Università degli Studi di Padova

Ingegneria informatica

Tesi magistrale

Ottimizzazione di un Problema di Packing 3D

Relatore:
Domenico Salvagnin

Laureando:
Simone Nigro

Sommario

Il documento illustra un lavoro svolto in collaborazione con Gibus, un'azienda di Padova. L'obiettivo della tesi consiste nella realizzazione di un software che possa permettere l'ottimizzazione del trasporto delle merci dalla sede centrale ai clienti attraverso la ricerca della miglior disposizione possibile dei colli all'interno di casse. Tale software deve garantire affidabilità e velocità di risposta in modo tale da permettere all'azienda di fornire al cliente un costo veritiero già in fase di preventivo. Sono stati utilizzati due approcci differenti, il primo fa uso del solver CPLEX permettendo l'ottimizzazione attraverso la risoluzione di un modello matematico che rappresenti i vincoli del problema da risolvere. Per quanto riguarda il secondo approccio è stato realizzato un algoritmo euristico che permetta un miglioramento in termini di tempo di risoluzione a discapito di una mancanza di certezza che la soluzione sia ottima.

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

In questo capitolo introduttivo verrà presentata l'azienda e il problema da essa riscontrato. Nell'ultima parte verrà infine presentata la soluzione proposta. Verranno inoltre elencate le ipotesi prese in considerazione e l'obiettivo della tesi.

1.1 L'azienda

Gibus è un'azienda che progetta e realizza tende da sole e pergole, vetrate, sistemi di illuminazione, sistemi audio e riscaldatori. Opera in questo settore dal 1982, portando il Made in Italy nel mondo, l'eccellenza, l'autenticità di una cultura del sole e dello stile italiano nel vivere la luce. I loro prodotti sono il risultato di strategia e design, di creatività, di ricerca ed innovazione. L'azienda ha sede a Saccolongo in provincia di Padova e nell'ufficio operativo vengono organizzati e gestiti gli ordini che verranno poi trasportati ai clienti. Da questa realtà si evince come si debba essere sempre pronti alla comunicazione con i clienti, fornendo loro un servizio adeguato che li soddisfi ma che permetta all'azienda di realizzare un profitto, è da qui che nasce la necessita di un software decisionale di supporto.

1.2 Descrizione del problema

L'azienda ricerca un software che possa permettere in breve tempo una stima delle casse necessarie per poter contenere i colli da spedire al cliente. Supponendo che una cassa abbia un costo proporzionale al suo volume, è necessario minimizzare il costo di spedizione attraverso la riduzione del numero di casse da utilizzare.
Per far ciò occorre disporre i colli nel miglior modo possibile ricercando la soluzione più economica in breve tempo. La velocità del software risulta importante in quanto un preventivo errato presentato al cliente potrebbe portare ad una perdita di profitto per l'azienda.
I colli da spedire hanno caratteristiche differenti in base al loro contenuto. Un collo potrebbe essere non impilabile, non permettendo così che un altro collo possa essere disposto sopra di esso, oppure potrebbe non essere rotabile nel caso in cui una sua rotazione possa portare ad un danneggiamento del contenuto presente al suo interno.

1.3 Soluzione proposta

Per risolvere il problema descritto in precedenza è necessario sviluppare un software che automatizzi la scelta della disposizione di un collo all'interno di una cassa. Per tale realizzazione l'azienda fornisce le seguenti ipotesi di lavoro:

- il software riceve in input le merci da trasportare approssimando i colli a parallelepipedi;
- per la stabilità degli oggetti, nel caso in cui la faccia inferiore di un collo non poggi per intero sulle facce superiori di altri colli sotto di sè o sul fondo della cassa, viene usato del polistirolo rinforzato per colmare i buchi che ne minerebbero l'equilibrio;
- il peso delle merci non viene tenuto in considerazione in quanto non porta problemi di nessun tipo vista le caratteristiche dei colli a disposizione.

Dati in input i file contenenti le dimensioni dei colli da spedire e le casse a disposizione, il software produrrà in output la lista delle casse da dover utilizzare, la disposizione dei colli al proprio interno, il costo totale della spedizione (derivante dalla somma dei costi delle singole casse utilizzate) ed infine una rappresentazione grafica che permetta una comprensione più immediata della disposizione dei colli in ogni cassa.

2

Modellazione del problema

In questo capitolo viene presentata una prima soluzione al problema di ottimizzazione. Tale risoluzione avviene tramite la creazione di un modello matematico e l'utilizzo del software CPLEX. Dopo aver descritto il modello, con i suoi vincoli, si riporta una breve descrizione dei comandi principali utilizzati in fase di programmazione. Infine vengono illustrati i risultati ottenuti dai test svolti.

2.1 Programmazione Lineare Intera

La programmazione lineare intera (PLI) è un problema di programmazione in cui tutte le variabili sono vincolate ad assumere valori interi. In generale tali problemi risultano essere NP-hard, ovvero la loro risoluzione richiede un tempo di calcolo esponenziale rispetto alla dimensione dell'input. La PLI tratta il problema della minimizzazione o massimizzazione di una funzione lineare composta da una o più variabili. La formulazione tipica utilizzata è la seguente:

$$
\begin{aligned}
&min \quad c^T x \\
&s.t. \; Ax \leq b \\
&\qquad 0 \leq x \leq u \\
&\qquad x \; intero
\end{aligned}
$$

dove $c^T x$ rappresenta la funzione obiettivo da minimizzare, c e $u \geq 1$ sono vettori di dimensione n, A è una matrice $m \times n$ e b è un vettore di dimensione m. La variabile x è detta binaria se $u = 1$ altrimenti è intera.
Una tecnica generale utilizzata per la risoluzione di problemi di ottimizzazione combinatoria è il *Branch & Bound*. Tale tecnica permette di scomporre il problema originale in sottoproblemi più semplici da risolvere e di ottimizzare la ricerca evitando di esplorare un percorso decisionale che porterebbe a soluzioni peggiori di quella in possesso.
Se si considerasse per esempio un problema di PLI e si individuasse una soluzione ottima x^* non intera, allora è necessario scegliere una variabile x_h^* frazionaria per costruire due sottoproblemi in cui viene aggiunto il vincolo $x_h \leq \lfloor x_h^* \rfloor$ nel primo e $x_h \geq \lceil x_h^* \rceil$ nel secondo. Questa operazione è detta di *branching* (ramificazione) rispetto alla variabile x_h. A questo punto si risolvono in maniera ricorsiva i due sottoproblemi *figli* dell'originale andando così a definire problemi sempre più vincolati e quindi più semplici da risolvere. In prartica, non è necessario però considerare tutti i nodi presenti nell'albero decisionale in quanto alcuni di questi possono corrispondere a problemi il cui rilassamento continuo risulta essere impossibile. Esiste inoltre un altro criterio chiamato di *bounding* che permette di sfruttare la stima del *lower bound* e nel caso in cui sia maggiore o uguale al valore ottimo individuato fino a quel momento è possibile evitare l'elaborazione del nodo in questione e dei relativi figli che ne verrebbero generati. Questo è possibile perchè il *lower bound* è crescente essendo i problemi sempre più vincolati di padre in figlio.
Un'altra tecnica è il *Branch & Cut*, l'idea di tale metodologia è simile a quella del *Branch & Bound* in cui ad ogni nodo dell'albero decisionale si generano alcuni tagli profondi per ricercare una soluzione intera o un *lower bound* più elevato. Nel caso si sia in grado di generare tagli validi globalmente lungo tutto l'albero decisionale è possibile memorizzarli in un'unica struttura dati detta *pool* di vincoli.

2.2 Modello matematico

Il problema di ottimizzazione che verrà trattato riguarda, come detto nel Capitolo 1, un problema di *packing 3D*. Per poter creare un modello matematico che rappresenti al meglio la natura di questo problema è necessario definire variabili di due tipologie, continue e binarie. Per quanto riguarda le variabili continue si ha:

- x_i : coordinata sull'asse x del vertice in basso a sinistra del collo i;

- y_i : coordinata sull'asse y del vertice in basso a sinistra del collo i;
- z_i : coordinata sull'asse z del vertice in basso a sinistra del collo i.

Per le variabili binarie invece si ha:

- s_i : 1 se il collo i è impilabile, 0 altrimenti;
- v_i : 1 se il collo i è rotabile invertendone l'altezza con la profondità o la larghezza, 0 altrimenti;
- l_{ij} : 1 se il collo i è a sinistra del collo j, 0 altrimenti;
- b_{ij} : 1 se il collo i è dietro al collo j, 0 altrimenti;
- t_{ij} : 1 se il collo j è sopra al collo i, 0 altrimenti;
- p_{ia} : 1 se il collo i è nella cassa a, 0 altrimenti;
- q_a : 1 se viene usata la cassa a, 0 altrimenti;
- k_{ij}^a : 1 se i colli i e j vengono inseriti entrambi nella cassa a, 0 altrimenti;
- α_i : 1 se il collo i è ruotato e ha profondità=d_i, larghezza=h_i e altezza=w_i, 0 altrimenti;
- β_i : 1 se il collo i è ruotato e ha profondità=w_i, larghezza=d_i e altezza=h_i, 0 altrimenti;
- γ_i : 1 se il collo i è ruotato e ha profondità=w_i, larghezza=h_i e altezza=d_i, 0 altrimenti;
- δ_i : 1 se il collo i è ruotato e ha profondità=h_i, larghezza=d_i e altezza=w_i, 0 altrimenti;
- μ_i : 1 se il collo i è ruotato e ha profondità=h_i, larghezza=w_i e altezza=d_i, 0 altrimenti;
- σ_{ij} : 1 se i colli i e j vengono inseriti entrambi nella stessa cassa, 0 altrimenti.

I dati forniti in input sono rappresentati dai seguenti coefficienti:

- w_i : larghezza del collo i;
- d_i : profondità del collo i;
- h_i : altezza del collo i;
- W_a : larghezza della cassa a;
- D_a : profondità della cassa a;
- H_a : altezza della cassa a;
- r_i : profondità del collo i considerando la rotazione;
- g_i : larghezza del collo i considerando la rotazione;
- u_i : altezza del collo i considerando la rotazione;
- c_a : costo della cassa a.

Esistono inoltre altri coefficienti, detti Big M, che permettono, sotto determinate condizioni, di attivare o disattivare i vincoli in cui vengono utilizzati:

- $M_{wa} = M + W_a$
- $M_{da} = M + D_a$
- $M_{ha} = M + H_a$

Sono presenti n colli che vengono identificati con gli indici i, $j \in$ P e m casse, di varie dimensioni e costi, identificate dagli indici $a \in$ B.

Date queste informazioni iniziali è possibile costruire il seguente modello matematico:

$$min \sum_{a=1}^{m} c_a q_a \tag{2.1}$$

$$b_{ij}, k_{ij}^a, l_{ij}, p_{ia}, q_a, s_i, t_{ij}, v_i, \alpha_i, \beta_i, \gamma_i, \delta_i, \mu_i, \sigma_{ij} \in \{0,1\} \quad \forall i,j \in P \quad i \neq j \quad \forall a \in B \tag{2.2}$$

$$c_a, d_i, r_i, g_i, h_i, u_i, w_i, x_i, y_i, z_i, D_a, H_a, W_a \in \{R^+\} \quad \forall i \in P \quad \forall a \in B \tag{2.3}$$

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + t_{ij} + t_{ji} \geq \sigma_{ij} \quad \forall i,j \in P \quad i \neq j \tag{2.4}$$

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + 2t_{ij} + 2t_{ji} \leq 2\sigma_{ij} \quad \forall i,j \in P \quad i \neq j \tag{2.5}$$

$$x_i + r_i + Mp_{ia} \leq D_a p_{ia} + M \quad \forall i \in P \quad \forall a \in B \tag{2.6}$$

$$y_i + g_i + Mp_{ia} \leq W_a p_{ia} + M \quad \forall i \in P \quad \forall a \in B \tag{2.7}$$

$$z_i + u_i + Mp_{ia} \leq H_a p_{ia} + M \quad \forall i \in P \quad \forall a \in B \tag{2.8}$$

$$\sum_{a=1}^{m} p_{ia} = 1 \quad \forall i \in P \tag{2.9}$$

$$k_{ij}^a \geq p_{ia} + p_{ja} - 1 \quad \forall i,j \in P \quad i \neq j \quad \forall a \in B \tag{2.10}$$

$$4k_{ij}^a \leq p_{ia} + p_{ja} + 2 \quad \forall i,j \in P \quad i \neq j \quad \forall a \in B \tag{2.11}$$

$$\sigma_{ij} = \sum_{a=1}^{m} k_{ij}^a \quad \forall i,j \in P \quad i \neq j \tag{2.12}$$

$$p_{ia} \leq q_a \quad \forall i \in P \quad \forall a \in B \tag{2.13}$$

$$x_i - x_j + M_{wa} b_{ij} \leq M_{wa} - r_i \quad \forall i,j \in P \quad i \neq j \quad \forall a \in B \tag{2.14}$$

$$y_i - y_j + M_{da} l_{ij} \leq M_{da} - g_i \quad \forall i,j \in P \quad i \neq j \quad \forall a \in B \tag{2.15}$$

$$z_i - z_j + M_{ha} t_{ij} \leq M_{ha} - u_i \quad \forall i,j \in P \quad i \neq j \quad \forall a \in B \tag{2.16}$$

$$r_i = d_i(1 - \beta_i - \gamma_i - \delta_i - \mu_i) + w_i(\beta_i + \gamma_i) + h_i(\delta_i + \mu_i) \quad \forall i \in P \tag{2.17}$$

$$g_i = d_i(\beta_i + \delta_i) + w_i(1 - \alpha_i - \beta_i - \gamma_i - \delta_i) + h_i(\alpha_i + \gamma_i) \quad \forall i \in P \tag{2.18}$$

$$u_i = d_i(\gamma_i + \mu_i) + w_i(\alpha_i + \delta_i) + h_i(1 - \alpha_i - \gamma_i - \delta_i - \mu_i) \quad \forall i \in P \tag{2.19}$$

$$t_{ij} \leq s_i \quad \forall i,j \in P \quad i \neq j \tag{2.20}$$

$$\alpha_i + \beta_i + \gamma_i + \delta_i + \mu_i \leq 1 \quad \forall i \in P \tag{2.21}$$

$$\alpha_i + \gamma_i + \delta_i + \mu_i \leq v_i \quad \forall i \in P \tag{2.22}$$

$$(\beta_i + \gamma_i) w_i + Mp_{ia} \leq M_{da} \quad \forall i \in P \quad \forall a \in B \tag{2.23}$$

$$(\alpha_i + \delta_i) w_i + Mp_{ia} \leq M_{ha} \quad \forall i \in P \quad \forall a \in B \tag{2.24}$$

$$\sum_{a=1}^{m} q_a w_a d_a h_a \geq \sum_{i=1}^{n} w_i d_i h_i \tag{2.25}$$

Nel dettaglio, è possibile dare le seguenti interpretazioni:

- funzione obiettivo (2.1) : minimizza il costo totale considerando la somma dei costi delle casse che vengono utilizzate;
- vincoli (2.4) (2.5) : presi due colli uno è dietro all'altro e/o alla sinistra dell'altro, oppure uno dei due è sopra l'altro;
- vincolo (2.6) : presa la coordinata x del vertice sinistro più arretrato e sommandoci la profondità del collo ottengo la coordinata x dell'angolo sinistro più avanzato che deve essere contenuto nella dimensione della cassa;
- vincolo (2.7) : presa la coordinata y del vertice sinistro più arretrato e sommandoci la larghezza del collo ottengo la coordinata y dell'angolo destro più arretrato che deve essere contenuto nella dimensione della cassa;
- vincolo (2.8) : presa la coordinata z del vertice sinistro più arretrato e sommandoci l'altezza del collo ottengo la coordinata z dell'angolo sinistro più in alto che deve essere contenuto nella dimensione della cassa;
- vincolo (2.9) : un collo deve essere presente solo in una cassa;
- vincoli (2.10), (2.11) e (2.12): due colli presenti entrambi nella cassa a allora sono presenti nella stessa cassa;
- vincolo (2.13) : se viene inserito un collo in una cassa allora la cassa viene utilizzata;
- vincolo (2.14) : se $b_{ij} = 1$ allora il collo i è dietro di j, altrimenti il vincolo viene disattivato;
- vincolo (2.15) : se $l_{ij} = 1$ allora il collo i è a sinistra j, altrimenti il vincolo viene disattivato;
- vincolo (2.16) : se $t_{ij} = 1$ allora il collo i è sotto j, altrimenti il vincolo viene disattivato;
- vincolo (2.17) : corrisponde alla profondità del collo considerando la rotazione;
- vincolo (2.18) : corrisponde alla larghezza del collo considerando la rotazione;

- vincolo (2.19) : corrisponde all'altezza del collo considerando la rotazione;
- vincolo (2.20) : se un collo non è impilabile non è possibile metterci niente sopra;
- vincolo (2.21) : considerando le rotazioni possibili il risultato delle dimensioni del collo è unico;
- vincolo (2.22) : un collo non rotabile può raggiungere solo la configurazione β in caso di rotazione;
- vincolo (2.23) : un collo non può avere una profondità $= w_i$ dopo la rotazione se la larghezza del collo i è maggiore alla profondità della cassa a in cui è stato inserito, altrimenti il vincolo viene disattivato;
- vincolo (2.24) : un collo non può avere un'altezza $= w_i$ dopo la rotazione se la larghezza del collo i è maggiore all'altezza della cassa a in cui è stato inserito, altrimenti il vincolo viene disattivato;
- vincolo (2.25) : il volume delle casse utilizzate non deve essere superato dal volume totale dei colli da spedire.

I vincoli (2.23) e (2.24) sono stati inseriti per la specificità del problema affrontato, in quanto molti colli a disposizione hanno una forma allungata che ne limita le possibili rotazioni.

2.3 Implementazione base in CPLEX

CPLEX ha la possibilità di lavorare in due modi differenti: il primo attraverso comandi utilizzati da interattivo, il secondo mediante alcune routine di libreria. In questa parte del capitolo verranno trattati alcuni dettagli implementativi svolti nella fase di programmazione, inerenti al secondo approccio. Lo scopo è quello di creare un programma che possa leggere dati di input forniti da riga di comando e da file di testo esterni, crei un modello matematico che risponda alle esigenze richieste, e infine lo ottimizzi.

2.3.1 Struttura "instance"

"Instance" è una struttura dati che contiene tutte le informazioni di input o i parametri di CPLEX utili a risolvere il problema e dove verrà salvata la soluzione. Quando nel *main*

viene utilizzata l'istruzione *instance inst*, viene chiesto al sistema operativo di ritagliare in memoria una struttura di tipo istanza, contenente al suo interno tutte le variabili sopracitate, il cui nome è *inst*. In questo modo si crea la struttura ma al momento tutti i campi risultano essere non inizializzati. Inizialmente vien assegnato a tutti questi dati un valore di default, per esempio il *time limit* viene posto ad infinito utilizzando una costante di CPLEX, *CPX_INFBOUND*. Successivamente vengono riempiti da riga di comando attraverso la funzione *parse_command_line* mentre altri da file esterni attraverso *read_input*.

2.3.2 Lettura dei file di input

Read input è una funzione che permette di leggere dei file di input, in questo caso due. Il primo contiene le dimensioni dei colli e il tipo (che permette di capire se il collo è impilabile o rotabile). Il secondo, invece, contiene le informazioni riguardanti le casse a disposizione, essi sono descritti dalle loro dimensioni interne e dal prezzo (identificato dal suo volume). I nomi di questi file vengono specificati da riga di comando, quindi sono letti dalla funzione *parse_command_line* e sono disponibili nell'istanza. Il *main*, quindi, dopo aver effettuato il parse della riga di comando e salvato i nomi dei file di input, chiama la funzione *read_input* e le passa l'indirizzo dell'istanza. Una volta letti i dati riempie con queste informazioni gli opportuni spazi precedentemente creati.

2.3.3 Ottimizzazione

Questa fase si svolge all'interno di *PACKINGopt* e si suddivide in due fasi: la prima consiste nel chiamare la funzione *build_model* addetta alla costruzione del modello matematico, la seconda invece sarà la vera e propria fase di risoluzione. Inizialmente è necessario creare due oggetti di CPLEX attraverso due sue funzioni caratteristiche:

- *CPXENVptr env = CPXopenCPLEX(&error)*
- *CPXLPptr lp = CPXcreateprob(env, &error, "PACKING")*

La prima definisce un environment di tipo *CPXENVptr* ovvero un puntatore di CPLEX ad un environment attraverso la funzione *CPXopenCPLEX*, se in questo momento avviene un errore viene catturato dalla variabile *error*. Questo environment successivamente viene passato ad un'altra funzione che si occupa di definire il modello lp, al momento ancora vuoto.

2.3.3.1 Costruzione del modello

L'operazione di costruzione del modello viene svolta da *build_model*. Ad essa è necessario passare l'istanza da cui prendere i dati di input, l'environment e lp, ovvero il modello vuoto che poi andrà a riempire.
Per creare una nuova variabile in CPLEX è necessario usare la funzione *CPXnewcols* che aggiunge una colonna al modello. Per creare, per esempio, la variabile p_{ia} è necessario ciclare su tutti i colli i, per i compreso tra 0 e il $n-1$ e su tutte le casse a, per a compreso tra 0 e il $n-1$, definire il nome da assegnarle e infine chiamare la funzione *CPXnewcols* che ad ogni iterazione ne aggiunge una, caratterizzata dal costo che ha nella funzione obiettivo, lower e upper bound, tipo (per esempio binaria o continua) e nome che le è stato assegnato. Dopo aver popolato il modello con un tipo di variabile, lo stesso lavoro verrà ripetuto per le altre.
Terminato l'inserimento delle variabili vengono inseriti i vincoli attraverso la funzione *CPXnewrows* che aggiunge una riga vuota al modello. Si specifica il senso del vincolo (E se equazione, G se maggiore o uguale e L se minore o uguale), un nome e un termine noto. É stata creata così una riga vuota nel modello. Per cambiare i coefficienti di questo vincolo si utilizza un ciclo che modifica quelli diversi da 0.

2.3.3.2 Risoluzione

La fase di risoluzione del modello matematico può essere svolta da interattivo, dopo aver fatto leggere il file LP creato con la funzione *CPXwriteprob*, tramite il comando *opt* da riga di comando. Una volta inviata la richiesta di ottimizzazione, il controllo passa a CPLEX che fornisce dei log a video e una volta trovata la soluzione terminerà. La fase di risoluzione solitamente ha una durata molto lunga, per questo motivo è spesso utile ricorrere all'utilizzo di algoritmi che possono facilitarne l'esecuzione.
Da libreria si lancia l'ottimizzazione con la funzione *CPXmipopt*. Terminata l'ottimizzazione dentro *lp* è presente la soluzione ottima.

3

Algoritmo euristico senza l'utilizzo di CPLEX

Analizzato l'algoritmo basato su CPLEX, in questo capitolo verranno riprodotti i risultati ottenuti senza l'utilizzo di questo software. Per far ciò è necessario affidarsi a dei metodi euristici che permettono di risolvere il problema senza la creazione esplicita di un modello matematico. Gli euristici possono essere classificati in due categorie: costruttivi e di raffinamento. La prima tipologia permette di ricercare una prima soluzione da cui partire, la seconda invece permette di migliorare la soluzione a propria disposizione.

3.1 Euristico costruttivo - Greedy

Uno dei metodi costruttivi più famosi è il *greedy*. Questa politica permette di costruire una soluzione al problema in passi successivi nei quali ogni volta viene effettuata la scelta che si ritiene migliore, senza mettere mai in discussione decisioni passate. Questa tecnica non garantisce l'ottimalità della soluzione.
Essendo la creazione di una soluzione molto rapida, è consigliabile randomizzare l'algoritmo in modo tale da effettuare più iterazioni e salvare la soluzione migliore da poi raffinare in seguito. Per far ciò la prima possibilità è chiamata *multi-start* e prevede di cambiare il punto di partenza della creazione dello spazio decisionale; la limitazione di questa tecnica è legata al numero di volte che può essere utilizzata, dato che il numero di differenti punti di partenza è lo stesso delle possibili decisioni iniziali. Una scelta migliore, chiamata

GRASP, consiste nel salvarsi, ad ogni passo, vari "migliori modi" di proseguire la ricerca e, invece di attuare ogni volta la scelta migliore, forzare la soluzione a scegliere ogni tanto ed in maniera random una tra le altre possibili alternative a disposizione. In questo modo siamo sicuri che in un numero elevato di iterazioni risulti praticamente impossibile ottenere due soluzioni identiche.

3.2 Euristico di raffinamento - Local search

Esistono varie tecniche di raffinamento, in particolare esiste una classe di algoritmi chiamati *Metaeuristici*. Data una soluzione da cui partire, queste tecniche tentano di migliorarla tramite un'operazione detta *mossa*, ovvero una regola che da una soluzione ne produce un'altra. Una possibile mossa consiste nel scegliere un collo, eliminarlo e sostituirlo con un altro. Così facendo si sta definendo un intorno della soluzione attuale e si sta ricercando la migliore al suo interno. La caratteristica di questo intorno è che tutte le soluzioni al suo interno sono a distanza *2-opt* da quella attuale. É necessario calcolare il costo di tutte le possibili soluzioni presenti nell'intorno e nel caso in cui ne venga trovata una migliore salvarla. Il passo successivo consiste nel centrare l'intorno nella nuova soluzione trovata e nel ripetere le operazioni appena descritte. Un problema di questa metodologia riguarda gli ottimi locali, ossia punti in cui la soluzione in considerazione, ad una data iterazione, ha un intorno dentro al quale non ne esistono di migliori. In queste situazioni l'algoritmo si ferma, poichè non riesce più a migliorare la soluzione. Un primo tentativo per risolvere tale problema consiste nell'allargare l'intorno di ricerca rendendolo *k-opt*, ma all'aumentare di k i tempi di calcolo aumentano notevolmente. É quindi necessaria una tecnica tale da forzare uno spostamento nel caso in cui la soluzione attuale sia un ottimo locale. Per far ciò, si prevede di ricercare la soluzione migliore diversa da quella attuale, così facendo se ne ottiene una peggiorativa ma che permette di muoversi evitando la situazione di stallo. Tuttavia, se si riapplicasse lo stesso metodo all'iterazione successiva, si riotterrebbe la soluzione precedente, ovvero l'ottimo locale da cui si sta cercando di uscire. Per evitare questo loop si usano i Metaeuristici come per esempio il Tabu search [3], il Simulated annealing [4] o il Variable Neighborhood Search [5].

3.3 Algoritmo sviluppato

Per la realizzazione dell'algoritmo euristico si è deciso di seguire la strada appena descritta, quindi sono state progettate e realizzate due fasi distinte. Nella prima fase viene costruita una soluzione attraverso un algoritmo greedy, nella seconda invece si effettua un raffinamento che permette alla soluzione di ricercare miglioramenti in un'intorno di quella di partenza.

3.3.1 Greedy

Inizialmente l'algoritmo ordina i colli e le casse in ordine crescente di larghezza. A questo punto viene avviata la fase costruttiva che eseguirà un numero di iterazioni uguale ad un valore dato in input all'esecuzione tramite riga di comando. In questo modo, insieme ad alcuni passi randomizzati all'interno dell'algoritmo, sarà possibile creare differenti soluzioni iniziali da cui partire con la fase di raffinamento (in accordo con il metodo *GRASP*). Una prima randomizzazione avviene all'avvio del greedy, vengono scelti casualmente dei colli che vengono subito ruotati, ancor prima di essere inseriti in qualunque cassa. Viene ora scelto il collo più largo da inserire e, di conseguenza, la cassa con larghezza minore in cui può essere inserito. Qualora esistessero più casse disponibili con la stessa larghezza l'algoritmo sceglierà casualmente una di queste. Nel caso in cui il collo selezionato è non impilabile e sono presenti altri colli da inserire, viene scelto un collo differente. Se quest'ultimo ha una dimensione tale da poter essere inserito nella cassa scelta precedentemente viene inserito, altrimenti viene cambiata cassa.
Identificata la prima cassa da dover utilizzare, ora si passa all'inserimento degli altri colli disponibili che vengono scelti in ordine decrescente di larghezza. Questa fase prevede un tentativo di inserimento davanti al collo predecessore. Se il collo in considerazione è stato già inserito in un'altra cassa, si seleziona il successivo, se invece tutti i colli sono stati collocati in una cassa l'iterazione termina. Nel caso in cui si riesca ad individuare un collo da inserire si verifica se lo spazio presente davanti al predecessore è sufficiente per contenerlo: in caso affermativo questo viene inserito, il predecessore viene aggiornato e si ricomincia con l'iterazione dal collo più largo; in caso negativo si passa al collo successivo. Dopo aver tentato di inserire davanti al predecessore tutti i colli, se ne sono presenti altri disponibili si riparte ma tentando l'inserimento sopra al predecessore. Allo stesso modo, nel caso in cui terminata questa fase siano disponibili ancora colli, si tenta un inserimento a destra del predecessore. Se il collo risulta rotabile e l'inserimento non ha avuto successo,

si verifica se una sua rotazione permetterebbe una sua collocazione ed in caso affermativo viene inserito.
Ogni volta che un inserimento ha successo si riparte con l'inserimento frontale. Le operazioni si ripetono fino a quando nessun collo può essere contenuto davanti, sopra o a destra del predecessore. In tal caso viene selezionata una nuova cassa e si ripete il tutto fino al completamento dei colli da inserire. Per velocizzare la verifica dello spazio disponibile nella cassa viene verificato se il volume ancora libero in essa non sia inferiore a quello del collo selezionato. Sono inoltre presenti dei passi di randomizzazione che scartano un collo selezionato senza tentare un suo inserimento.

Per maggior chiarezza viene ricapitolato l'algoritmo costruttivo in un elenco che definisce i passaggi principali in maniera più dettagliata:

- il numero di iterazioni viene passato da riga di comando e salvato nell'*instance* nella variabile *numitergreedy*;
- vengono scelti casualmente dei colli da ruotare inizialmente ancora prima di essere inseriti;
- si parte con le iterazioni:
    - si sceglie il collo più largo ancora da inserire;
    - viene scelta la cassa più piccola (larghezza minore) in cui può essere contenuto il collo selezionato; in caso di colli con larghezze uguali la scelta di uno di questi è casuale;
    - se il collo selezionato nella prima fase era non impilabile e sono disponibili ancora colli impilabili, questo viene cambiato e si verifica se può essere inserito nella cassa selezionata in precedenza, in caso negativo la cassa viene sostituita;
    - il collo viene inserito nella cassa;
    - si cercano altri colli da inserire in questa cassa seguendo l'ordine decrescente di larghezza e si tenta l'inserimento davanti al collo predecessore. Se tutti i colli sono stati inseriti l'iterazione termina, se invece il collo in considerazione è stato già inserito in un'altra cassa si procede come segue:
        - se sono presenti altri colli da provare ad inserire davanti al predecessore si prova con il prossimo collo;

    - ◇ se tutti i colli hanno tentato l'inserimento davanti al predecessore si riparte provandoli sopra;
    - ◇ se tutti i colli hanno tentato l'inserimento sopra al predecessore si riparte provandoli a destra;
    - ◇ se tutti i colli hanno tentato l'inserimento a destra del predecessore si riparte dal primo punto dell'iterazione con una nuova cassa da riempire.
- ◦ se non è presente abbastanza volume libero nella cassa per poter inserire il collo selezionato:
    - ◇ se sono presenti altri colli da provare ad inserire davanti al predecessore si prova con il prossimo collo;
    - ◇ se tutti i colli hanno tentato l'inserimento davanti al predecessore si riparte provandoli sopra;
    - ◇ se tutti i colli hanno tentato l'inserimento sopra al predecessore si riparte provandoli a destra;
    - ◇ se tutti i colli hanno tentato l'inserimento a destra del predecessore si riparte dal primo punto dell'iterazione con una nuova cassa da riempire.
- ◦ se si è in fase di inserimento davanti al predecessore:
    - ◇ se il collo non è impilabile e si è ancora al primo livello, questo collo viene scartato;
    - ◇ se si è al primo livello, al massimo al terzo posizionato davanti e non si ha ancora nessuno a destra, casualmente si sceglie se accettare o scartare il collo;
    - ◇ si individuano le giuste coordinate per inserire il collo e si controlla se è posizionabile in quel punto:
        - □ se ci sta, viene inserito;
        - □ se non ci sta, si prova a ruotarlo e nel caso ci stia viene inserito;
        - □ se questa fase ha avuto successo si riparte con la lista dei colli, altrimenti si prosegue con questa iterazione andando a provare l'inserimento sopra.
- ◦ se si è in fase di inserimento sopra al predecessore:
    - ◇ viene controllato se nella cassa sono presenti colli non impilabili:

        - se sono presenti si salta la fase di inserimento sopra e si passa a quella a destra;
        - se sono non sono presenti si prosegue, tentando un inserimento sopra.
      - se si è al primo livello e non è stato ancora inserito nulla a destra si sceglie casualmente se scartare il collo selezionato;
      - si individuano le giuste coordinate per inserire il collo e si controlla se è posizionabile in quel punto:
        - se ci sta, viene inserito;
        - se non ci sta, si prova a ruotarlo e nel caso ci stia viene inserito;
        - se questa fase ha avuto successo si riparte con la lista dei colli andando a tentare l'inserimento davanti al collo appena inserito, altrimenti si prosegue con questa iterazione andando a provare l'inserimento a destra.
    - se si è in fase di inserimento a destra del predecessore:
      - si sceglie casualmente se scartare questa fase;
      - se si è selezionato un collo non impilabile questo viene scartato;
      - si individuano le giuste coordinate per inserire il collo e si controlla se è posizionabile in quel punto:
        - se ci sta, viene inserito;
        - se non ci sta, si prova a ruotarlo e nel caso ci stia viene inserito;
        - se questa fase ha avuto successo si riparte con la lista dei colli andando a tentare l'inserimento davanti al collo appena inserito, altrimenti l'iterazione viene conclusa e si ripartirà con una nuova cassa da riempire con i colli rimasti.
    - viene controllato se la soluzione ottenuta contiene casse che possono essere ridotte di dimensioni conservando la disposizione interna dei colli invariata;
    - si controlla che la soluzione trovata sia ammissibile;
    - se il costo della soluzione attuale è minore della migliore ottenuta finora, quest'ultima viene sostituita.

- la soluzione migliore viene salvata e l'algoritmo termina.

3.3.2 Local search

Ottenuta una soluzione dalla fase costruttiva è necessario ora raffinarla per tentare di migliorarla. Anche in questa fase sono state inserite delle scelte randomizzate che permettano di ottenere soluzioni differenti data una di partenza. Il numero di iterazioni, ovvero il numero di tentativi di miglioramento, viene fornito in input da riga di comando e salvato nell'*instance* nella variabile *numiterlocalsearch*.
Sono state realizzate due differenti tecniche di ricerca locale: la prima consiste nel prendere due casse utilizzate e porre i colli al loro interno in una nuova cassa che li possa contenere tutti; la seconda invece effettua l'operazione inversa, ovvero prende i colli contenuti in una cassa e li smista in due differenti. Le due tecniche sono sviluppate in due diverse funzioni: *twoboxinone* e *oneboxintwo*.

Per maggior chiarezza viene ricapitolato l'algoritmo di raffinamento in un elenco che definisce i passaggi principali in maniera più dettagliata:

- LOCAL SEARCH:
    - il numero di iterazioni viene passato da riga di comando e salvato nell'*instance* nella variabile *numiterlocalsearch*;
    - si parte con le iterazioni:
        - viene scelto casualmente se tentare l'unione di due casse in una (*twoboxinone*) o se tentare di dividere una cassa in due (*oneboxintwo*);
        - nel caso in cui il tentativo non abbia avuto successo si riparte con una nuova iterazione, altrimenti si prosegue con i prossimi passaggi;
        - viene controllato se la soluzione ottenuta contiene casse che possono essere ridotte di dimensioni conservando la disposizione interna dei colli invariata;
        - si controlla che la soluzione trovata sia ammissibile;
        - se il costo della soluzione attuale è minore della migliore in assoluto, questa viene salvata come migliore assoluta e migliore delle iterazioni; se invece sono passati almeno *numtentative* (valore numerico dato in input da riga di comando) tentativi dall'ultimo miglioramento di iterazione, la soluzione viene salvata comunque come migliore delle iterazioni e si riparte con

l'iterazione successiva. In questo modo si ricomincia con una soluzione peggiore ma che potrebbe portare a migliorare quella assoluta per cercare di uscire da un ottimo locale.

- TWOBOXINONE:
  - si prova per *numtentative* tentativi ad inserire i colli contenuti nelle due casse scelte in un'altra non in uso;
  - si parte con i tentativi:
    - vengono scelte due casse da svuotare tra quelle in uso (la scelta è casuale se le casse sono almeno tre);
    - si calcola il volume totale dei colli presenti nelle due casse da svuotare tenendosi in memoria quanti colli sono presenti;
    - viene scelta una cassa non in uso che abbia le seguenti caratteristiche:
      - il volume della cassa scelta deve essere maggiore del volume totale dei colli da doverci inserire;
      - per ogni collo da dover inserire, le tre dimensioni dei colli devono essere minori o uguali alle tre dimensioni della cassa;
      - casualmente viene deciso se accettare la cassa risultata idonea o se scartarla.
    - si sceglie il primo collo da inserire nella cassa nuova, essendo il primo vengono scartati quelli non impilabili se sono presenti altri colli. Se possibile, casualmente a volte ne scarto alcuni (se è presente un solo collo la scelta è obbligata);
    - il riempimento della cassa a questo punto segue la metodologia greedy spiegata in precedenza.
- ONEBOXINTWO:
  - si prova per *numtentative* tentativi ad inserire i colli contenuti nella cassa scelta in altre due non in uso;
  - si parte con i tentativi:
    - viene scelta una cassa da svuotare tra quelle in uso (la scelta è casuale se le casse sono almeno due);

- ◇ si calcola il volume totale dei colli presenti nella cassa da svuotare tenendosi in memoria quanti colli sono presenti;
- ◇ vengono scelte due casse non in uso che abbiano le seguenti caratteristiche:
  - □ il volume totale delle due casse scelte deve essere maggiore del volume totale dei colli da doverci inserire;
  - □ per ogni collo da dover inserire, le tre dimensioni dei colli devono essere minori o uguali alle tre dimensioni della due casse;
  - □ casualmente viene deciso se accettare le due casse risultate idonee o se scartarle.
- ◇ si sceglie il primo collo da inserire nella prima cassa nuova, essendo il primo vengono scartati quelli non impilabili se sono presenti altri colli. Se possibile, casualmente a volte se ne scartano alcuni (se è presente un solo collo la scelta è obbligata);
- ◇ il riempimento della cassa a questo punto segue la metodologia greedy spiegata in precedenza.

4

Confronto algoritmi

In questo capitolo vengono presentati i test svolti ed i relativi risultati ottenuti. Inoltre viene effettuato un confronto tra le soluzioni dell'algoritmo che utilizza CPLEX, quelle dell'euristico e le soluzioni proposte dall'azienda. Per far ciò vengono analizzati i costi delle differenti disposizioni ed il tempo necessario per calcolarle. I test sono stati svolti sul cluster di calcolo dipartimentale Blade [6] e i relativi dati raccolti sono stati analizzati e graficati tramite l'utilizzo di Microsoft Excel.

4.1 Data set

Per svolgere dei test che potessero fornire un confronto diretto con i risultati ottenuti dall'azienda si è deciso di utilizzare dati reali derivanti da ordini di clienti. Sono state raccolte sette istanze per i colli e due per le casse. Essendo i primi tre ordini non molto recenti, purtroppo non si ha a disposizione la soluzione dell'azienda.
Ogni istanza di colli è identificata da un *numero di ordine* e tutti i colli che la compongono sono impilabili e rotabili ad eccezione dei pacchi telo che risultano essere molto fragili.
Le istanze dei colli vengono riportate nelle Tabelle C.1, C.2, C.3, C.4 e C.5.
Per quanto riguarda gli ordini 190527188 e 190620254 l'azienda ha fornito due ulteriori istanze che riportano le reali dimensioni dei colli e la soluzione definita dal magazziniere. Le due istanze vengono riportate in Tabella C.6 e C.7.

Per le istanze riguardanti le casse, come detto in precedenza, si è deciso di assegnare un costo ad ogni cassa uguale al volume della stessa. Questa scelta è legata alla mancanza di dati reali che forniscano questo dato. Nonostante il costo attribuito non sia veritiero, l'algoritmo permette di confrontare i risultati ottenuti con quelli dell'azienda attribuendo alla loro soluzione un costo pari alla somma dei volumi delle casse usate dal magazziniere. Si evidenzia tuttavia che nella realtà i costi delle casse sono effettivamente proporzionali al loro volume, dunque, per questa ragione, la scelta effettuata non porterebbe alcun conflitto qualora i costi delle casse dovessero variare mantenendo la proporzionalità.
Vengono riportate le due istanze composte da 52 e 306 casse nelle Tabelle C.8 e C.9.

4.2 Test svolti

Al fine di analizzare il comportamento degli algoritmi sviluppati sono stati svolti diversi test per ogni istanza. Per avere un confronto diretto tra i risultati ottenuti utilizzando CPLEX e quelli ricavati dall'algoritmo euristico sono state effettuate diverse esecuzioni a parità di tempo. Per verificare quanto tempo è necessario all'algoritmo per ottenere buoni risultati per l'azienda sono stati effettuati i test con *timelimit* di 1, 2, 5, 10, 30 e 60 minuti.

4.2.1 Risultati con l'utilizzo di CPLEX

I risultati ottenuti con l'utilizzo di CPLEX non risultano soddisfacenti in quanto le soluzioni ricavate attraverso la creazione di un modello matematico risultano essere molto costose. L'insuccesso di questo algoritmo è legato alla presenza dei BigM che rendono il modello molto debole.
Come mostrato in Figura 4.1 e 4.2, per quasi tutte le istanze la differenza tra la soluzione ottenuta nei primi minuti e il *lower bound* calcolato da CPLEX è piuttosto elevata.

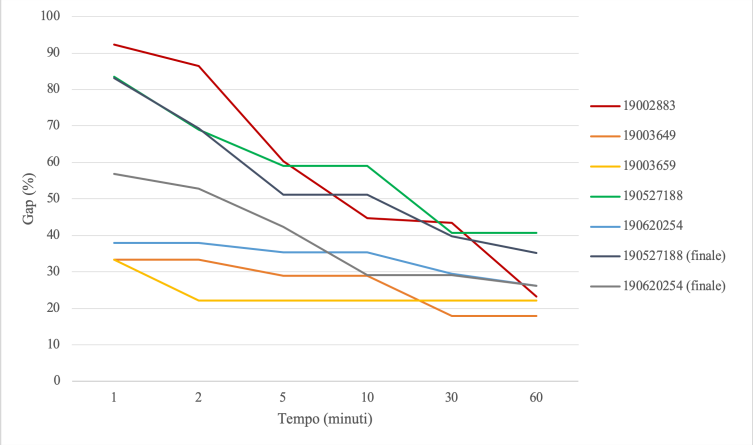

Figura 4.1: Risultati ottenuti con CPLEX con l'istanza *Casse 52*

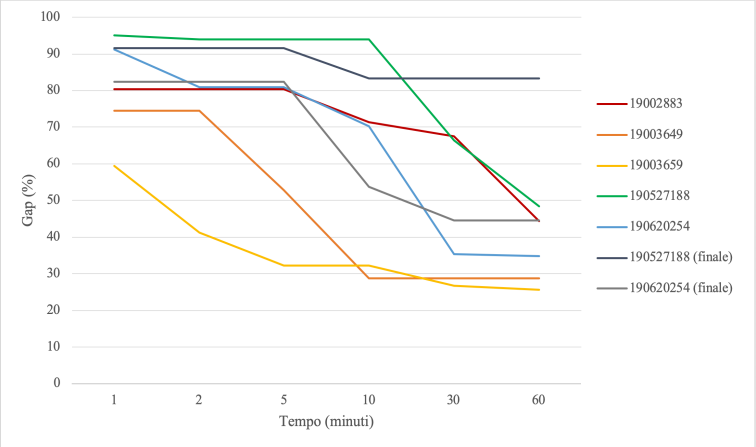

Figura 4.2: Risultati ottenuti con CPLEX con l'istanza *Casse 306*

Una possibile soluzione a tale problema potrebbe essere la creazione di un nuovo modello suddiviso in più parti, ognuna con una funzione obiettivo, che possa portare ad una soluzione migliore grazie ad una distribuzione del lavoro. Tuttavia questa possibilità non viene analizzata ulteriormente in quanto l'azienda preferisce un software che non richiede una licenza d'utilizzo, per tale ragione è stato approfondito lo studio di un algoritmo euristico.

4.2.2 Risultati con l'algoritmo euristico

A differenza dei risultati ottenuti tramite la creazione di un modello matematico, le soluzioni ricavate con l'utilizzo dell'algoritmo euristico realizzato sono molto buone. Come è possibile vedere in Figura 4.3 e 4.4 la distanza dal *lower bound* risulta essere più ridotta.

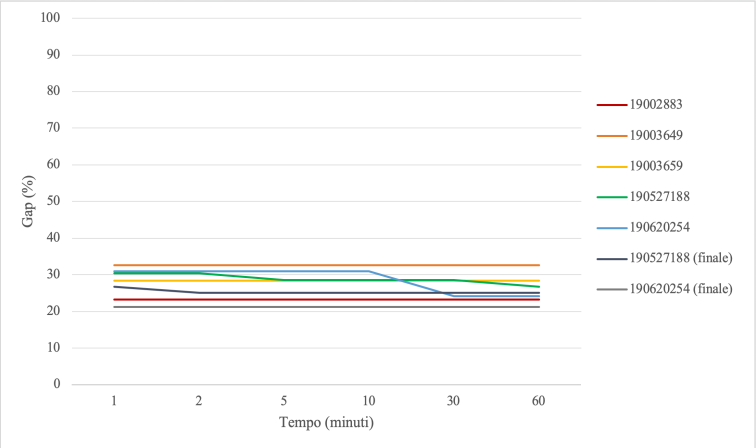

Figura 4.3: Risultati ottenuti con l'euristico con l'istanza *Casse 52*

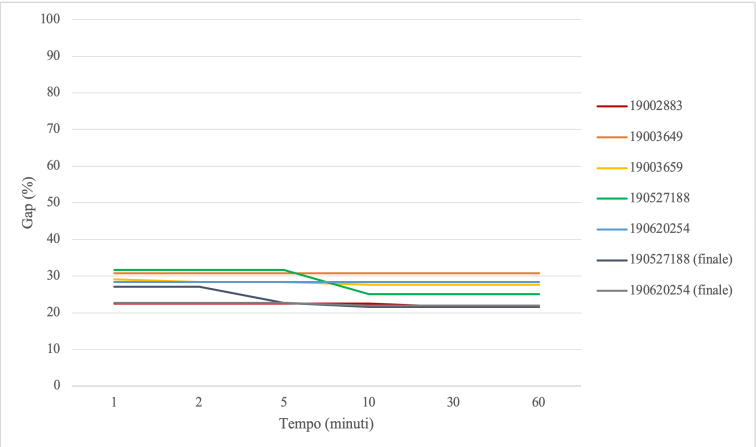

Figura 4.4: Risultati ottenuti con l'euristico con l'istanza *Casse 306*

É necessario sottolineare che per il calcolo del *lower bound* si è presa in considerazione la somma dei volumi dei colli dell'ordine, in quanto, secondo la nostra ipotesi iniziale, il costo delle casse della soluzione è il volume delle stesse.

Un'ulteriore evidenza emersa dai grafici è data dalla velocità con la quale si ottiene la soluzione, statistica molto rilevante vista la necessità dell'azienda di raggiungere buoni risultati nel tempo più breve possibile.

4.3 Confronto tra i due algoritmi creati

Per quanto descritto nei Capitoli 4.2.1 e 4.2.2 i risultati ottenuti con l'utilizzo dell'algoritmo euristico risultano essere notevolmente migliori rispetto a quelli ricavati dalla creazione del modello matematico. Il confronto è facilmente visibile nelle successive Figure.

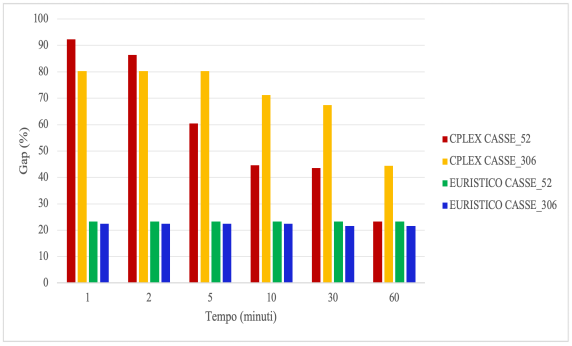

Figura 4.5: Risultati ottenuti per l'ordine 19002883

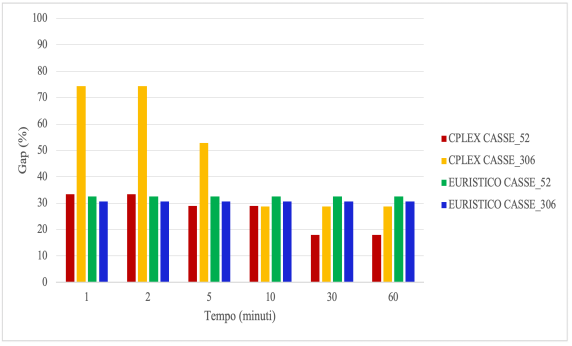

Figura 4.6: Risultati ottenuti per l'ordine 19003649

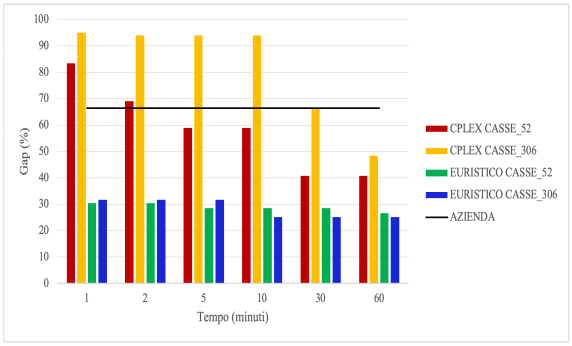

Figura 4.7: Risultati ottenuti per l'ordine 190527188

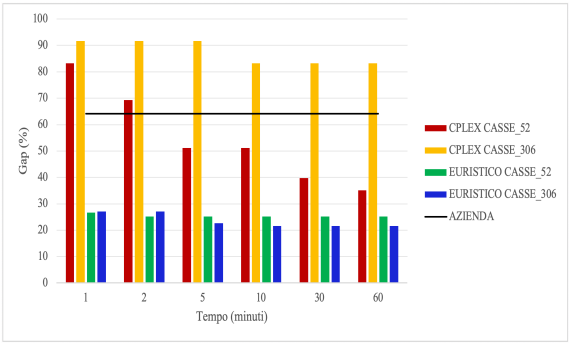

Figura 4.8: Risultati ottenuti per l'ordine 190527188 finale

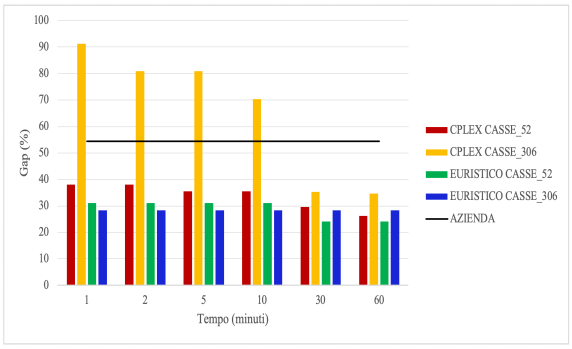

Figura 4.9: Risultati ottenuti per l'ordine 190620254

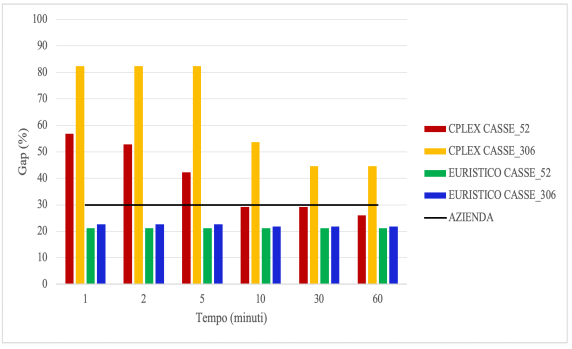

Figura 4.10: Risultati ottenuti per l'ordine 190620254 finale

Dai grafici riportati è possibile notare che la differenza è piuttosto marcata in istanze composte da molti colli o casse. Questo aspetto è facilmente spiegabile in quanto all'aumentare dei componenti dell'ordine aumenta la presenza dei BigM che di conseguenza rendono il modello di CPLEX sempre più debole. L'unica istanza in cui i due algoritmi risultano avere un comportamento simile e dove addirittura CPLEX riesce a superare le prestazioni dell'euristico nei primi minuti di esecuzione è l'ordine 19003659 (composto da solo 16 colli) con 52 casse a disposizione.
Come è possibile notare in Figura 4.11, la differenza è minima e per tale ragione l'euristico resta sicuramente preferibile per l'azienda committente vista la prevalenza di ordini con un numero più elevato di colli e casse.
É necessario sottolineare che il confronto tra i due algoritmi non è comunque equo in quanto si sta confrontando un metodo esatto con un euristico.

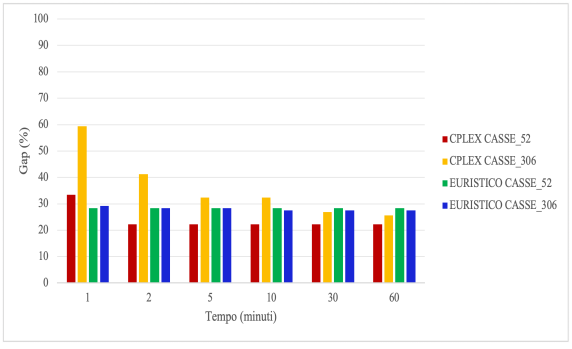

Figura 4.11: Risultati ottenuti per l'ordine 19003659

4.4 Confronto con i risultati dell'azienda

Avendo riscontrato che l'algoritmo euristico è preferibile all'utilizzo di CPLEX per la ricerca di una soluzione in breve termine che possa soddisfare l'azienda, in questa sezione verranno confrontati i migliori risultati ottenuti con l'algoritmo realizzato e quelli ottenuti dal magazziniere aziendale.
Come detto in precedenza purtroppo non si hanno a disposizione le soluzioni dell'azienda per tutti gli ordini. Concentrandosi quindi sulle ultime 4 istanze è possibile notare dalle Figure 4.7, 4.8, 4.9 e 4.10, riportate nel Capitolo 4.3, come il costo dell'euristico sia sempre inferiore rispetto a quella dell'azienda.

Ordine	**Costo euristico (m^3)**	**Costo azienda (m^3)**	**Risparmio (%)**	**Tempo (minuti)**
190527188	5,022	11,190144	55,12	2
190527188 finale	5,805	10,472853	44,57	1
190620254	5,778	8,974164	35,61	1
190620254 finale	5,778	5,843750	1,13	1

Tabella 4.1: Confronto soluzioni euristico con azienda

Come riportato dalla Tabella 4.1 il risparmio aziendale risulta essere maggiore negli "ordini-preventivo", ossia l'iniziale stima relativa ai colli necessari per spedire la merce effettuata dall'azienda al momento in cui riceve l'ordine dal cliente. Tale stima tuttavia spesso non corrisponde all'ordine finale. Il risparmio ovviamente si riduce negli ordini finali, dove il magazziniere ha la possibilità di avere una situazione reale dei colli a disposizione e quindi più tempo per realizzare una soluzione più economica per l'azienda mettendo in atto la sua esperienza. In ogni caso la soluzione ottenuta dall'algoritmo euristico in circa 1 o 2 minuti risulta essere migliore di quella dell'azienda.

5
Conclusioni

In questo elaborato sono stati presentati ed analizzati due approcci differenti per la realizzazione di un software di supporto decisionale per un'azienda. L'obiettivo principale riguardava la possibilità di ottenere in breve tempo una stima delle casse necessarie per poter contenere i colli da spedire al cliente. Per far ciò si è proposto inizialmente un algoritmo che, grazie al supporto di CPLEX, risolve un modello matematico che descrive il problema con i relativi vincoli da rispettare nella soluzione proposta. Questo approccio è risultato efficace solo nel caso in cui il numero di colli da spedire e di casse a disposizione sia piuttosto limitato. La presenza di ordini più grandi, accompagnata da una vasta lista di casse a disposizione rendono questa metodologia non la migliore possibile. Considerando inoltre che l'utilizzo di CPLEX richiede il pagamento di una licenza commerciale si è deciso di seguire una via alternativa, ossia lo sviluppo di un algoritmo euristico.
Questo secondo approccio è risultato più idoneo alle richieste dell'azienda. Dai risultati dei test svolti si è riscontrato che l'algoritmo fornisce risultati molto interessanti in pochissimi minuti di esecuzione, fornendo all'azienda una soluzione più economica già dalla fase di preventivo al cliente.
Bisogna inoltre considerare che, oltre al risparmio economico, si otterranno altri benefici dall'utilizzo del software, come per esempio il risparmio di ore lavorative degli addetti alla preparazione del trasporto della merce da spedire al cliente, la riduzione di mezzi di trasporto necessari per la spedizione con la rispettiva diminuzione dell'inquinamento ambientale creato dagli stessi.

Un software decisionale porterebbe quindi diversi vantaggi all'azienda che diminuirebbe anche il rischio di disallineamento tra il costo predetto in fase di preventivo e quello reale di spedizione, con relativa maggior soddisfazione del cliente.

A

CPLEX

IBM ILOG CPLEX Optimization Studio, o più comunemente CPLEX, è un programma di ottimizzazione. Prende il nome dal metodo del simplesso (simplex method) implementato in linguaggio C, anche se al giorno d'oggi offre interfacce verso altri linguaggi come C++, C#, Python, Java e Matlab. Fu sviluppato da Robert E. Bixby e fu commercializzato a partire dal 1988 dalla CPLEX Optimization Inc., che venne successivamente acquistata, in un primo momento da ILOG nel 1997, e successivamente da IBM nel 2009. IBM ILOG CPLEX Optimizer risolve problemi di programmazione lineare, lineare intera e problemi di programmazione quadratica convessa. Esistono due modi differenti per interagire con il software. Il primo avviene tramite la scrittura di un modello matematico in un file di testo in formato LP che verrà letto e risolto da CPLEX in interattivo. Il secondo, invece, prevede l'utilizzo delle API del risolutore per l'implementazione di un interfaccia attraverso codice sorgente. La risoluzione utilizza le varianti primale o duale del metodo del simplesso. CPLEX viene utilizzato in vari ambiti rendendo problemi aziendali veri e propri problemi matematici modellizzati e ottimizzati. Questo meccanismo permette alle più grandi aziende al mondo di risparmiare milioni di euro.

B

Gnuplot

Gnuplot è un programma che permette la rappresentazione di funzioni matematiche in due o tre dimensioni. É disponibile per diversi sistemi operativi e possiede un'interfaccia a riga di comando. Il suo utilizzo permette di esportare grafici in differenti formati. Una volta scaricato e installato il programma tramite il sito `http://www.gnuplot.info` è possibile da subito utilizzare il software aprendo il terminale e digitando il comando gnuplot. Nel caso specifico di interesse si è deciso, per comodità, di creare un file in formato *.gp* che, fornito in input a gnuplot, permette al software di seguire in ordine le istruzioni scritte. Così facendo l'unica istruzione necessaria per la rappresentazione di uno o più grafici è *gnuplot nome_file.gp*. Questo metodo permette di creare veri e propri programmi script con gnuplot. Si è scelto di plottare i colli con dei parallelepipedi colorati e la cassa con una linea che permettesse di rappresentarne i bordi interni.
Il risultato ottenuto grazie alle istruzioni elencate e grazie a [9] è il seguente:

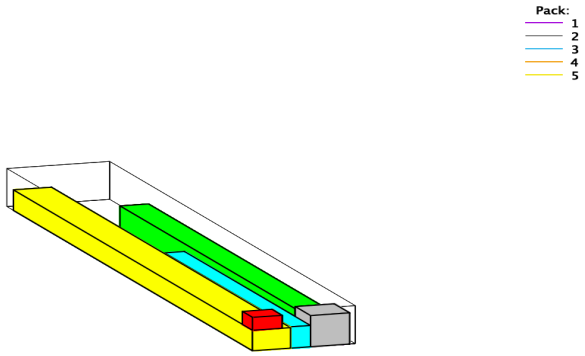

Figura B.1: Rappresentazione gnuplot colli all'interno di una cassa

Vengono riporte le istruzioni scritte nel *plot.gp* per ottenere il precedente risultato.

```
reset
set terminal png size 1280, 960
set cbrange [1:5]
set palette defined (\ 1 '#C0C0C0',\ 2 '#FF0000',\ 3 '#FFFF00',\ 4 '#00FF00',\ 5 '#00FFFF')
set style line  2 lc rgb 'black' lt 1 lw 0.5
set key reverse outside title "Pack:"
set border 0
unset tics
unset colorbox
set view 58,27,2
set view equal xyz
load 'cube.fct'
load 'box.fct'
set pm3d depthorder hidden3d
set pm3d explicit
unset hidden3d
set lmargin 2
set rmargin 0
set bmargin 0
set tmargin 0
add_box(x,y,z,r,g,u) = sprintf('box(%f,%f,%f,%f,%f,%f) w l ls 2 notitle,',x,y,z,r,g,u)
add_cube(x,y,z,r,g,u,c) = sprintf('cube(%f,%f,%f,%f,%f,%f,%i) with pm3d notitle,cube(%f,%f,%f,%f,%f,%f
,%i) w l ls %i title "%i",',x,y,z,r,g,u,c,x,y,z,r,g,u,c,c,c)
CMD = ''
set output './Plot/box_38.png'
stats './Plot/box_38.txt' u 1:(CMD = CMD.add_cube($1,$2,$3,$4,$5,$6,$7)) nooutput
stats './Plot/boxout_38.txt' u 1:(CMD = CMD.add_box($1,$2,$3,$4,$5,$6)) nooutput
CMD = 'splot '.CMD.'1/0 w l ls 2 notitle'
eval(CMD)
```

Il file *box_38.txt* riporta le dimensioni dei colli posizionati nella cassa 38 le cui dimensioni interne sono presenti nel file *boxout_38.txt*. I due file vengono creati dalle funzioni *cube.fct* e *box.fct* e sono formattati in modo da avere nelle prime tre colonne le coordinate X, Y, Z di uno spigolo del parallelepipedo e nelle successive tre colonne le sue tre dimensioni.

```
610 0 0 410 470 410 1
0 0 270 310 310 170 2
0 0 0 400 7300 270 3
610 470 0 300 5350 380 4
400 0 0 210 3850 280 5
```

```
# Usage: cube(x,y,z,r,g,u,c)
cube(x,y,z,r,g,u,c) = sprintf('<echo "\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i\n\
%f %f %f %i"',\
0+x,0+y,0+z,c,\
0+x,0+y,u+z,c,\
0+x,g+y,u+z,c,\
0+x,g+y,0+z,c,\
0+x,0+y,0+z,c,\
r+x,0+y,0+z,c,\
r+x,0+y,u+z,c,\
r+x,g+y,u+z,c,\
r+x,g+y,0+z,c,\
r+x,0+y,0+z,c,\
0+x,0+y,0+z,c,\
r+x,0+y,0+z,c,\
r+x,g+y,0+z,c,\
0+x,g+y,0+z,c,\
0+x,0+y,0+z,c,\
0+x,0+y,u+z,c,\
r+x,0+y,u+z,c,\
r+x,g+y,u+z,c,\
0+x,g+y,u+z,c,\
0+x,0+y,u+z,c\
)
```

Per ulteriori informazioni sull'utilizzo di gnuplot o per la creazione dei file utilizzati si rimanda al codice del programma sviluppato.

C

Tabelle Data set

Id	Tipologia	Larghezza (mm)	Profondità (mm)	Altezza (mm)
1	COLLO PORTA BANDE	3073	378	175
2	COLLO PORTA BANDE	3162	392	185
3	COLLO PORTA BANDE	4869	234	320
4	COLLO PORTA BANDE	4829	260	320
5	GRONDAIETTA PORTA LED, GRONDAIA	4610	220	270
6	GRONDAIETTA PORTA LED, GRONDAIA	2870	220	275
7	GRONDAIETTA PORTA LED, GRONDAIA	4690	225	270
8	GRONDAIETTA PORTA LED, GRONDAIA	2773	223	270
9	GRONDAIETTA PORTA LED, GRONDAIA	4720	220	275
10	COLLO GAMBE	2773	220	275
11	COLLO GAMBE	2819	233	270
12	COLLO GAMBE	2843	225	270
13	COLLO GAMBE	2884	220	270
14	COLLO SCATOLA ACCESSORI	310	370	310
15	COLLO SCATOLA ACCESSORI	395	455	395
16	COLLO PACCO TELO	6300	800	200

Tabella C.1: Ordine 19003649

Id	Tipologia	Larghezza (mm)	Profondità (mm)	Altezza (mm)
1	COLLO PORTA BANDE	2746	260	340
2	COLLO PORTA BANDE	2760	226	325
3	COLLO PORTA BANDE	6299	230	345
4	COLLO PORTA BANDE	6688	230	330
5	GRONDAIETTA PORTA LED, GRONDAIA	2553	221	270
6	GRONDAIETTA PORTA LED, GRONDAIA	6725	220	275
7	GRONDAIETTA PORTA LED, GRONDAIA	6539	226	280
8	GRONDAIETTA PORTA LED, GRONDAIA	6551	235	270
9	COLLO PACCO LAME	3400	280	200
10	COLLO PACCO LAME	3400	280	200
11	COLLO PACCO LAME	3400	280	200
12	COLLO PACCO LAME	3400	280	200
13	COLLO PACCO LAME	3400	280	200
14	COLLO PACCO LAME	3400	280	200
15	COLLO PACCO LAME	3400	280	200
16	COLLO PACCO LAME	3400	280	200
17	COLLO PACCO LAME	3400	280	200
18	COLLO PACCO LAME	3400	280	200
19	COLLO PACCO LAME	3400	280	200
20	COLLO GAMBE	2583	225	270
21	COLLO GAMBE	2553	216	275
22	COLLO GAMBE	2576	230	270
23	COLLO GAMBE	2535	216	275
24	COLLO GAMBE	2545	234	270
25	COLLO GAMBE	2545	221	270
26	GRONDAIETTA PORTA LED, GRONDAIA	2515	225	275
27	GRONDAIETTA PORTA LED, GRONDAIA	2503	245	275
28	COLLO PACCO TELO	5400	700	200

Tabella C.2: Ordine 19002883

Id	Tipologia	Larghezza (mm)	Profondità (mm)	Altezza (mm)
1	COLLO PORTA BANDE	3103	380	180
2	COLLO PORTA BANDE	3169	387	190
3	COLLO PORTA BANDE	3016	225	330
4	COLLO PORTA BANDE	3001	265	325
5	GRONDAIETTA PORTA LED, GRONDAIA	2792	220	270
6	GRONDAIETTA PORTA LED, GRONDAIA	2760	215	270
7	GRONDAIETTA PORTA LED, GRONDAIA	2804	235	275
8	GRONDAIETTA PORTA LED, GRONDAIA	2836	220	270
9	COLLO GRONDAIA-PR INTERMEDIO	2903	224	275
10	COLLO GAMBE	2834	223	275
11	COLLO GAMBE	2811	221	275
12	COLLO GAMBE	2881	225	270
13	COLLO GAMBE	3020	218	275
14	COLLO SCATOLA ACCESSORI	310	370	310
15	COLLO SCATOLA ACCESSORI	395	455	395
16	COLLO PACCO TELO	5500	900	200

Tabella C.3: Ordine 19003659

Id	Tipologia	Larghezza (mm)	Profondità (mm)	Altezza (mm)
1	COLLO AUTOMATISMI	200	200	200
2	COLLO AUTOMATISMI	200	200	200
3	COLLO CARTER LATERALE	2750	280	200
4	COLLO CARTER LATERALE	2750	280	200
5	COLLO CARTER LATERALE	4150	280	200
6	COLLO CARTER LATERALE	4150	280	200
7	COLLO GAMBE	2500	280	200
8	COLLO GAMBE	2500	280	200
9	COLLO GAMBE	2500	280	200
10	COLLO GAMBE	2500	280	200
11	COLLO LAME	2700	280	200
12	COLLO LAME	2700	280	200
13	COLLO LAME	2700	280	200
14	COLLO LAME	2700	280	200
15	COLLO LAME	2700	280	200
16	COLLO LAME	2700	280	200
17	COLLO LAME	2700	280	200
18	GRONDAIA, GRONDAIETTA PORTA LED	4200	280	200
19	GRONDAIA, GRONDAIETTA PORTA LED	4200	280	200
20	GRONDAIA, GRONDAIETTA PORTA LED	2600	280	200
21	GRONDAIA, GRONDAIETTA PORTA LED	2600	280	200
22	GRONDAIETTA PORTA LED, GRONDAIA	2800	280	200
23	GRONDAIETTA PORTA LED, GRONDAIA	4200	280	200
24	COLLO STRUTTURA	2700	120	230
25	COLLO SCATOLA ACCESSORI	160	240	160
26	COLLO STRUTTURA	4060	120	230
27	COLLO SCATOLA ACCESSORI	160	240	160

Tabella C.4: Ordine 190527188

Id	Tipologia	Larghezza (mm)	Profondità (mm)	Altezza (mm)
1	COLLO AUTOMATISMI	200	200	200
2	COLLO LAME	3750	280	200
3	COLLO LAME	3750	280	200
4	COLLO LAME	3750	280	200
5	COLLO LAME	3750	280	200
6	COLLO LAME	3750	280	200
7	COLLO LAME	3750	280	200
8	COLLO LAME	3750	280	200
9	COLLO LAME	3750	280	200
10	COLLO LAME	3750	280	200
11	COLLO LAME	3750	280	200
12	COLLO PORTA BANDE	5900	280	200
13	COLLO PORTA BANDE	5900	280	200
14	COLLO SCATOLA ACCESSORI TWIST	460	410	410
15	GRONDAIA, GRONDAIETTA PORTA LED	3800	280	200
16	GRONDAIA, GRONDAIETTA PORTA LED	3800	280	200
17	GRONDAIETTA PORTA LED, GRONDAIA	6000	280	200
18	GRONDAIETTA PORTA LED, GRONDAIA	6000	280	200
19	SCATOLA ACCESSORI ANGOLARI ANIMA	460	410	410
20	SCATOLA ACCESSORI ANGOLARI ANIMA	460	410	410

Tabella C.5: Ordine 190620254

Id	Tipologia	Larghezza (mm)	Profondità (mm)	Altezza (mm)
1	COLLO LAME	4000	280	200
2	COLLO LAME	4000	280	200
3	COLLO LAME	4000	280	200
4	COLLO LAME	4000	280	200
5	COLLO LAME	4000	280	200
6	COLLO LAME	4000	280	200
7	COLLO LAME	4000	280	200
8	COLLO LAME	4000	280	200
9	COLLO LAME	4000	280	200
10	COLLO LAME	4000	280	200
11	GRONDAIETTA PORTA LED, GRONDAIA	3850	235	275
12	GRONDAIETTA PORTA LED, GRONDAIA	3810	225	275
13	COLLO PORTA BANDE	6180	256	275
14	COLLO PORTA BANDE	6180	247	275
15	COLLO SCATOLA ACCESSORI	310	370	310
16	COLLO SCATOLA ACCESSORI	310	370	310
17	COLLO SCATOLA ACCESSORI	750	20	270
18	GRONDAIETTA PORTA LED, GRONDAIA	6000	232	280
19	GRONDAIETTA PORTA LED, GRONDAIA	6220	245	280

Tabella C.6: Ordine finale 190620254

Id	Tipologia	Larghezza (mm)	Profondità (mm)	Altezza (mm)
1	COLLO LAME	2700	280	200
2	COLLO LAME	2700	280	200
3	GRONDAIETTA PORTA LED, GRONDAIA	2800	228	275
4	GRONDAIETTA PORTA LED, GRONDAIA	4280	216	270
5	COLLO GAMBE	2601	218	270
6	COLLO GAMBE	2550	220	275
7	COLLO GAMBE	2584	235	275
8	COLLO GAMBE	2543	235	275
9	COLLO LAME	2700	280	200
10	COLLO LAME	2700	280	200
11	COLLO LAME	2700	280	200
12	COLLO LAME	2700	280	200
13	COLLO LAME	2700	280	200
14	COLLO SCATOLA ACCESSORI	310	370	310
15	COLLO SCATOLA ACCESSORI	395	455	395
16	GRONDAIETTA PORTA LED, GRONDAIA	2908	225	270
17	GRONDAIETTA PORTA LED, GRONDAIA	2841	224	275
18	GRONDAIETTA PORTA LED, GRONDAIA	4292	223	275
19	GRONDAIETTA PORTA LED, GRONDAIA	4330	225	280
20	COLLO PORTA BANDE	4606	239	330
21	COLLO PORTA BANDE	4578	237	325
22	COLLO PORTA BANDE	3045	231	320
23	COLLO PORTA BANDE	3030	260	345
24	COLLO SCATOLA ACCESSORI TS	160	160	240
25	COLLO STRUTTURA	2972	226	120
26	COLLO MOTORE/AUTOMATISMI	110	110	220
27	COLLO SCATOLA ACCESSORI TS	160	160	240
28	COLLO STRUTTURA	4478	232	120

Tabella C.7: Ordine finale 190527188

Id	Larghezza (mm)	Altezza (mm)			
		250	**500**	**750**	**1000**
1-4	3000	0,810	1,620	2,430	3,240
5-8	3500	0,945	1,890	2,835	3,780
9-12	4000	1,080	2,160	3,240	4,320
13-16	4500	1,215	2,430	3,645	4,860
17-20	5000	1,350	2,700	4,050	5,400
21-24	5500	1,485	2,970	4,455	5,940
25-28	6000	1,620	3,240	4,860	6,480
29-32	6500	1,755	3,510	5,265	7,020
33-36	7000	1,890	3,780	5,670	7,560
37-40	7500	2,025	4,050	6,075	8,100
41-44	8000	2,160	4,320	6,480	8,640
45-48	8500	2,295	4,590	6,885	9,180
49-52	9000	2,430	4,860	7,290	9,720

Tabella C.8: Casse 52 - Costo cassa (m^3) in funzione delle dimensioni (Profondità 1080 mm)

Id	Larghezza (mm)	Altezza (mm)					
		250	**500**	**750**	**1000**	**1250**	**1500**
1-6	3000	0,810	1,620	2,430	3,240	4,050	4,860
7-12	3100	0,837	1,674	2,511	3,348	4,185	5,022
13-18	3200	0,864	1,728	2,592	3,456	4,320	5,184
19-24	3300	0,891	1,782	2,673	3,564	4,455	5,346
25-30	3400	0,918	1,836	2,754	3,672	4,590	5,508
31-36	3500	0,945	1,890	2,835	3,780	4,725	5,670
37-42	3600	0,972	1,944	2,916	3,888	4,860	5,832
43-48	3700	0,999	1,998	2,997	3,996	4,995	5,994
49-54	3800	1,026	2,052	3,078	4,104	5,130	6,156
55-60	3900	1,053	2,106	3,159	4,212	5,265	6,318
61-66	4000	1,080	2,160	3,240	4,320	5,400	6,480
67-72	4100	1,107	2,214	3,321	4,428	5,535	6,642
73-78	4200	1,134	2,268	3,402	4,536	5,670	6,804
79-84	4300	1,161	2,322	3,483	4,644	5,805	6,966
85-90	4400	1,188	2,376	3,564	4,752	5,940	7,128
91-96	4500	1,215	2,430	3,645	4,860	6,075	7,290

97-102	4600	1,242	2,484	3,726	4,968	6,210	7,452
103-108	4700	1,269	2,538	3,807	5,076	6,345	7,614
109-114	4800	1,296	2,592	3,888	5,184	6,480	7,776
115-120	4900	1,323	2,646	3,969	5,292	6,615	7,938
121-126	5000	1,350	2,700	4,050	5,400	6,750	8,100
127-132	5100	1,377	2,754	4,131	5,508	6,885	8,262
133-138	5200	1,404	2,808	4,212	5,616	7,020	8,424
139-144	5300	1,431	2,862	4,293	5,724	7,155	8,586
145-150	5400	1,458	2,916	4,374	5,832	7,290	8,748
151-156	5500	1,485	2,970	4,455	5,940	7,425	8,910
157-162	5600	1,512	3,024	4,536	6,048	7,560	9,072
163-168	5700	1,539	3,078	4,617	6,156	7,695	9,234
169-174	5800	1,566	3,132	4,698	6,264	7,830	9,396
175-180	5900	1,593	3,186	4,779	6,372	7,965	9,558
181-186	6000	1,620	3,240	4,860	6,480	8,100	9,720
187-192	6100	1,647	3,294	4,941	6,588	8,235	9,882
193-198	6200	1,674	3,348	5,022	6,696	8,370	10,044
199-204	6300	1,701	3,402	5,103	6,804	8,505	10,206
205-210	6400	1,728	3,456	5,184	6,912	8,640	10,368
211-216	6500	1,755	3,510	5,265	7,020	8,775	10,530
217-222	6600	1,782	3,564	5,346	7,128	8,910	10,692
223-228	6700	1,809	3,618	5,427	7,236	9,045	10,854
229-234	6800	1,836	3,672	5,508	7,344	9,180	11,016
235-240	6900	1,863	3,726	5,589	7,452	9,315	11,178
241-246	7000	1,890	3,780	5,670	7,560	9,450	11,340
247-252	7100	1,917	3,834	5,751	7,668	9,585	11,502
253-258	7200	1,944	3,888	5,832	7,776	9,720	11,664
259-264	7300	1,971	3,942	5,913	7,884	9,855	11,826
265-270	7400	1,998	3,996	5,994	7,992	9,990	11,988
271-276	7500	2,025	4,050	6,075	8,100	10,125	12,150
277-282	7600	2,052	4,104	6,156	8,208	10,260	12,312
283-288	7700	2,079	4,158	6,237	8,316	10,395	12,474
289-294	7800	2,106	4,212	6,318	8,424	10,530	12,636
295-300	7900	2,133	4,266	6,399	8,532	10,665	12,798
301-306	8000	2,160	4,320	6,480	8,640	10,800	12,960

Tabella C.9: Casse 306 - Costo cassa (m^3) in funzione delle dimensioni (Profondità 1080 mm)

Riferimenti

[1] CPLEX Optimizer. https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer

[2] Fischetti M., Lodi A., *"Local branching"*, Springer-Verlag Berlin Heidelberg, 2003.

[3] Glover F., Laguna M., *"Tabu search"*, Kluwer Academic Publishers Dordrecht, 1997.

[4] Kirkpatrick S., Gelatt C.D., Vecchi M.P., *"Optimization by simulated annealing"*, Science 220 pag:671-680, 1983.

[5] Hansen P., Mladenovi N., Uroevi D., *"Variable neighborhood search and local branching"*, Computers & Operations Research vol:33 pag:3034-3045, 2006.

[6] Cluster di calcolo dipartimentale Blade. https://www.dei.unipd.it/bladecluster

[7] Gnuplot. http://gnuplot.sourceforge.net

[8] Documentazione Gnuplot. http://www.gnuplot.info/docs_5.2/Gnuplot_5.2.pdf

[9] Guida gnuplot. http://www.gnuplotting.org/plotting-cubes/