Definizione e implementazione di algoritmi per l'ottimizzazione del trasferimento di container tra banchina e piazzale in terminal portuali

Tesi di Laurea triennale in Ingegneria Industriale

Gestione Energia Ambiente

Scuola Politecnica
Università degli Studi di Genova

31 Ottobre 2014

Relatore

Prof. Marco Maratea

Candidato

Giacomo Gilardi

Relatore

Prof. Silvia Siri

Allora Giobbe rispose al Signore e disse: «Io riconosco che tu puoi tutto e che nulla può impedirti di eseguire un tuo disegno. Chi è colui che senza intelligenza offusca il tuo disegno? Sì, ne ho parlato; ma non lo capivo; sono cose per me troppo meravigliose e io non le conosco. Ti prego, ascoltami, e io parlerò; ti farò delle domande e tu insegnami!» (Giobbe 12:13)

Ringraziamenti

Le persone che desidero ringraziare sono sicuramente i miei relatori Marco e Silvia per l'aiuto e i consigli che mi hanno dato senza i quali non avrei potuto realizzare questa tesi. Si ringrazia inoltre Claudia Caballini per avere fornito le informazioni che mi erano necessarie per modellare correttamente l'operatività interna dei terminal portuali. Non posso non ringraziare i miei compagni Davide, Tommaso e Giorgia per il tempo trascorso insieme in questi tre anni, per le ore trascorse insieme a studiare; uno speciale ringraziamento va a Matteo: senza i tuoi appunti e senza la tua penna rossa non avrei passato numerosi esami.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Motivazioni ed introduzione al problema

Negli ultimi decenni il commercio containerizzato è sempre stato il segmento di mercato in maggiore crescita, costituendo circa il 16% in volume del commercio marittimo globale nel 2012 e più di metà in valore (nel 2007) [1]. Il grafico riportato in figura 1.1 mostra come, nonostante la contrazione dei consumi e l'avversa situazione economica degli ultimi anni, il mercato globale che utilizza come unità di carico il container sia in un trend positivo di crescita; infatti anche se l'anno 2009 ha mostrato una flessione negativa molto marcata, questo mercato negli ultimi quattro anni mostra segnali positivi di ripresa. Le motivazioni di questo sviluppo, nel corso degli ultimi decenni, sono da ricercarsi nei vantaggi che contraddistinguono questa tipologia di unità di carico: eliminazione della fase di spacchettamento delle merci ad ogni fase del trasferimento, design ideato per un'operatività semplice e veloce facilitando sia il carico che lo scarico del mezzo di trasporto, protezione da urti e maltempo, e semplificazione del controllo e della programmazione delle spedizioni.

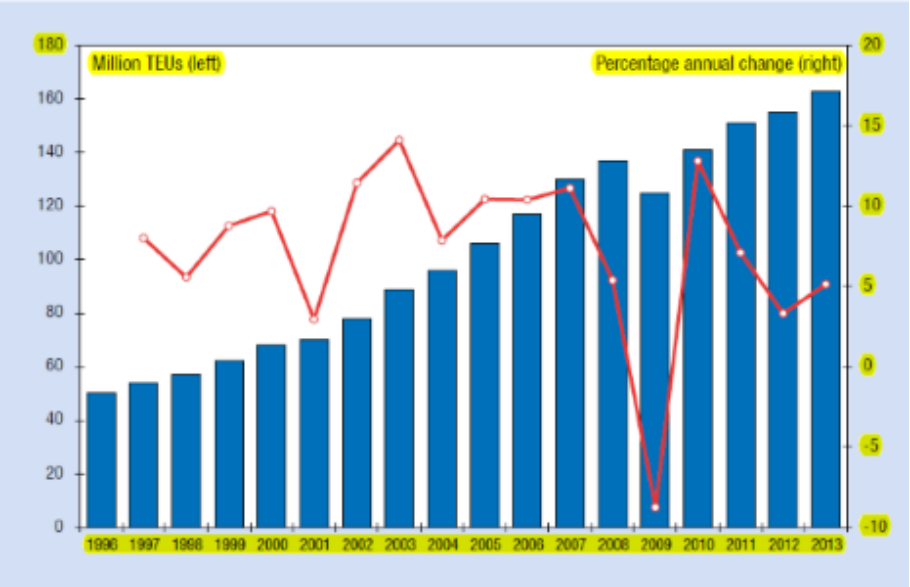

Figura 1.1: Commercio Globale di Container, 1996-2013 (Milioni di TEU e variazione percentuale annua, Review of Meritime Transport, UNCTAD, 2013

Con il diffondersi dell'utilizzo dei container è aumentato il numero di terminal container portuali e la competizione fra questi, soprattutto tra quelli geograficamente vicini, è diventata sempre più importante. Come conseguenza, l'utilizzo dell'*information technology* e delle tecniche della Ricerca Operativa sono diventati di fondamentale importanza per l'ottimizzazione della maggior parte dell'operatività dei terminal. I parametri che rendono un terminal più competitivo rispetto ad un altro, e quindi preferibile dalle principali compagnie di navigazione come punto d'attracco per le navi, sono principalmente: il tempo di permanenza (medio) per una nave in porto e i ratei di carico e scarico delle gru (movimenti/h). Per effettuare l'ottimizzazione di uno di questi parametri è necessario individuare all'interno dell'operatività del terminal il cosidetto *collo di bottiglia*, (*bottleneck*), ossia l'operazione o la sequenza di operazioni che ha il maggior impatto negativo sulla funzione obiettivo, ossia sul parametro che vogliamo migliorare, ed intervenire su questo.

1.1.1 Terminal Container

Un terminal container può essere descritto come un sistema aperto, all'interno del quale il flusso di merci ha due direzioni prevalenti: da terra verso il mare e viceversa.

Il flusso di merci viene gestito mediante due interfacce:

- la banchina per lo scarico e il carico delle navi;
- il piazzale dove i container sono caricati e scaricati su camion e treni.

La banchina è dotata di gru cosidette *portainer* o *quay crane*, le quali sono dotate di un un carrello (trolley) il quale si muove lungo il suo braccio ed è dotato di uno *spreader*, dispositivo per prelevare e trattenere il container durante la movimentazione. Queste gru hanno una produttività teorica di 50-60 TEU/h (il TEU, *Twenty feet Equivalent Unit*, è l'unità di misura delle unità di carico), che in condizioni di operatività normale si riducono a circa 22-30 TEU/h. I container vengono stoccati a piazzale sulla base di un attenta pianificazione, detta *macroplanning*; essi vengono raggruppati in blocchi divisi a loro volta in baie, file e tiri in base alla loro classe di peso, alla loro destinazione e al mezzo di trasporto che li deve prelevare. è possibile quindi identificare la posizione di ogni container mediante un sistema di coordinate definite su tre assi cartesiani: la fila e la baia forniscono la posizione sul piano, i tiri determinano l'informazione riguardante l'impilamento verticale dei container. L'area di stoccaggio è usualmente suddivisa in due macro-aree: una *Import Area* dove vengono stoccati i container in attesa di essere veicolati verso il territorio e una *Export Area* dove si trovano i container che dovranno essere spediti via nave. Vi sono inoltre sezioni del piazzale dotate di prese elettriche per container che necessitano refrigerazione (postazioni *reefer*) ed altre a cui sono destinati i container contrassegnati come pericolosi dato il loro contenuto; questi ultimi vengono gestiti da un iter burocratico in genere differente da quello degli altri conteiner e controllati con particolare cautela.

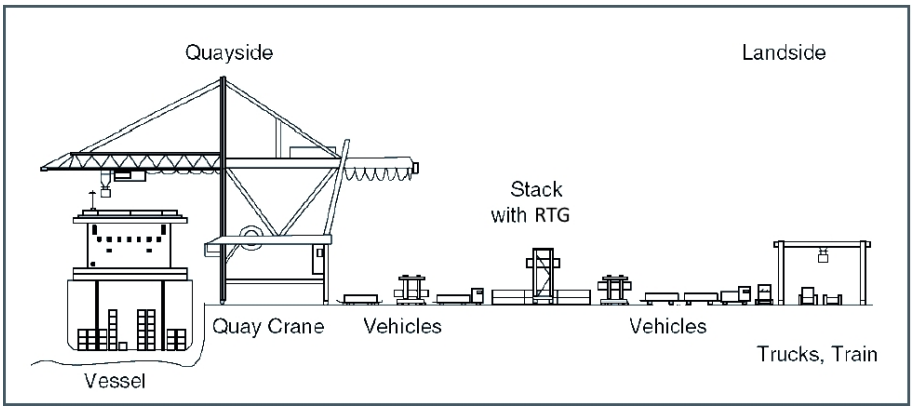

Figura 1.2: Rappresentazione schematica del layout e dell'equipaggiamento di un terminal portuale [5].

Lo stoccaggio dei container in import avviene generalmente senza un'azione di pianificazione sulla loro collocazione in quanto non è noto a priori nè il momento in cui essi verranno prelevati nè la sequenza in cui essi verranno prelevati; non avviene così per i container in export in quanto una loro disposizione pianificata riduce di molto la durata delle operazioni di carico. Nel corso della tesi si ipotizzerà di essere a conoscenza della disposizione dei container a piazzale.

La movimentazione dei container all'interno del terminal può avvenire mediante diverse tipologie di mezzi di trasporto:

- *RTG, Rubber Tyred Gantry*, sono gru a portico utilizzate per lo stoccaggio e il disimpilaggio dei container a piazzale, vengono generalmente assegnate ad un blocco di container, possono muoversi da una parte all'altra del piazzale essendo dotate di gomme e di un motore a combustione interna, in genere nella loro ampiezza comprendono anche una corsia per facilitare il carico/scarico delle ralle;

Figura 1.3: Rubber Tyred Gantry, RTG [6].

- *RMG, Rail Mounted Gantry*, sono gru a portale dotate di una capacità di stoccaggio molto maggiore alle RTG (gestiscono fino a 15 file, 50 m di larghezza), si spostano lungo traittorie rettilinee solo lungo binari mediante un motore elettrico rimanendo vincolate quindi all'area del piazzale a cui sono assegnate;

Figura 1.4: Rail Mounted Gantry, RMG [7].

- *reach stacker*, mezzo di trasporto e movimentazione container molto flessibile

dotato di spreader telescopico, permette l'impilamento di container da un minimo di 3 ad un massimo di 5 tiri, utilizzato in genere come supporto per le gru di piazzale;

Figura 1.5: Reach Stacker [7].

- *ralle, trattori stradali*, si occupano del trasferimento container tra banchina e piazzale di stoccaggio, vengono caricati e scaricati dalle gru o da *reach stacker*;

Figura 1.6: Ralla [7].

- *straddle carrier*, o carrello cavaliere, sono carrelli elevatori in grado di prelevare i container fino ad un limitato numero di tiri (2 o 4) e di trasportarli molto rapidamente, inoltre presentano dimensioni e ingombri limitati. Per poter prelevare un container la struttura a portico del mezzo deve sormontare la fila, per questo i terminal che ne fanno uso distanziano le file tra loro di almeno 1,5 metri; per questo motivo questi mezzi sono adeguati solo per terminal che dispongono di ampi spazi per lo stoccaggio.

Figura 1.7: Straddle Carrier, [6].

Terminal portuali che non dispongono di ampi spazi per lo stoccaggio devono necessariamente, cercare di effettuare uno sfruttamento intensivo dello spazio a disposizione, ricercando quindi uno sviluppo verticale dell'area dello stoccaggio; di conseguenza, nella scelta dell'equipaggiamento, la scelta ricade su reach stacker, gru a portico e ralle per lo stoccaggio e il trasferimento alla banchina dei container. Per terminal che hanno un'ampia disponibilità di spazio, risulta essere preferibile effettuare uno sfruttamento estensivo dell'area di stoccaggio mediante l'utilizzo di straddle carrier.

1.2 Stato dell'arte

Avendo l'obiettivo di realizzare l'ottimizzazione del trasferimento di container tra il piazzale di stoccaggio e la banchina, si è preferito procedere per passi successivi, analizzando le operazioni di scarico e carico delle navi e affrontando ogni parte dell'operatività come un problema a sè.

Quando una nave attracca in un terminal portuale, ad essa viene assegnato un certo tratto della banchina e quindi un certo numero di gru di banchina e di ralle per il trasferimento dei container. Lo scarico/carico di una nave prevede che:

- la gru di banchina prelevi dalla nave/ralla un container e lo posizioni su una ralla/nave;
- la ralla quindi trasporta il container alla locazione per quel container stabilita dal macroplanning/ritorna vuota al piazzale;
- nel piazzale, in ciascun blocco, una RTG provvede a stoccare il container appena scaricato oppure a prelevare un nuovo container e a posizionarlo sulla ralla designata per il trasporto in banchina.

In questa sequenza di operazioni il *terminal planner* deve definire: lo scheduling delle gru di banchina, l'assegnamento delle missioni di trasporto alle ralle e lo scheduling delle gru di piazzale.

Operativamente lo scheduling delle gru di banchina viene effettuato mediante la definizione delle sequenze di scarico e carico della nave; la lista dei container che devono essere scaricati dalla nave viene comunicata al terminal planner dalla

compagnia di navigazione; il sequenziamento delle operazioni di scarico viene deciso con l'obiettivo di non compromettere la stabilità della nave. La sequenza di carico della nave è invece stabilita dagli operatori del terminal sulla base dello *stowage plan* fornito dalla compagnia di navigazione; obiettivo del terminal planner è di definire quali container vadano caricati sulla nave, in che ordine essi vadano caricati e dove debbano essere posizionati sulla nave rispettando i vincoli definiti dalla compagnia di navigazione; tale operazione detta *stowage sequencing* definisce un problema detto Master Bay Plan Problem; questo problema è affrontato in letteratura ad esempio mediante la definizione di un modello di programmazione lineare binaria e di un approccio euristico in [3], in [4] è presentato un confronto fra differenti euristiche per risovere questo problema insieme ad un'analisi computazionale.

In letteratura il problema dell'assegnamento delle ralle è stato affrontato in [2] mediante la definizione di algoritmi euristici, dimostrando inoltre l'ottimalità di tali algoritmi nel caso delle singole sequenze di scarico e carico;

Il sequenziamento della gru di piazzale è determinato dalla definizione della sequenza di scarico e carico; è possibile migliorare la produttività utilizzando le gru di piazzale in modalità *twin lift* ovvero agganciando due container da 20' contemporaneamente oppure utilizzando due gru, a diverse altezze, sullo stesso blocco.

1.3 Obiettivi della tesi

Questa tesi si propone di:

- definire ed implementare in C# algoritmi euristici che effettuino l'assegnamento ad un fissato numero di ralle delle missioni di trasporto dei container tra banchina e piazzale e viceversa, ipotizzando che una sola gru di banchina sia stata assegnata alla nave e di conoscere le sequenze di scarico e carico della nave;

- definire ed implementare in C# un algoritmo che rielabori la sequenza di carico delle navi con l'obiettivo di ottimizzare gli spostamenti ed i *rehandling*, ossia le operazioni inproduttive, di ciascuna RTG nel piazzale; questo algoritmo farà

utilizzo dell'algoritmo di assegnazione precedente tenendo conto del tempo di trasporto di ogni container dal piazzale alla banchina;

- effettuare un'analisi computazionale degli algoritmi implementati allo scopo di quantificarne le prestazioni in termini di tempo di elaborazione al variare dei principali parametri.

1.4 Struttura della tesi

Questa tesi è strutturata nel seguente modo:

- nel Capitolo 2 *'Instradamento dei veicoli nelle operazioni di scarico e carico'* viene prima descritto il problema dell'assegnazione delle ralle delle missioni di trasporto dei container tra banchina e piazzale di stoccaggio e viceversa, vengono quindi descritti ed implementati due algoritmi euristici in C# per ottimizzare l'assegnamento nel caso dello scarico e del carico delle navi; questi due algoritmi vengono quindi combinati per effettuare l'instradamento delle ralle durante le operazioni sia di scarico che di carico; infine si è effettuata un'analisi delle prestazioni degli algoritmi in termini di tempo, al variare del numero di ralle assegnate alla gru di banchina e e del numero di container da scaricare e caricare;

- nel Capitolo 3 *'Ottimizzazione degli spostamenti delle RTG durante le operazioni di carico'* viene descritta l'operatività della gru di piazzale facendo riferimento ad una RTG, viene quindi implementato in C# un algoritmo che cerca di minimizzare il numero di spostamenti e di *rehandling* effettuati dalla RTG; quindi si è combinato questo algoritmo con quello implementato nel Capitolo 2 al fine di tenere conto anche dei tempi di trasporto dei container tra piazzale e banchina; infine si è effettuata un'analisi prestazionale per quantificare l'aumento nei tempi di elaborazione dell'algoritmo al variare del numero di container che la gru deve caricare sulle ralle;

- nel Capitolo 4 *Conclusioni*, sulla base dei risultati ottenuti, vengono formulate le opportune considerazione e descritti possibili sviluppi futuri del presente lavoro.

Capitolo 2

Instradamento dei veicoli nelle operazioni di carico e scarico

In questo capitolo si descrive il problema di assegnamento delle missioni di trasporto dei container ai veicoli terrestri all'interno di un terminal container. Nella prima sezione viene definito il problema, nella seconda sezione si descrive l'implementazione di algoritmi euristici per la minimizzazione del makespan delle operazioni di scarico e carico della nave, nella terza sezione viene esposta l'analisi computazionale degli algoritmi descritti.

2.1 Descrizione del problema

Quadro generale La situazione presa in esame è quella di un terminal container nel quale, all'arrivo di una nave, un certo numero di container devono essere scaricati e poi caricati. Prima dell'arrivo della nave al terminal tutte le informazioni relative al suo carico sono inviate al terminal dalla compagnia di navigazione; in base a queste informazioni vengono definite le sequenze di carico e scarico dei singoli container.

Nelle operazioni di scarico, ogni veicolo, dopo essere stato caricato dalla gru di banchina, consegna il container alla zona di stoccaggio del piazzale designata; una volta a destinazione il veicolo viene scaricato da un *reach stacker* e ritorna alla banchina; per semplicità si è fatta ipotesi che un *reach stacker* sia sempre disponibile all'arrivo della ralla. Le operazioni di carico si svolgono in maniera simmetrica.

In questo capitolo si ipotizza che le sequenze di carico e scarico dei container siano già state definite e si descrivono gli algoritmi euristici implementati, per effettuare l'assegnamento delle missioni di trasporto dei container da caricare e scaricare ai veicoli assegnati alla nave.

Definizione del problema Nel caso analizzato si ipotizza che la singola nave sia servita da una singola gru di banchina e si assume che ad essa sia stata assegnata una flotta di k veicoli posti inizialmente nell'area della banchina assegnata alla nave. Ogni missione di trasporto di un container che un veicolo deve effettuare viene definita *job*. Quando un veicolo porta a termine un job ritorna alla banchina.

Alla gru di banchina viene assegnata una sequenza ordinata di n job,

$$J_{-/+} : \{J_1^{-/+}, J_2^{-/+}, .., J_{n+m}^{-/+}\},$$

La sequenza $J_{-/+}$ è una sequenza composta di due parti: una prima parte sono i container che devono essere scaricati dalla nave, $J_k^-, i = 1, 2, ..., n$, (container in *import*), la seconda parte sono i container che devono essere caricati sulla nave, $J_j^+, j = 1, 2, ..., m$, (container in *export*).

Il fatto che la sequenza $J_{-/+}$ sia una sequenza ordinata impone tra i job vincoli di precedenza. Infatti, la gru di banchina non inizia a scaricare l'$i-$esimo container fino a quando tutti i predecessori del job i sono stati scaricati ossia la gru di banchina non inizia a caricare il $j-$esimo container fino a quando tutti i predecessori del job j sono stati caricati. Il rispetto dei vincoli di precendenza è fondamentale in quanto le sequenze di scarico e carico sono definite in modo da non alterare la stabilità della nave.

Ad ogni job J_i, *i=1,2,...,n*, è associato un punto di carico (scarico) all'interno del piazzale di stoccaggio di cui è noto il tempo di trasporto d_i necessario per raggiungerlo dalla banchina. Si è assunto che la distanza tra le locazioni a piazzale di due job, J_i, J_h è pari alla differenza, eventuamente in modulo, dei tempi di trasporto di J_i, J_h,

$$d_{ih} = d_{hi} = |d_i - d_h|.$$

Questo significa che da ogni locazione di piazzale è possibile raggiungere ogni altra locazione di piazzale direttamente.

L'operazione di carico (scarico) di un container richiede alla gru 2 movimenti, uno per sollevare il container dalla ralla (dalla nave) e un altro per posizionarlo sulla nave (sul veicolo); il tempo di processamento da parte della gru di ogni job viene indicato con s.

Date le ingombranti dimensioni dei container di taglia standard, le gru scaricano i container sempre su un veicolo in modo da evitare l'operazione aggiuntiva di sollevare il container da terra al veicolo.

2.2 Gli algoritmi e la loro implementazione

Nell'implementazione degli algoritmi si è proceduto per passi successivi, implementando gli algoritmi di ottimizzazione, applicandoli a sequenze solamente di scarico e carico, e solo in seguito si sono combinati i due algoritmi in uno solo per ottimizzare l'insieme delle operazioni relative alla nave. Gli algoritmi implementati costruiscono la soluzione effettuando una serie di scelte; in particolare essi fanno sempre la scelta che sembra ottima in un dato momento, ossia fanno una scelta localmente ottima, nella speranza che tale scelta porti ad una soluzione globalmente ottima. Questa modalità nel procedere 'avida' è propria di questa famiglia di algoritmi detti *greedy algorithms*, appunto algoritmi 'avidi'.

Le operazioni di scarico Nell'ottimizzazione delle operazioni di scarico, lo scopo dell'algoritmo è di assegnare i job della sequenza J^- ai k veicoli della flotta di ralle assegnata alla nave. L'algoritmo procede nel seguente modo: assegna i primi k job ai singoli veicoli, quindi procede nell'assegnare il job successivo al primo veicolo disponibile che arriva alla gru di banchina; si rende perciò necessario per ogni job verificare quale sarà il primo veicolo a giungere alla banchina. L'algoritmo termina quando tutti i job sono assegnati ai veicoli, e l'output della procedura saranno le sequenze di container che ogni veicolo dovrà trasportare.
La porzione di codice sottostante mostra come l'algoritmo sia stato implementato

andando a scorrere iterativamente la sequenza di job, andando ad assegnare il trasporto di ogni container al veicolo avente *tempoArrivo* minore.

```
/*Assegno i job ai veicoli con l'algoritmo greedy*/
for (int j = 0; j < dropOffSequence.length; j++)
{
 int vehicle = 0;
 int disponibilita = 100000;
 for (int i = 0; i < k; i++)
 {
  int serviceTime = 0;
  for (int h = 0; h < J; h++)
  {
   serviceTime += M[i, h] * (dropOffSequence[H] * 2 + s);
  }
  if (serviceTime < disponibilita)
  {
   disponibilita = serviceTime;
   vehicle = i;
  }
 }
 M[vehicle, j]++;
}
```

La sequenza di container che la gru di banchina deve scaricare è memorizzata in un array monodimensionale *DropOffSequence* ciascuno con la rispettiva distanza d_j *j=1,2,...,n* dalla nave misurata in termini tempo di trasporto; la matrice M è una matrice rettangolare avente k righe (numero di veicoli) e *DropOffSequence.length* colonne (numero di job), i cui elementi m_{ij} sono tali che,

$$m_{ij} = \begin{cases} 1 & \text{se } job\ j \text{ è } assegnato\ al\ veicolo\ i \\ 0 & \text{se } job\ j\ non \text{ è } assegnato\ al\ veicolo\ i \end{cases}$$

$i = 1, ..., k$ e $j = 1, ..., dropOffSequence.length$.

Il calcolo del *Makespan* della gru di banchina è realizzato andando a scorrere la matrice di assegnazione dei job ai veicoli e sommando ai tempi di processamento s della gru i ritardi dei veicoli nel trasporto dei container (*tempoArrivo* + s).

```
/*Calcolo Makespan Gru per la sequenza generata dall'algoritmo*/
for (int j = 0; j < J; j++)
{
 for (int i = 0; i < k; i++)
```

```
5   {
6    if (M[i, j] == 1)
7    {
8
9     int tempoArrivo = 0;
10    for (int h = 0; h < j; h++)
11    {
12     tempoArrivo += M[i, h] * (dropOffSequence[H] * 2 + s);
13    }
14
15    if (Makespan < tempoArrivo)
16    {
17     Makespan += tempoArrivo-Makespan + s;
18    }
19    else
20    {
21     Makespan +=  s;
22    }
23   }
24  }
25 }
```

Le operazioni di carico Nell'ottimizzazione delle operazioni di carico , l'algoritmo utilizzato è detto *reversed greedy*; questo algoritmo lavora nel seguente modo:

- considera ogni job della sequenza J^+ come se fosse un container che deve essere scaricato dalla nave;
- inverte l'ordine della sequenza J^+ che viene denominata J_R^+;
- applica l'algoritmo *greedy* alla sequenza J_R^+ e ottiene una lista V_i di job J_j^- assegnati al veicolo i-esimo, $V_i = \{J_{i_1}^-, ..., J_{i_h}^-\}$,$i = 1, .., k$,$j = 1, .., h$ e $h =$ *numero job assegnati al veicolo* i;
- inverte l'ordine dei job delle liste V_i , $i = 1, .., k$, ottenendo per l'i-esimo veicolo la lista $v_i = \{J_{i_h}^-, J_{i_{h-1}}^-..., J_{i_1}^-\}$.

Il secondo passo dell'algoritmo viene implementato in questo caso scorrendo la sequenza J^+ dalla fine all'inizio.

```
1 /*Assegno i job ai veicoli con l'algoritmo reversed greedy*/
2 for (int j = uploadSequence.length-1; j >= 0; j--)
```

```
3  {
4   int vehicle = 0;
5   int disponibilita = 100000;
6
7   for (int i = 0; i < k; i++)
8   {
9    int serviceTime = 0;
10   for (int h = 0; h < J; h++)
11   {
12    serviceTime += M[i, h] * (uploadSequence[H] * 2 + s);
13   }
14   if (serviceTime < disponibilita)
15   {
16    disponibilita = serviceTime;
17    vehicle = i;
18   }
19  }
20  M[vehicle, j]++;
21 }
22 /*Inverto la sequenza di job assegnata ad ogni veicolo*/
23 for (int i = 0; i < k; i++)
24 {
25
26  for (int j = 0; j < J; j++)
27  {
28   Mreversed[i, J - j-1] = M[i, j];
29  }
30
31 }
```

La sequenza di container che la gru di banchina deve caricare sulla nave è memorizzata in un array monodimensionale *uploadSequence* ciascuno con la rispettiva distanza d_j dalla nave misurata in termini di tempo di trasporto; la matrice *Mreversed* è una matrice rettangolare avente k righe(numero di veicoli) e *uploadSequence.length* colonne (numero di job), i cui elementi m_{ij} sono tali che,

$$m_{ij} = \begin{cases} 1 & \text{se } job\ j \text{ è } assegnato\ al\ veicolo\ i \\ 0 & \text{se } job\ j\ non \text{ è } assegnato\ al\ veicolo\ i \end{cases}$$

$i = 1, ..., k$ e $j = 1, ..., uploadSequence.length$

Il calcolo del *Makespan* della gru di banchina è realizzato andando a scorrere la matrice di assegnazione dei job ai veicoli *Mreversed* e sommando ai tempi di processamento

s della gru i ritardi dei veicoli nel trasporto dei container e il tempo d_j di trasporto del j-esimo job (*tempoArrivo + (uploadSequence[j] + s*).

```
/*Calcolo Makespan Gru per la sequenza generata dall'algoritmo*/
for (int j = 0; j < J; j++)
{
 for (int i = 0; i < k; i++)
 {
  if (Mreversed[i, j] == 1)
  {

   int tempoArrivo = 0;
   for (int h = 0; h < j; h++)
   {
    tempoArrivo += Mreversed[i, h] * (uploadSequence[H] * 2 + s);
   }

   if (Makespan < tempoArrivo)
   {
    Makespan += tempoArrivo-Makespan + (uploadSequence[j] + s);
   }
   else
   {
    Makespan += (uploadSequence[j] + s);
   }
  }
 }
}
```

L'operazione di scarico e carico completa Per ottimizzare l'intera operazione di scarico e carico della nave si è implementato un algoritmo che combina gli algoritmi descritti precedentemente, detto *combined greedy*. Sia $J^{-/+}$ la sequenza di container che devono essere scaricati e poi caricati dalla nave, l'algoritmo *combined greedy* applica l'algoritmo *greedy* alla sottosequenza di container che devono essere caricati, quindi procede con l'applicare l'algoritmo *reversed greedy* alla sottosequenza di container che devono essere caricati sulla nave. Ogni volta che la lista di container che un veicolo deve trasportare termina con un job J^- il veicolo deve tornare alla banchina, compiendo un viaggio "a vuoto". L'idea di questo algoritmo è di utilizzare questo viaggio per trasportare un container J^+, che deve essere caricato sulla nave, in modo da ridurre al minimo i tempi morti. Per distinguere i container in import da

quelli in export si è scelto di renderli rispettivamente negativi e positivi all'interno della matrice di assegnamento.

```
/*Assegno alle ralle i job da scaricare con algoritmo greedy */

for (int j = 0; j < dropOffSequence; j++)
{
 int vehicle = 0;
 int disponibilita = 100000;

 for (int i = 0; i < k; i++)
 {
  int serviceTime = 0;
  for (int h = 0; h < dropOffSequence.length; h++)
  {
   serviceTime += Math.Abs(M[i, h]) * (dropOffSequence[H] * 2 + s);
  }
  if (serviceTime < disponibilita)
  {
   disponibilita = serviceTime;
   vehicle = i;
  }
 }
 M[vehicle, j] = -1;
}

/*Assegno alle ralle i job da scaricare con algoritmo reversed greedy */

for (int j = uploadSequence.length - 1; j >= 0; j--)
{
 int vehicle = 0;
 int disponibilita = 100000;

 for (int i = 0; i < k; i++)
 {
  int serviceTime = 0;
  for (int h = 0; h < uploadSequence.length; h++)
  {
   serviceTime += Mupload[i, h] * (uploadSequence[H] * 2 + s);
  }
  if (serviceTime < disponibilita)
  {
   disponibilita = serviceTime;
   vehicle = i;
  }
 }
 Mupload[vehicle, j]++;
```

```
45 }
46 /*Inverto la sequenza di job assegnata ad ogni veicolo*/
47 for (int i = 0; i < k; i++)
48 {
49  for (int j = 0; j < uploadSequence.length; j++)
50  {
51   MuploadReversed[i, uploadSequence.length - j - 1] = Mupload[i, j];
52  }
53 }
54 /*Genero la matrice di assegnamento completa*/
55 for (int i = 0; i < k; i++)
56 {
57  for (int j = 0; j < uploadSequence.length; j++)
58  {
59   M[i, dropOffSequence.length + j] = MuploadReversed[i, j];
60  }
61 }
```

La sequenza di container che la gru di banchina deve caricare sulla nave è memorizzata in due array monodimensionali *uploadSequence* e *dropOffSequence*, ciascuno con la rispettiva distanza d_j dalla nave misurata in termini di tempo di trasporto; la matrice M è una matrice rettangolare avente k righe e *uploadSequence.length* + *dropOffSequence.length* colonne (numero di job totali), i cui elementi m_{ij} sono tali che,

$$m_{ij} = \begin{cases} 1 & \text{se } job\ j\ di\ carico \text{ è } assegnato\ al\ veicolo\ i \\ -1 & \text{se } job\ j\ di\ scarico \text{ è } assegnato\ al\ veicolo\ i \\ 0 & \text{se } job\ j\ non \text{ è } assegnato\ al\ veicolo\ i \end{cases}$$

$i = 1, ..., k$, e $j = 1, ..., uploadSequence.length + dropOffSequence.length$

Il calcolo del *Makespan* della gru di banchina è realizzato andando a scorrere la matrice di assegnazione M dei job ai veicoli e distinguendo se il job corrente è un container in import o in export e se l'ultimo job eseguito dal veicolo è di segno diverso da quello corrente (ovvero il veicolo si trova già a piazzale e deve trasportare un container dal piazzale alla nave).

```
1 /*Calcolo makespan della gru data la matrice di assegnazione dei job */
2 int[] Sequence = new int[dropOffSequence.length + uploadSequence.length];
3 dropOffSequence.CopyTo(Sequence, 0);
4 uploadSequence.CopyTo(Sequence, uploadSequence.length);
5 for (int j = 0; j < (uploadSequence.length + dropOffSequence.length); j++)
```

```
6  {
7
8   for (int i = 0; i < k; i++)
9   {
10   if (M[i, j] != 0)
11   {
12    int tempoArrivo = 0;
13    for (int h = 0; h < j; h++)
14    {
15     if (j + 1 < (uploadSequence.length + dropOffSequence.length))
16     {
17      if (M[i, h] < 0)
18      {
19       for (int p = h + 1; p <= j + 1; p++)
20       {
21        if (M[i, h] * M[i, p] >= 0)
22        {
23         tempoArrivo += Math.Abs(M[i, h]) * (Sequence[H] * 2 + s);
24         break;
25        }
26        else
27        {
28         tempoArrivo += Math.Abs(M[i, h])*((Sequence[H] + s)
29         + Math.Abs(Sequence[h + 1]-Sequence[H]));
30         break;
31        }
32       }
33      }
34     }
35    }
36    if (M[i, j] == -1)
37    {
38     if (Makespan < tempoArrivo)
39     {
40      Makespan += tempoArrivo - Makespan + s;
41     }
42     else
43     {
44      Makespan += s;
45     }
46    }
47    else
48    {
49     if (Makespan < tempoArrivo)
50     {
51      Makespan += tempoArrivo - Makespan + Sequence[j] + s;
52     }
```

```
53      else
54      {
55       Makespan += Sequence[j] + s;
56      }
57     }
58    }
59   }
60  }
```

2.3 Analisi Computazionale

Per verificare l'efficienza con la quale i tre algoritmi lavorano si è scelto di effettuare un'analisi che verifichi come aumentano i tempi di elaborazione degli algoritmi in funzione del numero di job della sequenza $J^{-/+}$ e del numero k di veicoli assegnati alla nave. Viene fissato un numero di veicoli $k \in \{5, 10, 15, 20\}$ quindi viene generata mediante una distribuzione uniforme una sequenza di job di lunghezza $l \in \{1000, 2000, 3000, 4000, 5000\}$, ciascuno con la rispettiva distanza dalla nave misurata in termini di tempo di trasporto. Per ciascuna coppia (k,l) si sono effettuate cinque repliche andando a misurare il tempo di eleborazione e poi si è mediato il risultato. Come strumento per effettuare l'analisi si è utilizzato Microsoft Visual Studio 2013 su un laptop dotato di un processore AMD A8 - 4500M 1.9-2.8 GHz e 8 Gb di RAM.

2.3.1 Algoritmo greedy

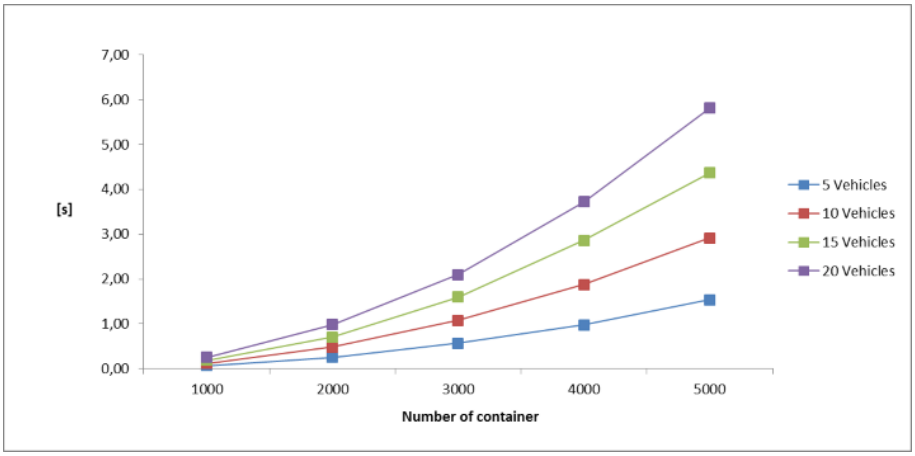

Figura 2.1: Algoritmo greedy, tempi di elaborazione

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,078	0,281	0,577	1,014	1,560
Rep. 2	0,062	0,250	0,577	0,983	1,529
Rep. 3	0,062	0,250	0,546	0,967	1,529
Rep. 4	0,062	0,250	0,577	0,967	1,544
Rep. 5	0,062	0,250	0,562	0,967	1,529
Media	0,066	0,256	0,568	0,980	1,538

Tabella 2.1: Algoritmo greedy, tempi elaborazione con $k = 5$

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,140	0,499	1,092	1,919	2,948
Rep. 2	0,125	0,484	1,076	1,856	2,902
Rep. 3	0,109	0,484	1,092	1,872	2,902
Rep. 4	0,109	0,484	1,061	1,888	2,902
Rep. 5	0,125	0,484	1,076	1,872	2,902
Media	0,122	0,487	1,080	1,881	2,911

Tabella 2.2: Algoritmo greedy, tempi elaborazione con $k = 10$

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,203	0,733	1,607	2,808	4,384
Rep. 2	0,172	0,718	1,607	2,881	4,337
Rep. 3	0,172	0,702	1,591	2,808	4,352
Rep. 4	0,203	0,686	1,607	2,792	4,347
Rep. 5	0,172	0,702	1,591	3,011	4,431
Media	0,184	0,708	1,601	2,860	4,370

Tabella 2.3: Algoritmo greedy, tempi elaborazione con $k = 15$

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,250	0,952	2,106	3,713	5,912
Rep. 2	0,265	0,950	2,106	3,775	5,834
Rep. 3	0,250	0,952	2,090	3,682	5,741
Rep. 4	0,265	1,030	2,075	3,676	5,741
Rep. 5	0,250	1,030	2,106	3,744	5,815
Media	0,256	0,983	2,097	3,718	5,809

Tabella 2.4: Algoritmo greedy, tempi elaborazione con $k = 20$

2.3.2 Algoritmo reversed greedy

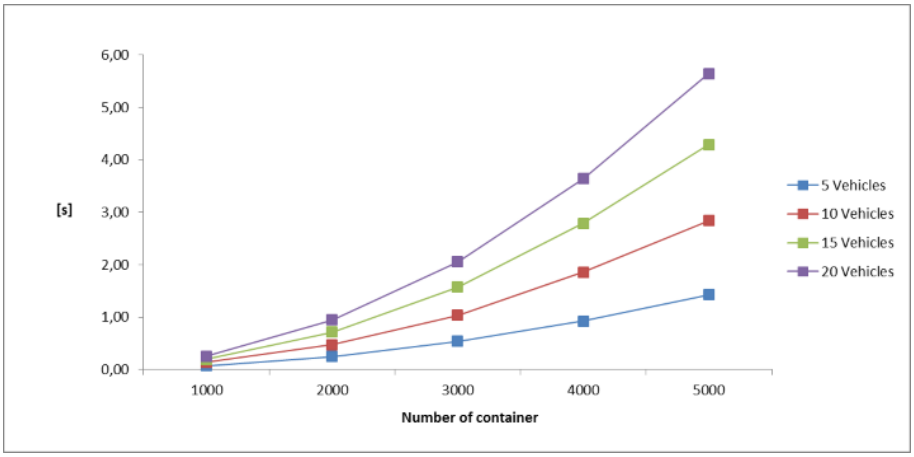

Figura 2.2: Algoritmo reversed greedy, tempi di elaborazione

Num. job	1000	2000	3000	4000	5000
Rep.1	0,062	0,265	0,530	0,936	1,435
Rep.2	0,068	0,250	0,530	0,920	1,420
Rep.3	0,078	0,234	0,546	0,920	1,435
Rep.4	0,078	0,250	0,546	0,936	1,441
Rep.5	0,078	0,250	0,530	0,928	1,420
Media	0,073	0,250	0,537	0,928	1,430

Tabella 2.5: Algoritmo reversed greedy, tempi elaborazione con $k = 5$

Num. job	1000	2000	3000	4000	5000
Rep.1	0,140	0,484	1,045	1,953	2,839
Rep.2	0,140	0,468	1,030	1,825	2,839
Rep.3	0,131	0,468	1,030	1,814	2,839
Rep.4	0,140	0,484	1,036	1,841	2,855
Rep.5	0,140	0,474	1,035	1,841	2,839
Media	0,138	0,475	1,035	1,855	2,842

Tabella 2.6: Algoritmo reversed greedy, tempi elaborazione con $k = 10$

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,203	0,718	1,622	2,961	4,384
Rep. 2	0,187	0,702	1,591	2,714	4,306
Rep. 3	0,187	0,709	1,560	2,730	4,228
Rep. 4	0,218	0,718	1,560	2,816	4,243
Rep. 5	0,203	0,718	1,529	2,730	4,275
Media	0,200	0,713	1,573	2,790	4,287

Tabella 2.7: Algoritmo reversed greedy, tempi elaborazione con $k = 15$

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,250	0,952	2,044	3,666	5,569
Rep. 2	0,265	0,936	2,044	3,635	5,601
Rep. 3	0,250	0,936	2,059	3,619	5,663
Rep. 4	0,250	0,952	2,059	3,619	5,632
Rep. 5	0,250	0,943	2,066	3,651	5,741
Media	0,253	0,944	2,054	3,638	5,641

Tabella 2.8: Algoritmo reversed greedy, tempi elaborazione con $k = 20$

2.3.3 Algoritmo combinato

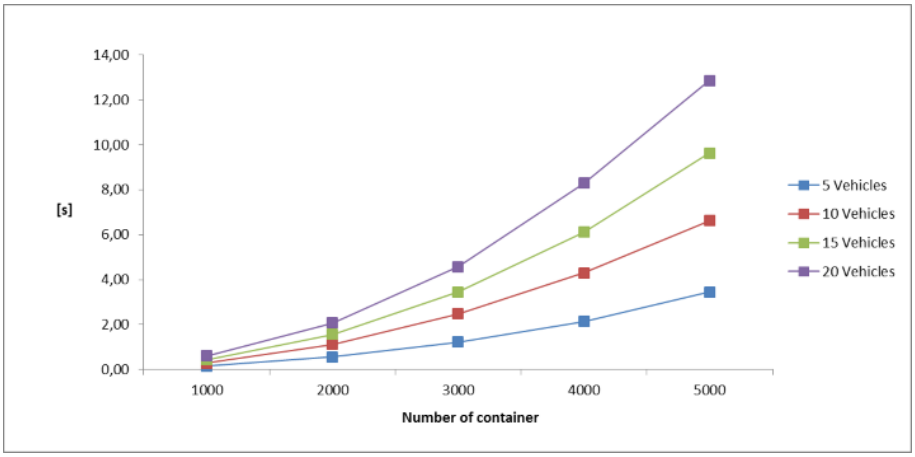

Figura 2.3: Algoritmo combinato, tempi di elaborazione

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,156	0,546	1,232	2,168	3,338
Rep. 2	0,156	0,562	1,217	2,122	3,698
Rep. 3	0,156	0,562	1,217	2,137	3,619
Rep. 4	0,156	0,562	1,212	2,137	3,292
Rep. 5	0,156	0,577	1,201	2,122	3,292
Media	0,156	0,562	1,216	2,137	3,448

Tabella 2.9: Algoritmo combinato, tempi elaborazione con $k = 5$

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,281	1,045	2,356	4,274	6,396
Rep. 2	0,281	1,154	2,387	4,696	6,490
Rep. 3	0,281	1,154	2,543	4,165	7,005
Rep. 4	0,281	1,139	2,543	4,212	6,739
Rep. 5	0,281	1,154	2,558	4,150	6,474
Media	0,281	1,129	2,477	4,299	6,621

Tabella 2.10: Algoritmo combinato, tempi elaborazione con $k = 10$

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,421	1,560	3,448	6,162	9,500
Rep. 2	0,437	1,560	3,526	6,131	9,641
Rep. 3	0,437	1,560	3,432	6,100	9,688
Rep. 4	0,437	1,576	3,448	6,100	9,688
Rep. 5	0,437	1,545	3,432	6,115	9,656
Media	0,434	1,560	3,457	6,121	9,635

Tabella 2.11: Algoritmo combinato, tempi elaborazione con $k = 15$

Num. job	1000	2000	3000	4000	5000
Rep. 1	0,625	2,059	4,618	8,096	12,636
Rep. 2	0,611	2,153	4,571	8,050	12,621
Rep. 3	0,605	2,044	4,571	8,096	12,558
Rep. 4	0,621	2,044	4,540	8,455	13,401
Rep. 5	0,611	2,044	4,540	8,752	13,089
Media	0,615	2,069	4,568	8,290	12,861

Tabella 2.12: Algoritmo combinato, tempi elaborazione con $k = 20$

2.3.4 Risultati

I risultati ottenuti mostrano come il tempo di elaborazione degli algoritmi descritti aumenti leggermente quando aumentano il numero di veicoli e di container. Inoltre per l'algoritmo *combined greedy* si sono ottenuti tempi computazionali che sono circa il doppio rispetto ai tempi degli altri due algoritmi. Anche se questi risultati possono sembrare attesi, è possibile effettuare alcune considerazioni:

- l'aumento dei tempi di elaborazione risulta essere ragionevolmente modesto, anche aumentando significativamente il numero dei container e/o dei veicoli impiegati;
- la maggior parte dei test eseguiti sono stati risolti in meno di 10 secondi, valore che risulta essere in linea con le esigenze di una situazione realistica, inoltre risultati ottimali sono garantiti per i primi due algoritmi euristici; il terzo algoritmo, non fornendo una soluzione ottima,è adatto per situazioni

che necessitano di una soluzione accettabile per l'intera procedura di scarico e carico in tempi brevi.

Capitolo 3

Ottimizzazione degli spostamenti delle RTG durante le operazioni di carico

In questo capitolo si descrive il problema dell'ottimizzazione degli spostamenti delle gru di piazzale, in particolare delle RTG, durante le operazioni di carico delle navi. Nella prima sezione viene definito il problema e la sua similitudine con i problemi di scheduling con tempi di setup dipendenti dalla sequenza, nella seconda sezione si descrive l'algoritmo euristico per la minimizzazione della durata delle operazioni di carico di una nave e la sua implementazione, nella terza sezione viene mostrata l'analisi computazionale dell'algoritmo implementato.

3.1 Descrizione del problema

Quadro generale. L'oggetto di questa analisi è la descrizione della sequenza di operazioni mediante la quale viene effettuato il carico delle navi, con l'obiettivo di identificare dove è possibile intervenire per ridurre il tempo in cui viene effettuato il carico delle navi. La sequenza di operazioni, ripetuta per ogni contenitore, si compone di:

- prelievo del contenitore dal blocco dove è stoccato, mediante RTG, e posizionamento di quest'ultimo su ralla per il trasporto;

- trasporto mediante ralla fino alla banchina;
- prelievo del contenitore dalla ralla e suo posizionamento sulla nave.

Le informazioni necessarie agli operatori del terminal, per poter organizzare le operazioni di carico delle navi, sono:

- la lista dei container da caricare/scaricare, con i dati specifici di ogni container;
- il bayplan della nave, comunicato prima che la nave arrivi in porto, contenente i dati precisi e la posizione dei container sulla nave;
- le informazioni indicanti la posizione che i container in export dovranno occupare sulla nave.

Tali informazioni vengono comunicate al terminal con largo anticipo, rispetto all'arrivo delle navi, mediante l'utilizzo di standard internazionali, come EDIFACT (*Electronic Data Interchange For Administration, Commerce and Transport*).
In particolare in questo capitolo si vuole descrivere l'implementazione di un algoritmo per l'ottimizzazione degli spostamenti di ogni RTG sul blocco di contenitori di competenza. Infatti è necessario che l'ordine in cui i contenitori vengono prelevati riduca al minimo sia gli spostamenti della RTG lungo il blocco, sia il numero delle operazioni di *rehandling*, ovvero delle operazioni improduttive. Il numero di operazioni improduttive aumenta, infatti se l'ordine in cui i contenitori sono prelevati prevede, ad esempio, che per prelevare un contenitore vengano spostati temporaneamente altri contenitori.

Costruttore	Velocità della gru senza carico	Velocità del carrello senza carico
Liebherr	130 [m/min]	70 [m/min]
Konecranes	135 [m/min]	70 [m/min]

Tabella 3.1: Velocità di spostamento di RTG

Le informazioni contenute in Tabella 3.1 mostrano come, date le modeste velocità di traslazione delle RTG, sia importante intervenire per minimizzarne gli spostamenti al fine di ridurre la durata totale delle operazioni di carico.

3.2 Gli algoritmi e la loro implementazione

Il problema di scheduling con tempi di setup dipendenti dalla sequenza Il problema di scheduling su macchina singola consiste nel determinare il sequenziamento delle lavorazioni (*job*) che essa deve eseguire in modo da ottimizzare una qualche misura di prestazione. Esistono svariati problemi di scheduling, modelli per cercare di modellizzare la realtà produttiva; i problemi di scheduling con tempi di setup dipendenti dalla sequenza sono un particolare sottoinsieme che descrive i casi in cui il tempo di processamento di un job non può inglobare anche il tempo di allestimento della macchina per eseguire tale lavorazione. Oltre al tempo impiegato effettivamente per effettuare il job i, p_i, è necessario infatti considerare il tempo di setup, ossia il tempo impiegato per allestire la macchina per effettuare il job i; nel caso in cui i tempi di setup dipendano dal job precedente è possibile rappresentarli mediante una matrice S_{ij}, in generale non simmetrica, in cui l'elemento s_{ij} è il tempo impiegato per preparare la macchina per eseguire il job j, supposto che abbia terminato il job i; siano n i job che la macchina deve eseguire, tale matrice ha dimensione *n+1* in quanto esso deve tenere conto anche dello stato iniziale della macchina; il Makespan C_{max} è dato dal massimo *completion time* C_j, e quindi è funzione dei tempi di setup

$$C_{max} = max_{1<j<n}\{C_j\} = \sum_{j=1}^{n} p_j + \sum_{i=0}^{n} s_{i,i+1}$$

$$min\, C_{max} = min \sum_{i=0}^{n} s_{i,i+1}$$

Se si suppone di associare ad ogni *job* una città, che le distanze fra le città corrispondano ai tempi di setup fra i *job* e che il tempo di processamento p_i del *job* i-esimo corrisponda al tempo di visita della città i-esima, il problema di scheduling appena descritto è equivalente al problema del commesso viggiatore (TSP *Travelling Salesman Problem*), con *n+1* città.

La similitudine con il problema di scheduling Se si considera il problema argomento di questo capitolo, è possibile notare un'evidente similitudine con il problema di *scheduling* appena descritto. Tale similitudine diventa chiara se si considera ogni container come un *job*, avente un tempo di processamento ossia d_j il

tempo di trasporto tra piazzale e banchina, e se si considera come tempo di setup fra due job consecutivi i e *i+1*, il tempo che la RTG impiega per essere pronta a prelevare il container *i+1* dopo aver prelevato il container i, ossia $s_{i,i+1}$. Il tempo impiegato dalla RTG per essere pronta a prelevare un container è costituito sia dal tempo impiegato dalla RTG per muoversi lungo il blocco fino alla posizione del container da prelevare, sia dal tempo impiegato in *rehandling* necessari per rimuovere container che sovrastano il container da prelevare. Tale tempo di setup è chiaramente dipendente dalla sequenza con cui i contenitori vengono prelevati, in quanto il tempo impiegato dalla RTG per essere pronta a prelevare il container k dipende sia dalla posizione del container *k-1*, sia dal numero di container che sovrastano il container k. É quindi possibile affrontare il problema dell'ottimizzazione delle operazioni di carico delle navi come se fosse un problema di scheduling con tempi di setup dipendenti dalla sequenza; quindi il problema verrà rappresentato tramite un grafo *G(V,E)* in cui V è l'insieme dei vertici, a cui corrispondono i job/container, ed E è l'insieme degli archi, i cui pesi sono dati da una matrice di adiacenza S_{ij} di dimensione *n+1*, tale che $s_{i,i+1}$ è il tempo che la RTG impiega per essere pronta per prelevare il container *i+1*, supposto che abbia terminato di prelevare il container *i+1*.

L'algoritmo swap Dal momento che il TSP, a cui il problema di sequenziamento è ricondotto, è un problema NP-hard, per effettuare l'ottimizzazione degli spostamenti delle RTG si è scelto di orientarsi verso un algoritmo euristico. L'algoritmo implementato si ispira all'algoritmo *swap*, utilizzato per risolvere il TSP, anche se con qualche necessaria caratterizzazione. Gli algoritmi di Ricerca Locale (*Local Search*), come l'algoritmo *swap*, data una soluzione iniziale x_0, ricercano la soluzione ottima costruendo un intorno di x_0, *N(0)*, all'interno di tale intorno ricercano la soluzione migliore x_1, se tale soluzione fornisce un valore migliore della funzione obiettivo, x_1 diventa la soluzione corrente nell'intorno della quale si ricerca una soluzione migliore, l'algoritmo si ferma quando non riesce più ad effettuare miglioramenti della funzione obiettivo. La tecnica con cui si genera l'intorno della soluzione corrente è ciò che differenzia gli algoritmi della Ricerca Locale. Chiaramente questo tipo di algoritmi terminano facilmente in un ottimo locale, ma non globale.

L'algoritmo implementato prende in input la sequenza di carico della nave, ossia la lista ordinata dei container che andranno caricati sulla nave, e restituisce una versione migliorata di tale sequenza. Il funzionamento dell'algoritmo è riassumibile nei seguenti step:

- inizializzazione, sia *sequenzaCarico* la sequenza di carico della nave, contenente n container ciascuno identificato dalla sua posizione (baia e fila) e dal numero di container che lo sovrastano (ossia che hanno stessa baia e stessa fila e si trovino ad un tiro superiore) e sia L la durata del ciclo, ossia il tempo impiegato per effettuare l'intera operazione di carico;

- in funzione della sequenza, per ogni container, viene calcolato e memorizzato in *numerocontaineropra* il numero di container che dovranno essere spostati perchè ogni container possa essere prelevato (tale numero non coincide con il numero di container che sovrastano il container ad esempio nel caso che essi vengano prelevati prima del container considerato);

- la sequenza *sequenzaCarico* viene scorsa e per ogni coppia (i, j) di container viene calcolata la variazione *delta* della durata del ciclo causata dallo scambio di i e j.;

- se *delta* è negativo si effettua lo scambio dei container i e j, in quanto esso genera un miglioramento della sequenza, e si rinizia a scorrere la sequenza dal principio, in caso contrario l'algoritmo contiunua a scorrere i contenitori fino al termine della sequenza;

- l'algoritmo termina quando riesce a scorrere l'intera sequenza *sequenzaCarico* e nessuno scambio effettuato genera un miglioramento in termini di durata della sequenza.

La porzione di codice sottostante mostra come gli step appena descritti siano stati implementati in C#.

```
1 int delta = 0;
2 Boolean migliorato = true;
3 while (migliorato)
4 {
```

```
5  migliorato = false;
6  for (int i = 1; i < sequenzaCarico.numeroContainer - 1; i++)
7  {
8  for (int j = i + 2; j < sequenzaCarico.numeroContainer - 2; j++)
9  {
10    sequenzaCarico.aggiornacontainerepraSequenza();
11    delta = -sequenzaCarico.calcolaPesoArco(i - 1, i)
12    - sequenzaCarico.calcolaPesoArco(i, i + 1)
13    - sequenzaCarico.calcolaPesoArco(j - 1, j)
14    - sequenzaCarico.calcolaPesoArco(j, j + 1)
15    + sequenzaCarico.calcolaPesoArco(i - 1, j)
16    + sequenzaCarico.calcolaPesoArco(j, i + 1)
17    + sequenzaCarico.calcolaPesoArco(j - 1, i)
18    + sequenzaCarico.calcolaPesoArco(i, j + 1);
19    for (int h = i + 1; h < j; h++)
20    {
21     if ((sequenzaCarico.selezionaContainer(h).baia ==
22     sequenzaCarico.selezionaContainer(j).baia) &&
23     (sequenzaCarico.selezionaContainer(h).fila ==
24     sequenzaCarico.selezionaContainer(j).fila) &&
25     (sequenzaCarico.selezionaContainer(h).containeropraReali >
26     sequenzaCarico.selezionaContainer(j).containeropraReali))
27     {
28       delta -= Sequenza.tempoHandling;
29     }
30     else if ((sequenzaCarico.selezionaContainer(h).baia ==
31      sequenzaCarico.selezionaContainer(j).baia) &&
32      (sequenzaCarico.selezionaContainer(h).fila ==
33      sequenzaCarico.selezionaContainer(j).fila) &&
34      (sequenzaCarico.selezionaContainer(h).containeropraReali <
35      sequenzaCarico.selezionaContainer(j).containeropraReali))
36      {
37        delta += Sequenza.tempoHandling;
38     }
39     if ((sequenzaCarico.selezionaContainer(h).baia ==
40     sequenzaCarico.selezionaContainer(i).baia) &&
41     (sequenzaCarico.selezionaContainer(h).fila ==
42     sequenzaCarico.selezionaContainer(i).fila) &&
43     (sequenzaCarico.selezionaContainer(h).containeropraReali >
44     sequenzaCarico.selezionaContainer(i).containeropraReali))
45     {
46        delta += Sequenza.tempoHandling;
47     }
48     else if ((sequenzaCarico.selezionaContainer(h).baia ==
49     sequenzaCarico.selezionaContainer(i).baia) &&
50     (sequenzaCarico.selezionaContainer(h).fila ==
51     sequenzaCarico.selezionaContainer(i).fila) &&
```

```
52      (sequenzaCarico.selezionaContainer(h).containeropraReali <
53      sequenzaCarico.selezionaContainer(i).containeropraReali))
54      {
55        delta -= Sequenza.tempoHandling;
56      }
57    }
58    if (delta < 0)
59    {
60      lunghezzaCiclo += delta;
61      migliorato = true;
62      sequenzaCarico.scambioContainer(i, j);
63      sequenzaCarico.aggiornacontaineropraSequenza();
64      j = sequenzaCarico.numeroContainer;
65    }
66  }
67    if (migliorato)
68    {
69      i = sequenzaCarico.numeroContainer;
70    }
71  }
72  }
```

Durante l'esplorazione dell'intorno della soluzione corrente, mediante gli *swap*, l'algoritmo calcola la variazione *delta* causata da ogni *swap*, considerando quali archi vengono eliminati e quali vengono introdotti effettuando uno scambio; inoltre la sequenza viene scorsa per tenere conto di eventuali *rehandling* introdotti o rimossi per effetto dello scambio. Il calcolo della variazione *delta* dovuto allo scambio del container i e j viene effettuato nel seguente modo:

- vengono rimossi gli archi che collegano il nodo i e il nodo j con i rispettivi nodi precessori e successori $i-1$, $i+1$ e $j-1$, $j+1$;
- vengono inseriti gli archi che collegano il nodo j con i nodi $i-1$ e $i+1$ e il nodo i con i nodi $j-1$ e $j+1$;
- vengono scorsi i container compresi tra i+1 e j per verificare eventuali operazioni di *rehandling* inserite o rimosse.

In figura è mostrato come avviene l'eliminazione e l'inserimento di archi nel caso di uno *swap* tra i container B ed E.

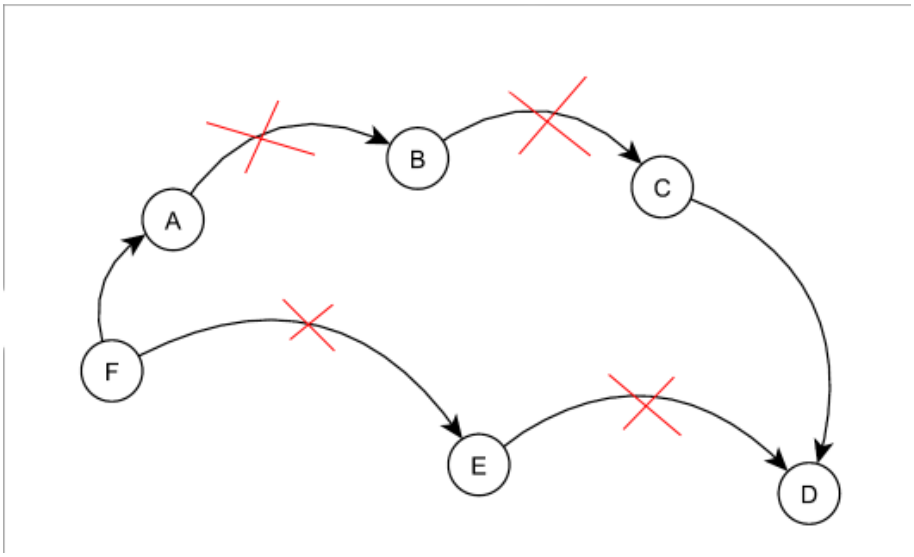

Figura 3.1: Gli archi A-B, B-C, F-E e E-D vengono rimossi

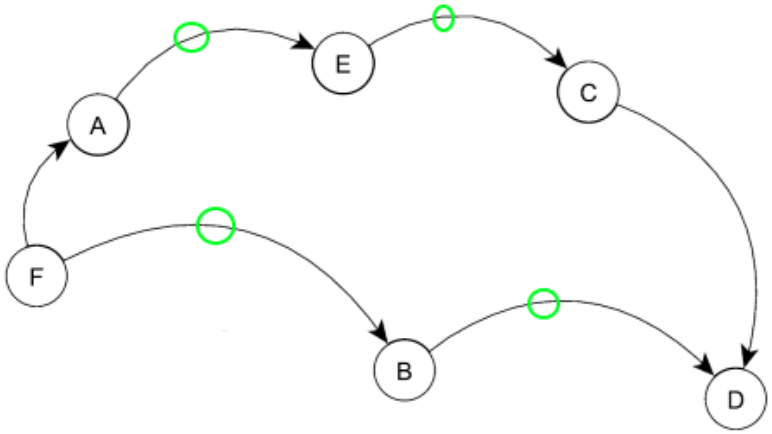

Figura 3.2: Gli archi A-E, E-C, F-B e B-D vengono inseriti

Per calcolare l'effettiva durata delle operazioni di carico inclusi i tempi di trasporto di ogni container fino alla banchina, una volta terminato l'algoritmo, si è implementata la subroutine *calcolaLunghezzaCicloRalle* la quale utilizza, per effettuare l'assegnazione alle ralle delle missioni di trasporto dei container, l'algoritmo *greedy* descritto nel capitolo precedente.

```
1 public double calcolaLunghezzaCicloRalle()
2 {
3  double l = this.selezionaContainer(0).distanzaDaBanchina;
4  int k = 3;
5  int[,] M = this.assegnaContainerRalle(k);
```

```
6   // matrice di assegnazione dei job alle ralle
7   double pesoArco = 0; // il peso dell'arco  tiene conto
8         // degli spostamenti della RTG e del tempo
9         // necessario per prelevare un container
10
11  for (int i = 0; i < this.numeroContainer - 1; i++)
12  {
13   pesoArco = this.calcolaPesoArco(i, i + 1);
14   int ralla = this.selezionaRallaTrasportatrice(i + 1, M, k);
15   double tempoArrivo = this.calcolaTempoArrivo(i + 1, ralla, M);
16   if (l < tempoArrivo)
17   {
18    if (tempoArrivo > pesoArco)
19
20    {
21     l += tempoArrivo - l;
22    }
23    else
24    {
25     l += pesoArco;
26    }
27   }
28   else
29   {
30    l += pesoArco;
31   }
32  }
33  return l;
34 }
```

L'algoritmo swap enhanced Dall'analisi computazionale svolta sull'algoritmo *swap*, mostrata nella sezione successiva, è risultato che tale algoritmo risulta inefficente in termini di tempo di elaborazione. Per questo motivo si è scelto di realizzare una versione *enhanced* volta a diminuire sensibilmente il tempo di elaborazione. Il motivo dell'inefficenza dell'algoritmo *swap* è da ricercarsi nella tecnica di esplorazione dell'intorno della soluzione corrente utilizzata; infatti, come si è mostrato in precedenza l'intorno esplorato è costituito da tutti i possibili scambi dei container presenti nella sequenza di carico. L'algoritmo *swap enhanced* limita l'esplorazione dell'intorno della soluzione corrente, considerando solo lo scambio di ogni container con il suo successore nella sequenza di carico. Tale scambio può portare un miglioramento della sequenza in quanto la matrice $S_{i,j}$ che contiene i tempi di setup è una matrice

asimmetrica ($s_{i,i+1} \neq s_{i+1,i}$). Di seguito è riportato il codice relativo all'algoritmo *swap enhanced*.

```
Boolean migliorato = true;
while (migliorato)
{
 migliorato = false;
 for (int i = 1; i < sequenzaCarico.numeroContainer - 2; i++)
 {
  int j = i + 1;
  sequenzaCarico.aggiornacontaineropraSequenza();

  delta = -sequenzaCarico.calcolaPesoArco(i - 1, i)
    -sequenzaCarico.calcolaPesoArco(i, j)
   - sequenzaCarico.calcolaPesoArco(j, j + 1)
   + sequenzaCarico.calcolaPesoArco(i - 1, j)
   + sequenzaCarico.calcolaPesoArco(i, j + 1)
   + sequenzaCarico.calcolaPesoArco(j, i);

  if ((sequenzaCarico.selezionaContainer(i).baia ==
       sequenzaCarico.selezionaContainer(j).baia) &&
    (sequenzaCarico.selezionaContainer(i).fila ==
    sequenzaCarico.selezionaContainer(j).fila) &&
    (sequenzaCarico.selezionaContainer(i).containeropraReali >
    sequenzaCarico.selezionaContainer(j).containeropraReali))
  {
  delta -= Sequenza.tempoHandling;
  }
  else if ((sequenzaCarico.selezionaContainer(i).baia ==
      sequenzaCarico.selezionaContainer(j).baia) &&
   (sequenzaCarico.selezionaContainer(i).fila ==
    sequenzaCarico.selezionaContainer(j).fila) &&
   (sequenzaCarico.selezionaContainer(i).containeropraReali <
    sequenzaCarico.selezionaContainer(j).containeropraReali))
  {
  delta += Sequenza.tempoHandling;
  }
  if (delta < 0)
  {
  lunghezzaCiclo += delta;
  migliorato = true;
  sequenzaCarico.scambioContainer(i, j);
  sequenzaCarico.aggiornacontaineropraSequenza();
  }
  if (migliorato)
  {
  i = sequenzaCarico.numeroContainer;
```

```
45    }
46   }
47  }
```

3.3 Analisi Computazionale

Per verificare l'efficenza con la quale lavorano gli algoritmi sviluppati si è scelto di effettuare un'analisi volta a verificare come aumentano i tempi di elaborazione degli algoritmi in funzione della dimensione delle istanze del problema ovvero del numero di container che devono essere prelevati e caricati a bordo delle navi. Per generare sequenze di carico verosimili si è generato un piazzale di test, avente 5 file, 33 baie e fino a 5 tiri; per generare ogni sequenza di carico vengono prelevati N container dal piazzale, con probabilità decrescente in funzione del tiro a cui è situato ogni container. Come strumento per effettuare l'analisi si è utilizzato Microsoft Visual Studio 2013 su un laptop dotato di un processore AMD A8 - 4500M 1.9-2.8 GHz e 8 Gb di RAM.

```
1
2  int N = 500;
3  double[] Setprobability = { 1, 0.95, 0.95, 0.80, 0.70, 0.60 };
4  List<Container> sequenzaCarico = new List<Container>();
5  int[, ,] prelevatiDaPiazzale = new int[5, 33, 5];
6  while(sequenzaCarico.Count<N)
7   {
8   int fila = rd.Next(5);
9   int baia = rd.Next(33);
10  int tiro = rd.Next(5);
11
12
13  if ((tiro == 3 || piazzalePresi[fila, baia, tiro + 1] == 1) &&
14      piazzalePresi[fila, baia, tiro] != 1)
15  {
16   double probability = rd.NextDouble();
17
18  if (probability <= Setprobability[tiro] )
19  {
20   temp[fila, baia, tiro] = 1;
21   sequenzaCarico.Add(piazzale[ fila,baia, tiro]);
22   prelevatiDaPiazzale[fila, baia, tiro] = 1;
23  }
24  else
```

```
25  {
26    prelevatiDaPiazzale[fila, baia, tiro] = 0;
27  }
28  }
29 }
30 sequenzaCarico.Add(sequenzaCarico[0]);
```

Nell'array *Setprobability* sono memorizzate le probabilità di prelievo di ogni container in funzione del tiro a cui esso è situato, ad ogni iterazione del *loop* viene selezionata la tripla (fila,baia,tiro) e in funzione delle probabilità memorizzate in *Setprobability* il container viene prelevato. Le tabelle seguenti mostrano i tempi di elaborazione, misurati in secondi, degli algoritmi *swap* e *swap enhanced* al variare del numero N di container che devono essere caricati sulla nave. Al variare di N si sono effettuate 10 repliche e si è misurato il tempo impiegato dall'algoritmo a convergere sulla soluzione migliore. Gli stessi risultati sono proposti anche rappresentati mediante grafici a dispersione, nei quali in ascissa è indicato il numero N di container ed in ordinata i tempi di elaborazione degli algoritmi misurati in secondi.

N	50	100	150	200	250	300	350
Rep. 1	0,156	6,861	42,665	117,164	496,669	1480,110	4427,553
Rep. 2	0,125	2,142	31,928	304,229	434,532	1572,039	4832,038
Rep. 3	0,359	4,142	22,410	193,588	558,913	1681,464	4036,577
Rep. 4	0,250	4,907	38,317	169,713	551,701	1386,688	3812,394
Rep. 5	0,156	5,003	32,585	180,020	434,419	1792,115	4661,448
Rep. 6	0,141	2,954	46,721	130,199	477,479	1716,051	3777,810
Rep. 7	0,125	1,610	24,652	169,383	675,221	1317,776	4139,192
Rep. 8	0,141	3,298	45,805	181,678	395,419	1764,708	4824,496
Rep. 9	0,141	3,542	23,111	153,666	650,733	1999,188	4611,185
Rep. 10	0,109	2,375	30,786	74,148	452,790	2114,175	3326,864
Media	0,169	3,972	34,695	162,814	511,322	1664,039	4261,555

Tabella 3.2: Algoritmo swap, tempi di elaborazione [s]

N	50	100	150	200	250	300	350	400	450	500
Rep. 1	0,031	0,109	0,766	2,985	5,641	12,423	34,004	62,054	69,855	158,468
Rep. 2	0,016	0,156	1,094	3,438	7,980	20,143	30,871	58,960	90,351	141,382
Rep. 3	0,000	0,172	0,734	2,813	8,329	21,042	38,371	46,178	90,800	124,501
Rep. 4	0,000	0,188	1,188	2,860	7,813	14,736	39,931	48,785	88,598	158,113
Rep. 5	0,016	0,187	0,734	1,781	9,798	16,549	25,121	44,612	79,858	100,242
Rep. 6	0,016	0,172	0,703	2,704	6,063	17,982	30,401	51,967	85,249	132,845
Rep. 7	0,016	0,219	0,781	3,578	5,533	18,291	32,323	39,567	89,366	186,217
Rep. 8	0,016	0,297	0,797	4,094	6,574	12,736	29,942	56,506	68,797	161,523
Rep. 9	0,000	0,156	0,844	3,407	7,251	13,642	24,506	55,985	69,271	178,624
Rep. 10	0,000	0,188	0,813	3,063	7,329	17,414	26,560	51,966	71,711	181,491
Media	0,011	0,184	0,845	3,072	7,231	16,496	31,203	51,658	80,386	152,341

Tabella 3.3: Algoritmo swap enhanced, tempi di elaborazione [s]

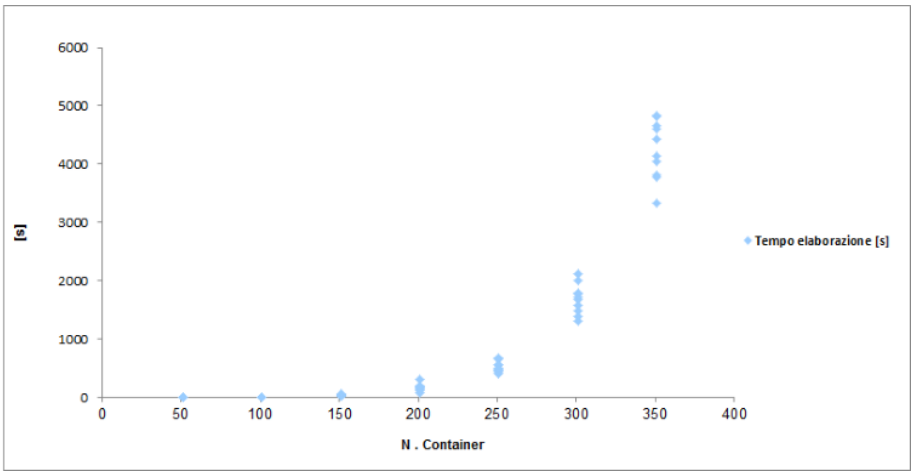

Figura 3.3: Algoritmo swap, tempi di elaborazione

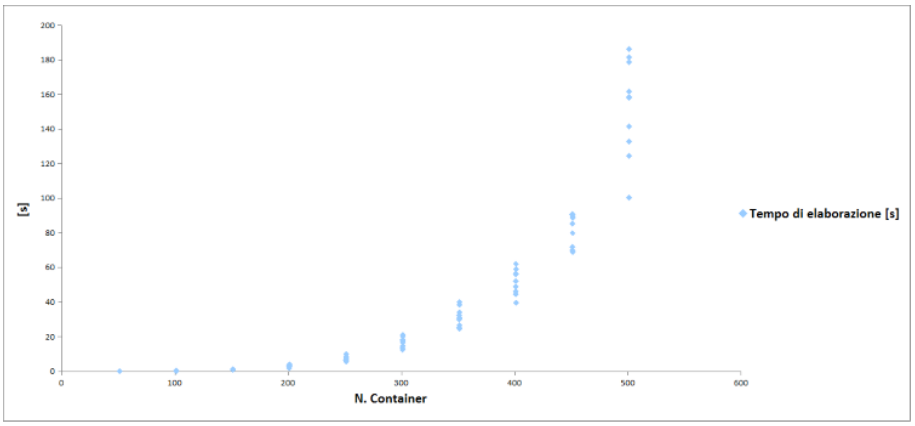

Figura 3.4: Algoritmo swap enhanced, tempi di elaborazione

Per effettuare un analisi prestazionale dei due algoritmi si sono inoltre scelti alcuni indici di performance sulla base dei quali si è effettuato un confronto tra i due algoritmi. Gli indici di performance scelti sono:

- il tempo impiegato dalla RTG per prelevare tutti i container della sequenza di carico;
- la durata totale dell'operazione di carico, compreso il trasporto dei container fino alla banchina;
- il numero di rehandling compiuti dalla RTG.

All'aumentare del numero di container si sono misurati i risultati ottenuti dai due algoritmi sulle stesse istanze, si è quindi calcolata la differenza percentuale tra l'algoritmo *enhanced swap* ed *swap* degli indici misurati. I risultati ottenuti sono stati rappresentati mediante grafici a dispersione, nei quali in ascissa è indicato il numero N di container ed in ordinata la differenza percentuale tra gli indici di performance scelti.

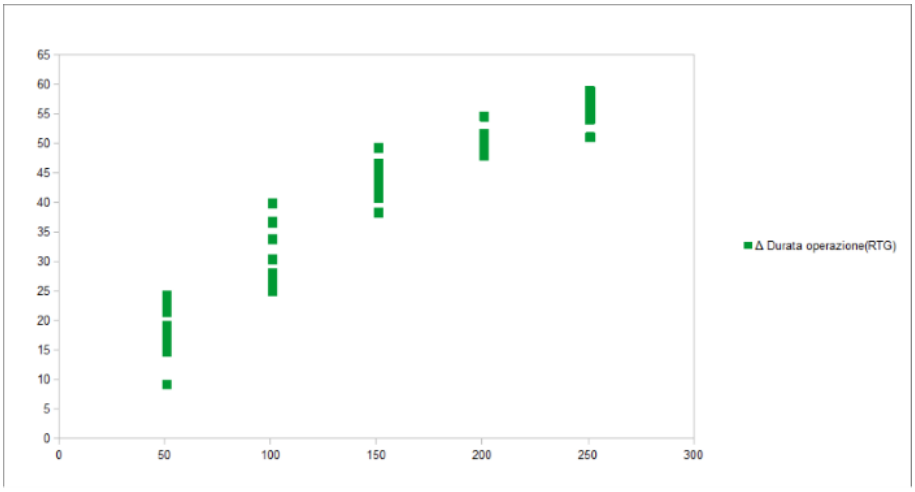

Figura 3.5: Differenza percentuale tra l'algoritmo *enhanced swap* ed *swap* del il tempo impiegato dalla RTG per prelevare tutti i container

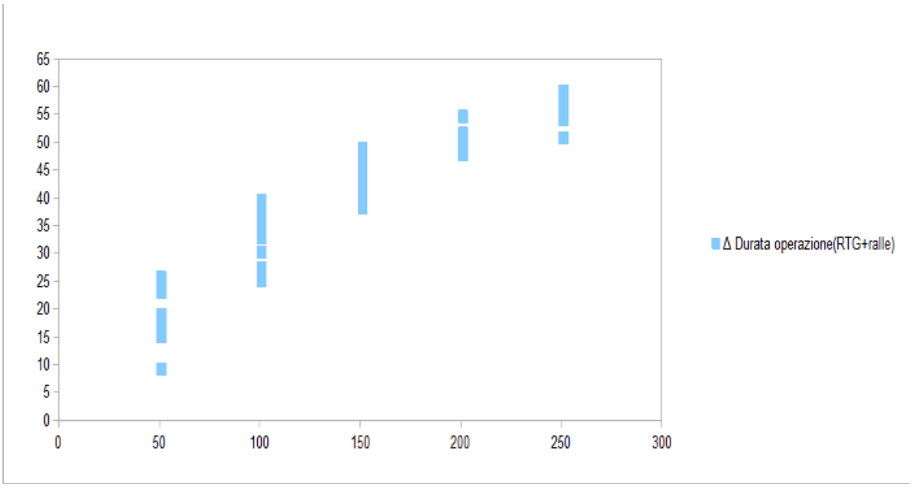

Figura 3.6: Differenza percentuale tra l'algoritmo *enhanced swap* ed *swap* del durata dell'operazione di carico, compreso il trasporto

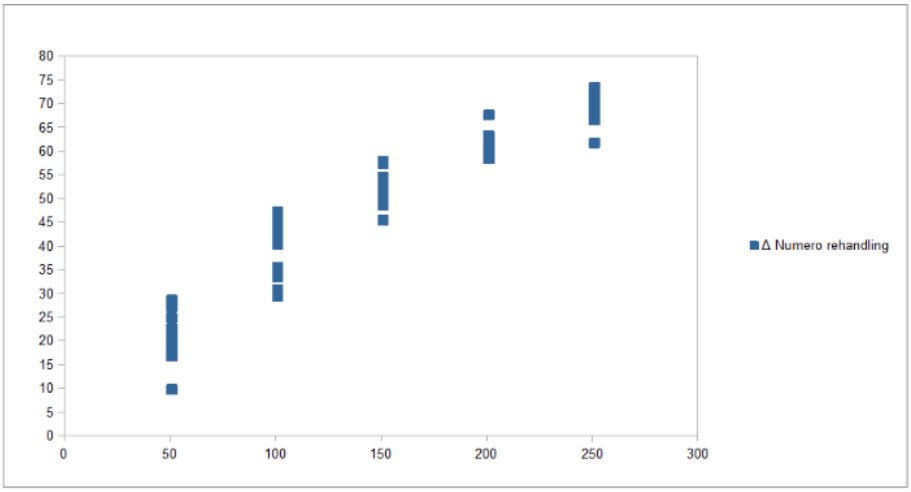

Figura 3.7: Differenza percentuale tra l'algoritmo *enhanced swap* ed *swap* del numero di *rehandling*

3.3.1 Risultati

I risultati ottenuti mostrano che, come atteso, il numero di container influisce sul tempo di elaborazione di entrambi gli algoritmi. Il fattore che per la sua influenza risulta essere però fondamentale è sicuramente la tecnica di esplorazione dell'intorno delle soluzioni. Inoltre dal confronto prestazionale effettuato tra i due algoritmi si evince che i risultati ottenuti dall'algoritmo *enhanced swap* sono sempre meno

soddisfacenti in termini di prestazioni rispetto a quelli ottenuti dall'algoritmo *swap*; inoltre al crescere della dimensione delle istanze dei problemi da risolvere la differenza percentuale tra gli indici di performance misurati cresce notevolmente. Mediante questi risultati è possibile effettuare le seguenti considerazioni:

- l'algoritmo *swap* non è una soluzione implementabile in condizioni di esercizio reali, in quanto all'aumentare della dimensione delle istanze dei problemi il tempo di elaborazione esplode, tuttavia i risultati dell'algoritmo in termini di prestazioni sono molto soddisfacenti;

- l'algoritmo *swap enhanced* risulta molto più performante dell'algoritmo *swap*, con tempi che possono essere considerati ragionevoli in un contesto operativo, tuttavia esso, al crescere della dimensione delle istanze dei problemi da risolvere, non fornisce un'ottimizzazione di particolare qualità;

Capitolo 4

Conclusioni

4.0.2 Risultati ottenuti

In questa tesi è stato affrontato il problema del trasferimento dei container dall'area di stoccaggio alla banchina, ponendo particolare attenzione ai container in *export* andando ad analizzare anche il processo di prelievo dai blocchi dei container mediante RTG. In particolare:

- sono stati descritti tre algoritmi euristici (*greedy, reversed greedy e combined greedy*) per risolvere il problema dell'instradamento dei mezzi di trasporto durante le operazioni di scarico e carico. Gli algoritmi *greedy e reversed greedy* applicati, rispettivamente, ad una sequenza di scarico e carico di una nave, sono in grado di fornire la soluzione ottima, come dimostrato in [2], l'algoritmo *combined greedy* fornisce una soluzione non ottimale al problema di instradamento delle ralle durante le operazioni di scarico e carico considerate in serie;

- gli algoritmi descritti sona stati implementati in C# ed è stata effettuata un'analisi computazionale al fine di verificare la dipendenza del tempo di elaborazione degli algoritmi dal numero di container da trasportare e dal numero di ralle assegnate; dall'analisi dei risultati ottenuti è stato riscontrato che l'incremento del tempo di elaborazione all'aumentare del numero di container e/o dei veicoli è modesto, inoltre la maggior parte dei test eseguiti è stata risolta in meno di 10 secondi;

- sono state descritte le fasi che compongono l'operazione di carico di una nave con specifico riferimento all'utilizzo di RTG per il prelievo dei container dai blocchi di stoccaggio; il problema della minimizzazione degli spostamenti e dei *rehandling* delle RTG è stato assimilato ad un problema di *scheduling* con tempi di setup dipendenti dalla sequenza dei *job*; essendo tale problema di *scheduling* rappresentabile mediante un TSP sono stati proposti due algoritmi euristici per la sua soluzione;

- sono stati implementati in C# gli algoritmi *swap* ed *enhanced swap* ed è stata effettuata un'analisi computazionale e prestazionale per verificare le prestazioni di tali algoritmi, all'aumentare della dimensione delle istanze dei problemi da risolvere, sia in termini di tempo di elaborazione sia in termini di specifici indici di performance; dall'analisi svolta è risultato che , anche se l'algoritmo *enhanced swap* è molto più performante in termini di tempo di elaborazione, esso non fornisce un'ottimizzazione particolarmente soddisfacente soprattutto se confrontata con le prestazioni in termini di indici di performance dell'algoritmo *swap*. Tale trade-off risulta essere sempre più marcato al crescere della dimensione delle istanze dei problemi da risolvere, mentre per istanze di dimensioni più contenute esso si riduce anche se non del tutto;

4.0.3 Sviluppi futuri

L'analisi svolta sugli algoritmi *swap* ed *enhanced swap* ha permesso di individuare nel primo di questi due algoritmi un interessante sviluppo di questo lavoro di ricerca; a partire dai risultati assolutamente incoraggianti ottenuti da questo algoritmo sarà infatti possibile:

  - effettuare un' ottimizzazione dei tempi di elaborazione di tale algoritmo elaborando opportuni criteri d'arresto;
  - effettuare un upgrade di tale algoritmo per tenere conto anche del peso dei container e quindi della loro influenza sulla stabilità della nave sulla quale saranno caricati;

– sviluppare una versione di tale algoritmo considerando la presenza di due RTG sullo stesso blocco.

Bibliografia

[1] UNCTAD, United Nations Conference on Trade and Development (2013), *Review of Maritime Transport*.

[2] Ebru K. Bish, Frank Y. Chen, Yin Thin Leong, Barry L. Nelson, JonathanWing Cheong Ng, and David Simchi-Levi, *Dispatching vehicles in a mega container terminal*, OR Spectrum(2005) 27:491-506.

[3] Daniela Ambrosino, Anna Sciomachen, Elena Tanfani, *Stowing a containership: the master bay plan problem*, Trasportation Reasearch Part A: Policy and Practice, 2,(2004) 81-99, Genova.

[4] Daniela Ambrosino,Davide Anghinolfi,Massimo Paolucci,Anna Sciomachen, *An Experimental Comparison of Different Heuristics for the Master Bay Plan Problem*, Experimental Algorithms, Lecture Notes in Computer Science, 2010 Volume 6049/2010,314-325, Genova.

[5] Ilaria Vacca, Matteo Salani, Michel Bierlaire (2010), *Optimization of operations in container terminals: hierarchical vs integrated approaches*, 10th Swiss Transport Research Conference, Ascona, Switzerland.

[6] Larry Sam (1988), *Selecting Container Handling Equipment for Growth*, Presented in Terminal Operators Conference 1988 (TOC1988), Marseille, Francia.

[7] www.worldshipping.org.

[8] H.-O. Gunther and K.-H. Kim (2006), Container terminals and terminal operations, OR Spectrum, 28, 437-445

[9] H.-O. Gunther and K.-H. Kim (2006), Container terminals and terminal operations, OR Spectrum, 28, 437-445

[10] I.F.A. Vis and R. de Koster, (2003) Transshipment of containers at a container terminal: an overview, European Journal of Operational Research, 147, 1-16

[11] D. Steenken, S. Voss, R. Stahlbock (2004), Container terminal operation and operations research - a classification and literature review, OR Spectrum, 26, 3-49

[12] R. Stahlbock and S. Voss (2008), Operations research at container terminals: a literature update, OR Spectrum, 30, 1-52