliore soluzione ottenuta dopo $180s$.

I risultati sono riportati nelle tabelle da 5.1 a 5.3:

- in colonna 1 sono riportate le istanze utilizzate nei test;
- in colonna 2 sono riportati i valori delle migliori soluzioni conosciute finora in letteratura, ottenute con una algoritmo LNS;
- in colonna 3 sono riportati i rapporti tra i valori delle soluzioni ottenute con l'algoritmo HVNS in $10s$ e i valori in colonna 2;
- in colonna 4 sono riportati i rapporti tra i valori delle soluzioni ottenute con il mio algoritmo genetico in $10s$ e i valori in colonna 2;
- in colonna 5 sono riportati i rapporti tra i valori delle soluzioni ottenute con l'algoritmo HVNS in $180s$ e i valori in colonna 2;
- in colonna 6 sono riportati i rapporti tra i valori delle soluzioni ottenute con il mio algoritmo genetico in $180s$ e i valori in colonna 2.

5.0.1 Analisi dei risultati

Analizzando i risultati ottenuti con istanze da 33 customer (tabella 5.1) si può notare come le soluzioni dell'algoritmo siano abbastanza distanti dal concorrente HVNS, nonostante la media dei valori sia molto buona. Questo è dovuto soprattutto ad alcuni casi in cui l'algoritmo ha fornito soluzioni molto buone (ad esempio nelle istanze R04, R11 e R12).

Dalla tabella si può anche notare come lasciando in esecuzione l'algoritmo per molto tempo non si abbia un miglioramento sensibile. Molto probabilmente questo significa che l'algoritmo è rimasto intrappolato in un ottimo locale.

Istanza	**Best**	**10s**		**180s**	
		HVNS	**Genetic**	**HVNS**	**Genetic**
R00	1063	1,008	1,056	1,000	1,056
R01	1032	1,008	1,040	1,000	1,040
R02	1065	1,008	1,054	1,000	1,054
R03	1100	1,000	1,039	1,000	1,030
R04	1052	1,005	1,007	1,000	1,007
R05	1008	1,022	1,043	1,000	1,043
R06	1110	1,000	1,023	1,000	1,023
R07	1105	1,004	1,031	1,000	1,031
R08	1109	1,000	1,033	1,000	1,033
R09	1091	1,000	1,068	1,000	1,034
R10	1016	1,000	1,065	1,000	1,065
R11	1001	1,000	1,008	1,000	1,008
R12	1109	1,002	1,008	1,000	1,008
R13	1084	1,000	1,025	1,000	1,025
R14	1034	1,017	1,051	1,000	1,051
R15	1142	1,014	1,012	1,000	1,012
R16	1093	1,000	1,026	1,000	1,026
R17	1073	1,009	1,029	1,000	1,023
R18	1118	1,028	1,087	1,007	1,045
R19	1089	1,006	1,048	1,002	1,048
Media		1,008	1,038	1,000	1,033

Tabella 5.1: Risultati test con istanze da 33 customer

I risultati ottenuti con istanze da 66 customer (tabella 5.2) invece mostrano come l'algoritmo sia stato in grado di sfruttare il tempo aggiuntivo per migliorare notevolmente la qualità delle soluzioni. La media dei rapporti è peggiorata rispetto ai risultati precedenti ma questo è dovuto all'aumento della dimensione delle istanze.

Istanza	**Best**	**10s**		**180s**	
		HVNS	**Genetic**	**HVNS**	**Genetic**
R00	1594	1,038	1,205	1,031	1,139
R01	1600	1,064	1,241	1,043	1,154
R02	1576	1,077	1,220	1,062	1,148
R03	1631	1,059	1,208	1,025	1,095
R04	1611	1,077	1,212	1,029	1,097
R05	1528	1,069	1,180	1,033	1,117
R06	1651	1,105	1,195	1,036	1,118
R07	1653	1,064	1,206	1,010	1,081
R08	1607	1,111	1,230	1,034	1,111
R09	1598	1,086	1,201	1,031	1,101
R10	1702	1,078	1,199	1,039	1,085
R11	1575	1,067	1,219	1,053	1,085
R12	1652	1,060	1,193	1,022	1,085
R13	1617	1,087	1,255	1,025	1,141
R14	1611	1,066	1,209	1,014	1,063
R15	1608	1,065	1,193	1,019	1,138
R16	1725	1,082	1,215	1,026	1,066
R17	1627	1,075	1,251	1,051	1,096
R18	1671	1,065	1,218	1,023	1,133
R19	1635	1,052	1,226	1,029	1,097
Media		1,072	1,214	1,032	1,108

Tabella 5.2: Risultati test con istanze da 66 customer

Infine, come si può notare dai risultati in tabella 5.3, anche con 132 customer il tempo aggiuntivo permette di migliorare sensibilmente la qualità delle soluzioni, ma non abbastanza da riuscire ad ottenere buone soluzioni. I risultati infatti si discostano molto dai valori delle migliori soluzioni disponibili, fino ad essere oltre il 31% peggiori.

Istanza	**Best**	**10s**		**180s**	
		HVNS	**Genetic**	**HVNS**	**Genetic**
R00	2591	1,157	1,362	1,066	1,270
R01	2645	1,167	1,387	1,054	1,268
R02	2639	1,139	1,371	1,052	1,299
R03	2752	1,124	1,321	1,032	1,203
R04	2603	1,131	1,381	1,046	1,290
R05	2616	1,158	1,379	1,057	1,285
R06	2576	1,160	1,335	1,071	1,313
R07	2615	1,147	1,368	1,075	1,243
R08	2638	1,143	1,386	1,053	1,229
R09	2554	1,136	1,315	1,035	1,255
R10	2646	1,190	1,372	1,064	1,303
R11	2632	1,133	1,360	1,046	1,259
R12	2555	1,185	1,411	1,068	1,272
R13	2659	1,157	1,359	1,050	1,279
R14	2605	1,140	1,390	1,037	1,253
R15	2626	1,185	1,366	1,053	1,272
R16	2534	1,160	1,361	1,067	1,309
R17	2569	1,142	1,334	1,046	1,292
R18	2652	1,151	1,336	1,035	1,263
R19	2644	1,160	1,363	1,042	1,281
Media		1,153	1,363	1,053	1,272

Tabella 5.3: Risultati test con istanze da 132 customer

CAPITOLO 6

Conclusioni

In questo documento ho mostrato alcune tecniche per implementare un algoritmo genetico che risolva il problema del DTSPMS. In alcuni casi i risultati ottenuti sono molto incoraggianti e l'attuale implementazione può essere un punto di partenza per la realizzazione di un algoritmo genetico più complesso. L'algoritmo può essere migliorato sotto diversi aspetti:

- è possibile utilizzare degli algoritmi più sofisticati per risolvere alcuni sotto-problemi, come ad esempio la procedura che permette di ottenere i tour di visita da un loading plan;

- è possibile migliorare le euristiche utilizzate per la generazione delle soluzioni iniziali, ad esempio utilizzando l'euristica costruttiva ARL di cui si è parlato nel capitolo 3;

- è necessario trovare una soluzione ai problemi che si presentano con l'attuale politica di selezione naturale, ad esempio adattandola alla distribuzione dei valori di fitness degli individui della popolazione;

- è possibile introdurre un calcolo della diversità tra gli individui in modo da favorire la ricombinazione tra individui diversi e salvaguardare gli individui con tratti rari nella fase di selezione.

Questi sono solo alcuni esempi di ciò che può essere migliorato. Attualmente sto cercando di integrare l'euristica ARL all'interno dell'algoritmo in modo da avere fin dalla partenza alcuni individui molto promettenti.

Bibliografia

[1] Bela Bollobas. *Modern Graph Theory*. Springer, corrected edition, July 1998.

[2] M. Casazza, A. Ceselli, and M. Nunkesser. Efficient algorithms for the double tsp with multiple stacks. In *proc. of CTW 2009, Paris*, June 2009.

[3] G. A. Croes. A Method for Solving Traveling-Salesman Problems. *OPERATIONS RESEARCH*, 6(6):791–812, 1958.

[4] Karl F. Doerner, Guenther Fuellerer, Richard F. Hartl, Manfred Gronalt, and Manuel Iori. Metaheuristics for the vehicle routing problem with loading constraints. *Networks*, 49(4):294–307, 2007.

[5] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer, October 2008.

[6] Ángel Felipe, M. Teresa Ortuño, and Gregorio Tirado. The double traveling salesman problem with multiple stacks: A variable neighborhood search approach. *Computers & Operations Research*, 36(11), 2009.

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

[8] Fred Glover, Gregory Gutin, Anders Yeo, and Alexey Zverovich. Construction heuristics for the asymmetric tsp. *European Journal of Operational Research*, 129(3), 2001.

[9] Fred W. Glover and Gary A. Kochenberger. *Handbook of Metaheuristics*, volume 114 of *International Series in Operations Research & Management Science*. Springer, January 2003.

[10] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 8th edition, 2005.

[11] 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.

[12] 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.

[13] Hanne L. Petersen and Oli B. G. Madsen. The double travelling salesman problem with multiple stacks - formulation and heuristic solution approaches. *European Journal of Operational Research*, 198(1), 2009.

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

[15] M. Reimann, K. Doerner, and R. F. Hartl. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563–591, April 2004.

[16] M. Reimann, M. Stummer, and K. Doerner. A savings based ant system for the vehicle routing problem. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1317–1326, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[17] Paolo Toth and Daniele Vigo, editors. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[18] C. D. J. Waters. A Solution Procedure for the Vehicle-Scheduling Problem Based on Iterative Route Improvement. *The Journal of the Operational Research Society*, 38(9):833–839, September 1987.

[19] E. Zachariadis, C. Tarantilis, and C. 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.