UNIVERSITÀ DI PISA

DIPARTIMENTO DI MATEMATICA
CORSO DI LAUREA TRIENNALE IN MATEMATICA

TESI DI LAUREA TRIENNALE

Packing 2d: Studio e implementazione di un nuovo algoritmo

CANDIDATO:
Federico Giuseppe Pisanu

RELATORE:
Prof. Antonio Frangioni

ANNO ACCADEMICO 2015/2016

Indice

Introduzione

Il packing, o nesting, è un problema dalle svariate applicazioni tanto nel caso bidimensionale quanto in quello tridimensionale. L'obiettivo è quello di disporre nel modo migliore degli oggetti di forma irregolare in un contenitore. È facile intuire come vari settori dell'industria siano interessati alla risoluzione del problema, come ad esempio quella del taglio di componenti su lastre di metallo o quella tessile per quanto riguarda il caso bidimensionale, oppure ancora l'industria dei trasporti o dello stoccaggio per il caso in tre dimensioni. Questo interesse in diversi ambiti ha alimentato la ricerca nella soluzione di svariate varianti del problema. Infatti un'industria di taglio su lastre potrebbe essere interessata al nesting di figure bidimensionali irregolari su un contenitore di forma rettangolare, mentre invece un industria tessile potrebbe voler ottimizzare la disposizione di pezzi su una pelle di forma irregolare, oppure ancora si potrebbe essere interessati al packing di poligoni convessi o non convessi che possono essere piazzati ruotandoli in modo libero, discreto oppure anche invertendone il verso. I problemi relativi al packing bidimensionale possono variare anche per funzione obiettivo, ad esempio nel caso di contenitore rettangolare, si può supporre di avere fissata una sola delle due dimensioni del contenitore e minimizzare l'utilizzo dell'altra, oppure fissando entrambe le dimensioni e dati una quantità di pezzi che superano la capacità del foglio, scegliere quali pezzi piazzare e in quali posizioni, in modo da minimizzare l'area del contenitore inutilizzata. Il problema del packing rientra in una categoria più vasta di problemi, i "Cutting & Packing problems". Tra questi vi sono anche problemi apparentemente eterogenei come quello che cerca di trovare tra tanti poligoni quello con area minore che può contenere un certo insieme di poligoni più piccoli.

In questa trattazione ci occuperemo di studiare e proporre un algoritmo per risolvere le istanze del problema di packing bidimensionale con le seguenti caratteristiche: il contenitore è un foglio di lavoro rettangolare dalle dimensioni date, i pezzi da disporre sono delle figure che hanno come bordo una concatenazione chiusa di segmenti e archi di circonferenza, tutte le rotazioni sono consentite e la funzione obbiettivo cerca di minimizzare l'a-

rea inutilizzata del foglio. L'obiettivo è quello di fornire un algoritmo misto esatto-euristico che sia ragionevole nelle prestazioni. Arriveremo a formulare diverse varianti di formulazioni di programmazione lineare intera-mista per il problema, poi proporremo algoritmi che fanno uso del modello MILP e di qualche euristica.

Capitolo 1

Rappresentazioni geometriche del problema

Il primo ostacolo che si trova nell'affrontare il problema di nesting è sicuramente quello di formalizzare il concetto di non-intersezione tra le figure, un concetto molto intuitivo per l'uomo ma non altrettanto per un computatore in quanto dipende strettamente dal modo in cui viene rappresentato il pezzo. Le rappresentazioni più utilizzate sono sicuramente quella come griglia e quella come poligono, ma vi sono anche idee più originali come quelle basate sulla copertura con cerchi. Per ognuna di queste rappresentazioni, in letteratura, si trovano diversi metodi per controllare la sovrapposizione delle figure come ad esempio l'utilizzo del No-Fit-Polygon o della Phi-Function per la rappresentazione poligonale o con cerchi o ad esempio operazioni matriciali per quella a griglia. Di seguito proporremo più nel dettaglio alcune di queste idee.

1.1 Rappresentazione a griglia

L'idea dietro la rappresentazione come griglia è quella di dividere il foglio in zone quadrate di uguale dimensione ed utilizzare una matrice per memorizzare quali posizioni del foglio sono occupate e quali no. I metodi con cui viene utilizzata la matrice sono diversi:

Il più semplice è quello proposto da J.F. Oliveira e J.S. Ferreira in [JO93] che utilizza il valore 0 per le zone libere e a questa cifra viene aggiunto 1 ogni volta che un pezzo viene posizionato in quella posizione. In questo modo le zone che hanno un valore maggiore di 1 sono zone in cui vi è una sovrapposizione.

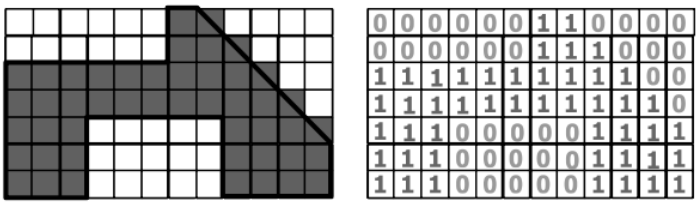


Figura 1.1: Rappresentazione di Oliveira e Ferreira

Un altro metodo è quello proposto da Sagenreich e Braga in [SS86] che permette di controllare anche il contatto fra i pezzi e non solo la sovrapposizione. Il valore zero nella matrice rappresenta sempre una zona vuota e quando viene posizionato un nuovo pezzo viene aggiunto un 3 nelle zone interne al pezzo ed un 1 nelle zone del bordo. In questo modo il controllo della sovrapposizione di riduce a controllare che la matrice non presenti numeri maggiori od uguali a 4 e le zone che presentano un 2 sono quelle di contatto tra i pezzi.

Un altro approccio che utilizza la matrice è quello proposto da Babu e Babu in [AB01] di cui si può vedere una raffigurazione nell'immagine 1.2. La codifica della matrice assegna un numero maggiore di 0 alle posizioni libere del foglio, in particolare,ogni volta che viene posizionato un pezzo, per ogni riga interessata viene posto un 1 nella posizione libera più a destra e incrementando di uno ogni volta ci si sposta da destra verso sinistra. Questa codifica è particolarmente efficace nel caso in cui si voglia adottare una euristica bottom-left, infatti il numero presente su una zona del foglio indica quante caselle bisogna spostarsi verso destra per trovare una possibile posizione ammissibile (senza intersezioni con i pezzi già disposti). D'altra parte questa codifica richiede però un maggiore costo nell'aggiornamento della matrice.

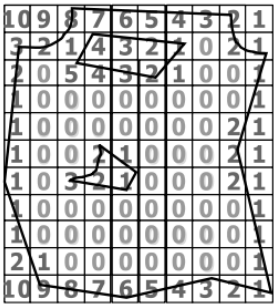


Figura 1.2: Rappresentazione di Babu e Babu

Tra i vantaggi dei metodi di rappresentazione a griglia che abbiamo descritto vi è sicuramente la velocità nel controllare l'ammissibilità di una data

soluzione, ma anche la versatilità dei pezzi che possono essere usati. Infatti non viene fatta differenza per pezzi convessi o meno, oppure pezzi più o meno complessi. Inoltre risulta anche facile calcolare le distanze che i pezzi devono percorrere per raggiungere una posizione ammissibile.

Tra gli svantaggi invece bisogna sicuramente citare il fatto che la griglia non approssima un modo accurato pezzi che non hanno dei bordi con soli angoli retti e che all'aumentare della precisione che si chiede aumenta sostanzialmente anche la memoria richiesta per mantenere la matrice. Infatti aumentare la precisione richiesta vorrebbe dire infittire la griglia e quindi aumentare la risoluzione della matrice e dunque la sua dimensione. Inoltre una grossa dimensione della matrice implica anche che i controlli di intersezione e gli spostamenti siano più costosi. La rappresentazione come griglia non è particolarmente compatibile con la possibilità di ruotare i pezzi. Una soluzione potrebbe essere quella di ruotare il pezzo originale e ricalcolarne la codifica in matrice, il che sarebbe come avere un nuovo pezzo, oppure si potrebbe ruotare direttamente la matrice ma questo è possibile solo con rotazioni multiple di 90 gradi infatti quest'ultimo metodo spesso non viene usato poichè troppo restrittivo nelle rotazioni.

1.2 Rappresentazione con Phi-Functions

Le Phi-Functions sono essenzialmente delle espressioni matematiche che derivano dalla distanza di due oggetti. L'obbiettivo è quello di avere una funzione delle posizioni dei pezzi e delle rotazioni che sia maggiore di zero se i due oggetti sono lontani, zero se vi è un contatto e negativa se si intersecano. Per fare questo ogni pezzo viene decomposto in pezzi più semplici come quelli proposti in figura per cui si riescono ad avere in modo analitico le funzioni di distanza a partire dalle posizione relative dei pezzi.

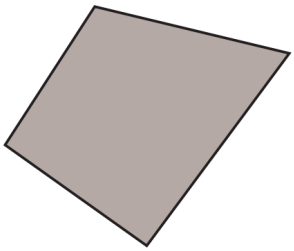


Figura 1.3: Poligono convesso

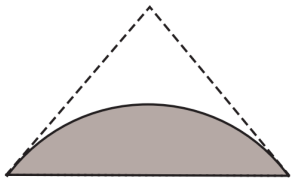


Figura 1.4: Segmento circolare

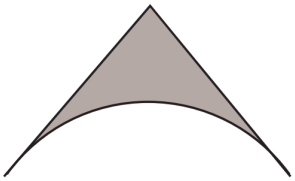


Figura 1.5: Cappello

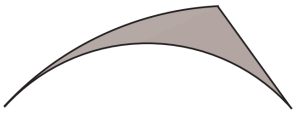


Figura 1.6: Corno

Le funzioni relative ai pezzi della composizione sono utilizzate per creare la funzione relativa all'intera combinazione di forme semplici. Questo metodo non è usato spesso perchè richiede l'utilizzo di modelli di programmazione non lineare che sono ancora complessi e difficili da risolvere. Inoltre l'impatto computazionale aumenta con l'aumentare della complessità dei pezzi e questo è il principale fattore che limita la diffusione di questo metodo anche se sono diverse le ricerche al momento che cercano di perfezionare l'idea.

1.3 Copertura con cerchi

Un altro modo per rappresentare i poligoni e controllarne le intersezioni è quello che utilizza la copertura per cerchi. L'obbiettivo è quello di trovare un insieme di cerchi che coprano la figura e rappresentare il poligono attraverso l'insieme dei centri e dei raggi di questi. Una volta trovata una copertura per ogni pezzo, controllare l'intersezione si riduce a controllare le intersezioni tra i cerchi ossia controllare posizioni dei centri e i raggi relativi.

Dunque il problema sta tutto nel trovare una buona copertura. In merito sono tanti gli studi proposti in quanto risulta molto interessante anche il caso tridimensionale del problema per il calcolo di collisioni tra oggetti. Nel nostro caso l'obbiettivo sarebbe quello di minimizzare il numero di cerchi utilizzati in modo da velocizzare il controllo di intersezione e uno degli algoritmi euristici che cerca di trovare una soluzione al problema è quello proposto da Moore in [Moo02] che cerca di dividere il pezzo da coprire con una griglia di quadrati dando a questa una struttura ad albero. Ogni quadrato non completamente contenuto nella figura viene diviso in 4 quadrati che rappresentano i figli di cui alcuni saranno contenuti nella figura, altri no. Per trovare una copertura per cerchi basta ora considerare dei cerchi che coprono i nodi foglia dell'albero.

Tra i vantaggi del metodo di rappresentazione per cerchi vi è sicuramente la velocità nel controllo di intersezione ma anche la versatilità nel considerare le rotazioni dei pezzi. È infatti sufficiente ruotare ogni singolo centro per avere la copertura del pezzo ruotato. Gli svantaggi risiedono chiaramente nel calcolo di una buona copertura che dipende fortemente dalle caratteristiche del pezzo e dalla sua complessità.

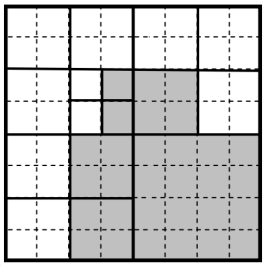


Figura 1.7: Griglia di quadrati

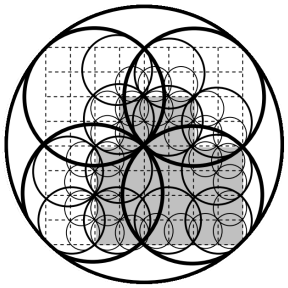


Figura 1.8: Griglia di cerchi

1.4 No-fit Polygon

Un altro metodo studiato per rappresentare i poligoni è quello di utilizzare il No-Fit Polygon. Intuitivamente il No-Fit Polygon tra i poligoni A e B è il poligono racchiuso dalla traccia dell'origine di B quando questo viene fatto scorrere adiacente sul perimetro del primo senza ruotare e senza che vi siano intersezioni. Più formalmente definiamo NFP_{AB} come l'insieme dei punti

$$\{p \quad | \quad A \cap (p + B) \neq \emptyset\}$$

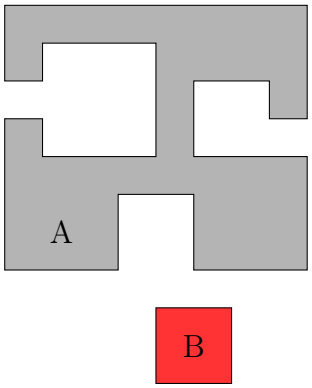


Figura 1.9: esempi di poligoni

Si può dimostrare che NFP_{AB} è uguale a NFP_{BA} ruotato di 180 gradi e che anche se i poligoni A e B sono senza buchi, NFP_{AB} può presentarne. La rappresentazione in figura 1.10 mostra che in alcuni casi molto particolari in cui ci sono degli incastri perfetti questo metodo non risulta essere preciso. Infatti il punto p_2 e il segmento p_3 appartengono al NFP_{AB} ma sono in realtà posizioni possibili. Di solito si sceglie di accettare questo fatto e non ammettere quelle posizioni come ammissibili. Questo in effetti non permette di ricostruire delle istanze di tipo puzzle con i pezzi che si incastrano perfettamente ma in realtà nelle applicazioni reali questo non è un problema in quanto un incastro perfetto è decisamente raro se i pezzi non sono creati in modo apposito.

I metodi per il calcolo del NFP verranno spiegati successivamente per ora basta sapere che il calcolo non è immediato e non può essere fatto in tempo reale per cui non risulta essere adatto all' utilizzo con rotazioni libere

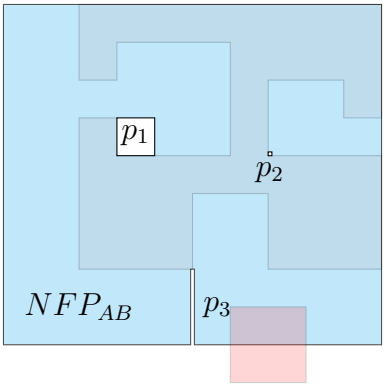


Figura 1.10: No-fit polygon

in quanto un cambio di rotazione di uno dei due pezzi obbliga ad un ricalcolo completo dell' NFP. Un vantaggio è sicuramente che esistono metodi per utilizzare questa rappresentazione per la creazione di un modello di programmazione lineare che cerca di risolvere il problema del packing. Questo è stato il motivo per cui è stato scelto tale metodo per la nostra implementazione. I successivi capitoli saranno mirati a mostrare la creazione di modelli di programmazione lineare che risolvano istanze di packing in due dimensioni con particolari proprietà.

Capitolo 2

Ricondurre packing 2D a problema MILP

Il primo passo che faremo sarà quello di ricondurci a studiare il problema di disporre poligoni invece che poligoni con archi, infatti a meno di fare un approssimazione molto fine e di usare un numero elevato di segmenti è possibile ricoprire in modo preciso con un poligono ogni figura del tipo che ci viene fornito. Questo ci permette di lavorare con figure più semplici e rende più facile il controllo dell'intersezione fra i pezzi. Un 'altra approssimazione che dobbiamo fare riguarda le rotazioni libere, per scrivere il problema con un modello di programmazione lineare intera mista occorre che il dominio delle variabili continue del modello, come quelle che regolano le rotazioni dei pezzi, sia un convesso. Questo non accade, infatti ci possono essere delle configurazioni e dei pezzi tali che una figura può essere disposta in una certa posizione solo con due rotazioni particolari. Questo ci spinge a restringere l'utilizzo a solo un numero finito di rotazioni per ogni oggetto. Perciò indicizziamo i poligoni in input con la variabile i e definiamo come R_i l'insieme delle rotazioni (modalità) del poligono i. Per ottenere la possibilità di scegliere tra le rotazioni, si utilizzeranno più copie dello stesso poligono ognuna con una rotazione diversa e si impone nel modello che solo una delle rotazioni possa essere piazzata.

2.1 Dati e prime variabili del modello

Attraverso le semplificazioni che abbiamo fatto l'input per la creazione del modello MILP è dato da:

- l'insieme dei poligoni da disporre
- per ogni poligono, l'insieme finito delle rotazioni che può assumere
- le dimensioni del foglio di lavoro (larghezza e altezza)

Nel resto della trattazione verrà usata la seguente notazione per riferirsi ai dati del problema:

X_F dimensione orizzontale del foglio di lavoro (dato in input)

Y_F dimensione verticale del foglio di lavoro (dato in input)

l_{ir} dimensione orizzontale del rettangolo di contenimento del poligono i in modalità r

h_{ir} dimensione verticale del rettangolo di contenimento del poligono i in modalità r

a_i area del poligono i

P insieme dei poligoni di input, contiene i poligoni a meno di rotazioni

R_i insieme di rotazioni del poligono i

Di seguito le prime variabili del modello con la loro semantica.

x_i $\forall i \in P$ coordinata orizzontale della posizione del poligono i (vertice bottom-left del rettangolo di contenimento)

y_i $\forall i \in P$ coordinata verticale della posizione del poligono i (vertice bottom-left del rettangolo di contenimento)

z_{ir} $\forall i \in P, \forall r \in R_i$ variabile binaria che vale 1 se il poligono i viene posizionato in modalità r, 0 altrimenti

Come si può vedere dalle variabili introdotte si è fatta la scelta di utilizzare una sola coppia di variabili di posizione per ogni pezzo indipendentemente dalla rotazione in cui si è scelto di porlo.

2.2 Funzione obiettivo e primi vincoli del modello

La funzione obiettivo del modello si può scrivere nel seguente modo

$$\min \quad \frac{X_F \cdot Y_F - \sum_{i \in P} \sum_{r=1}^{R_i} a_i z_{ir}}{X_F \cdot Y_F}$$

o equivalentemente

$$\max \sum_{i \in P} \sum_{r=1}^{R_i} a_i z_{ir}$$

Possiamo già da subito dare alcune delle disuguaglianze che verranno utilizzate come vincoli nella formulazione MILP del problema

$$x_i \leq (X_F - \sum_{r \in R_i} l_{ir} z_{ir}) \qquad i \in P \tag{2.1}$$

$$y_i \leq (Y_F - \sum_{r \in R_i} h_{ir} z_{ir}) \qquad i \in P \tag{2.2}$$

$$\sum_{r \in R_i} z_{ir} \leq 1 \qquad i \in P \tag{2.3}$$

$$z_{ir} \in \{0, 1\} \tag{2.4}$$

I vincoli (1) e (2) sono i vincoli di contenimento nel foglio di lavoro che si suppone posizionato nel piano in modo che il vertice in basso a sinistra coincida con l'origine. Il vincolo (3) impone che al più una delle modalità del poligono i venga disposta.

2.3 No-fit polygon e vincoli di non sovrapposizione

2.3.1 Come calcolare il No-fit polygon

Siano A e B due poligoni di cui si vuole calcolare il no-fit polygon, senza perdita di generalità possiamo fissare il loro punto di riferimento nell'origine e siano p_A e p_B rispettivamente le posizioni dei loro punti di riferimento dopo una qualche traslazione nel piano. Denotiamo con p_A+A il poligono ottenuto da A traslandolo in modo da portare il suo punto di riferimento in p_A (ossia traslandolo di p_A). Dimostriamo che

Teorema 2.1. $(p_A+A)\cap(p_B+B) \neq \emptyset \iff p_B-p_A \in \{a-b \mid a \in A \quad b \in B\}$

Dimostrazione.

$(p_A+A)\cap(p_B+B) \neq \emptyset \iff \exists x \in (p_A+A)\cap(p_B+B) \iff x = p_A+a = p_B+b$

con $a \in A$ e $b \in B$. Ma allora

$$p_B - p_A = a - b$$

□

Corollario 2.2.

$$NFP_{AB} = \{a - b \mid a \in A \quad b \in B\}$$

Se definiamo come $E \oplus F = \{e + f \mid e \in E \quad f \in F\}$ la somma di Minkowski tra gli insiemi E e F e se consideriamo $-B = \{-b \mid b \in B\}$ allora abbiamo che

$$NFP_{AB} = A \oplus -B$$

I vertici di $-B$ posso essere calcolati facilmente cambiando di segno quelli di B e invertendone l'ordine. Calcolare il no-fit polygon atraverso la somma di Minkowski non è il metodo più veloce conosciuto ma risulta comunque, secondo le statistiche trovate, molto efficiente.

2.3.2 Decomposizione di NFP^c

Vogliamo scrivere il complementare di NFP_{AB} come unione disgiunta di politopi convessi. Per farlo possiamo procedere per fasi:

- consideriamo l'inviluppo convesso di NFP_{AB} e cerchiamo una decomposizione in convessi della differenza tra NFP_{AB} e il suo inviluppo

- cerchiamo una decomposizione in convessi del complementare dell'inviluppo

fase 1 In questa fase si vuole decomporre con convessi il complementare di NFP_{AB} nel suo inviluppo convesso. Questo insieme comprende sia eventuali buchi di NFP_{AB} che eventuali concavità. Per farlo possiamo calcolare i poligoni che sono la differenza tra NFP_{AB} ed il suo inviluppo utilizzare un algoritmo che calcola una decomposizione in convessi.

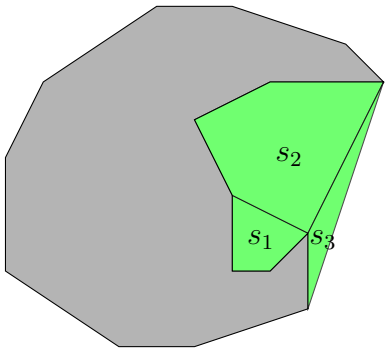


Figura 2.1: Esempio di decomposizione nella fase 1.2

Ad ogni componente convessa della decomposizione verrà associata una variabile binaria nel modello che sarà attiva se B sta in una certa componente convessa rispetto ad A. Il vantaggio dell'utilizzo di una decomposizione minimale in convessi è quello di avere, nel modello che stiamo creando, il numero minore possibile di variabili intere relative alla scelta di posizionamento relativo tra ogni coppia di pezzi.

fase 2 In ultimo vogliamo decomporre il complementare dell' inviluppo convesso, che chiameremo NFP^*_{AB} . Anche in questa fase possiamo agire in modi diversi come ad esempio:

1. decomporre il complementare del convesso utilizzando le bisettrici degli angoli interni del poligono
2. utilizzare per la decomposizione solo fasce orizzontali e procedere come segue per trovare le componenti: una volta individuati i vertici che hanno coordinata verticale massima M e quelli che invece ce l'hanno minima N (entrambi questi insiemi saranno connessi nella sequenza ordinata dei vertici), per ogni coppia di vertici consecutivi v_i, v_{i+1} non entrambi contenuti in M o in N si considera

il convesso esterno a NFP^*_{AB} delimitato da $\overline{v_i, v_{i+1}}$ e dalle rette orizzontali passanti per v_i e v_{i+1}. Rimangono solo da considerare i due semipiani sopra e sotto NFP^*_{AB}

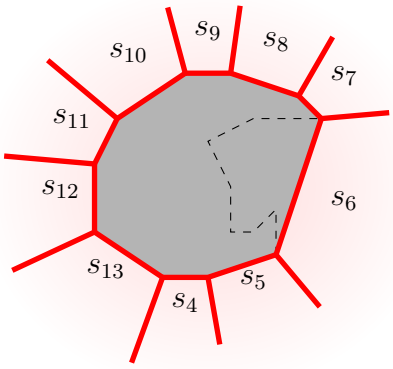


Figura 2.2: Decomposizione con bisettrici

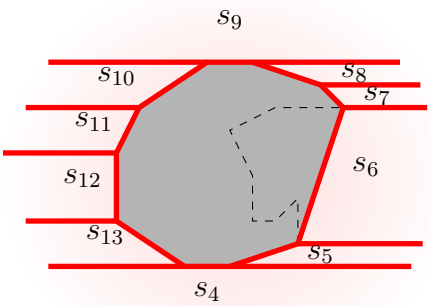


Figura 2.3: Decomposizione con le fasce orizzontali

Figura 2.4: Esempi di decomposizione di NFP^*

Le differenze fra questi due approcci stanno nel fatto che da una parte la decomposizione risulta maggiormente equipartita, il che potrebbe essere vantaggiosa visto che (come si capirá successivamente) questo si rifletterá su una maggiore equidisposizione delle soluzioni nelle partizioni definite naturalmente dalle variabili binarie nel modello, d'altra parte la seconda opzione sembra, a seconda della posizione presa dai pezzi coinvolti, più espressiva, spiega meglio le posizioni relative rispetto alla coordinata verticale. Questo aiuterebbe nel caso in cui si volesse attuare una strategia di tagli allo spazio delle soluzioni del rilassamento continuo in quanto se ad esempio so che il pezzo A è di fianco al pezzo B e il pezzo C è di fianco al pezzo B allora potrei sapere qualcosa sulla posizione relativa tra A e C.

2.3.3 Vincoli di non sovrapposizione

Per imporre la non sovrapposizione dei poligoni A e B vogliamo costringere il punto di riferimento di quest'ultimo a stare in uno dei convessi della decomposizione di NFP^c_{AB} per cui consideriamo $s^1_{AB} \dots s^{m_{AB}}_{AB}$ i convessi della decomposizione. Ogni s^k_{AB} è caratterizzato da f^k_{AB} vincoli lineari facilmente calcolabili durante la decomposizione che denotiamo con $\alpha^{kf}_{AB}x+\beta^{kf}_{AB}y \leq w^{kf}_{AB}$. Nell'espressione precedente A e B identificano i poligoni considerati, k la componente convessa della decomposizione di NFP^c_{AB} e f il vincolo di tale componente considerato. Introduciamo per ogni s^i_{AB} una variabile binaria:

$b_{AB,k}$ variabile binaria che vale 1 se B si trova nel convesso corrispondente a s^k_{AB}, 0 altrimenti

Utilizzando i vincoli standard per l'unione di convessi si ottengono le seguenti disuguaglianze:

$$\alpha^{kf}_{irjr'}(x_j - x_i) + \beta^{kf}_{irjr'}(y_j - y_i) \leq w^{kf}_{irjr'} + M(1 - b_{irjr'k})$$
$$i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \quad k = 1 \dots m_{irjr'} \quad f = 1 \dots f^k_{irjr'} \tag{2.5}$$

$$\sum_{h=1}^{m_{irjr'}} b_{irjr'h} \geq z_{ir} + z_{jr'} - 1 \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{2.6}$$

Un coefficiente M valido per le disequazione puó essere ad esempio

$$\sqrt{X_F^2 + Y_F^2}$$

ma la scelta di questa costante potrebbe essere significativa poichè decide quanto il poliedro delle soluzioni ammissibili del rilassamento continuo del modello sia "aderente" all'inviluppo delle soluzioni intere che è una caratteristica che velocizza la risoluzione del problema da parte del solutore. Per essere precisi basta valutare il vincolo nei 4 angoli del foglio per ottenere la migliore scelta possibile. Il vincolo (6) impone che se entrambi i pezzi i e j sono disposti, allora deve essere scelta una delle s^k_{ij} dove posizionare j.

Capitolo 3

Modelli MILP per il packing 2D

In questo capitolo utilizzeremo lo studio fatto nei capitoli precedenti per formulare diversi modelli mirati alla risoluzione del problema. I modelli che verranno proposti saranno principalmente di 3 tipi:

Modelli 1 e 2 sono due modelli che non richiedono ulteriori studi oltre quelli visti finora, risolvono il problema con le restrizioni descritte sulle rotazioni dei pezzi e lavorano con il foglio di lavoro rettangolare. Entrambi i modelli sono stati studiati da R. Alvarez-Valdes n, A. Martinez e J.M. Tamarit in [RAVn13] che però cercavano di risolvere una variante diversa del problema in cui il foglio di lavoro aveva una dimensione libera e la funzione obbiettivo minimizzava l'utilizzo di questa coordinata. Inoltre non venivano considerate più di una rotazione per ogni pezzo. Noi abbiamo riadattato il modello per l'utilizzo con la nostra funzione obbiettivo ed un numero arbitrario di rotazioni per pezzo.

Modello 3 è una generalizzazione del Modello 1 per poter lavorare su foglio di lavoro poligonale. Anche questa è una generalizzazione inedita dei modelli precedenti.

Modello 4 è un modello che descrive un problema diverso da quello del packing 2D ma in qualche modo correlato. Cerca di trovare, dati un numero finito di pezzi, una loro disposizione nello spazio che sia più "compatta" possibile. Verrà precisato quali sono le possibili interpretazioni dell'aggettivo "compatta" che permetteranno la formulazione di diverse varianti dello stesso modello. Queste verranno utilizzate nel capitolo successivo per implementare algoritmi diversi dalla semplice risoluzione di un modello del problema per cercare una buona soluzione.

3.1 Modelli 1 e 2

Con i dati raccolti possiamo finalmente formulare un primo modello completo per il problema

$$\max \quad \sum_{i\in P}\sum_{r\in R_i} a_i z_{ir}$$

$$x_i \le (X_F - \sum_{r\in R_i} l_{ir} z_{ir}) \qquad i \in P \tag{3.1}$$

$$y_i \le (Y_F - \sum_{r\in R_i} h_{ir} z_{ir}) \qquad i \in P \tag{3.2}$$

$$\alpha^{kf}_{irjr'}\,(x_j - x_i) + \beta^{kf}_{irjr'}\,(y_j - y_i) \le w^{kf}_{irjr'} + M(1 - b_{irjr'k})$$
$$i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \quad k = 1 \dots m_{irjr'} \quad f = 1 \dots f^k_{irjr'} \tag{3.3}$$

$$\sum_{h=1}^{m_{irjr'}} b_{irjr'h} \ge z_{ir} + z_{jr'} - 1 \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{3.4}$$

$$\sum_{r\in R_i} z_{ir} \le 1 \qquad i \in P \tag{3.5}$$

$$x_i, y_i \ge 0 \qquad i \in P \tag{3.6}$$

$$z_{ir} \in \{0,1\} \qquad i \in P \tag{3.7}$$

$$b_{irjr'h} \in \{0,1\} \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \quad h = 1 \dots m_{irjr'} \tag{3.8}$$

Con qualche accorgimento si può costruire ora un modello leggermente diverso. L'idea è quella di rendere il dominio del rilassamento continuo più aderente all'inviluppo convesso delle soluzioni intere. Per fare questo calcolo per ogni coppia di vincolo e componente convessa il massimo del problema di ottimizzazione

$$\delta^{kfh}_{ij} = \max_{(x,y)\in s^h_{ij}} \alpha^{kf}_{ij} x + \beta^{kf}_{ij} y$$

dimostriamo che

Teorema 3.1. $\sum_{h=1}^{m_{ij}} \delta^{kfh}_{ij} b_{ijh} \le M(1 - b_{ijk})$

Dimostrazione. Sappiamo che M deve essere tale che

$$M \ge \max_{(x,y)\in s^h_{ij}} \alpha^{kf}_{ij} x + \beta^{kf}_{ij} y \qquad \forall h = 1 \dots m_{ij}$$

poichè se $b_{ijk} = 0$ il vincolo deve essere rispettato per ogni altra posizione nel foglio di lavoro, dunque

$$M \geq \delta_{ij}^{kfh} \qquad \forall h = 1 \dots m_{ij}$$

allora

$$\sum_{h=1}^{m_{ij}} \delta_{ij}^{kfh} b_{ijh} \leq (\max_h \delta_{ij}^{kfh}) \sum_{h=1}^{m_{ij}} b_{ijh} \leq M$$

poichè $\sum_{h=1}^{m_{ij}} b_{ijh} \leq 1$

□

Si tratterebbe quindi di svolgere $m_{ij} \cdot \sum_{k=1}^{m_{ij}} f_{ij}^k$ problemi di ottimizzazione in due dimensioni molto simili tra loro per ogni coppia (i, j) di poligoni con $j \notin R_i$. In tal caso è possibile fare la formulazione:

$$\max \quad \sum_{i \in P} \sum_{r \in R_i} a_i z_{ir}$$

$$x_i \leq (X_F - \sum_{r \in R_i} l_{ir} z_{ir}) \qquad i \in P \tag{3.9}$$

$$y_i \leq (Y_F - \sum_{r \in R_i} h_{ir} z_{ir}) \qquad i \in P \tag{3.10}$$

$$\alpha^{kf}_{irjr'} (x_j - x_i) + \beta^{kf}_{irjr'} (y_j - y_i) \leq \sum_{h=1}^{m_{irjr'}} \delta^{kfh}_{irjr'} b_{irjr'h} + M(1 - c_{irjr'})$$
$$i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \quad k = 1 \dots m_{irjr'} \quad f = 1 \dots f^k_{irjr'} \tag{3.11}$$

$$\sum_{h=1}^{m_{irjr'}} b_{irjr'h} \geq z_{ir} + z_{jr'} - 1 \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{3.12}$$

$$\sum_{h=1}^{m_{irjr'}} b_{irjr'h} \leq 1 \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{3.13}$$

$$c_{irjr'} \geq z_{ir} + z_{jr'} - 1 \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{3.14}$$

$$c_{irjr'} \leq z_{ir} \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{3.15}$$

$$c_{irjr'} \leq z_{jr'} \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{3.16}$$

$$\sum_{r \in R_i} z_{ir} \leq 1 \qquad i \in P \tag{3.17}$$

$$x_i, y_i \geq 0 \qquad i \in P \tag{3.18}$$

$$z_{ir} \in \{0,1\} \qquad i \in P \tag{3.19}$$

$$b_{irjr'h} \in \{0,1\} \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \quad h = 1 \dots m_{irjr'} \tag{3.20}$$

$$c_{irjr'} \in \{0,1\} \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{3.21}$$

In questo modello è necessario utilizzare delle nuove variabili per controllare quando due pezzi sono disposti contemporaneamente.

$c_{irjr'}$ variabile binaria che vale 1 se il poligono i è inserito in modalità r e il poligono j in modalità r', 0 altrimenti.

3.2 Modello 3 (foglio di lavoro poligonale)

Un adattamento del modello precedente al caso in cui si debba lavorare con un foglio di lavoro non rettangolare e poligonale risulta molto veloce. Infatti dato P il poligono su cui si deve lavorare, T_P il suo rettangolo di contenimento e i poligoni da disporre all'interno, è sufficiente immaginare di dover disporre su T_P tutti i poligoni dati, insieme alla figura $T_P \setminus P$, con il vincolo che quest'ultima debba essere disposta e stia nell'unica posizione in cui può stare. Per fare questo ci occorre calcolare, per ogni pezzo da disporre, il no-fit polygon con $T_P \setminus P$, intersecarlo con T_P e effettuare una decomposizione di questo in convessi (corrispettivo della fase 1). Non occorre effettuare la fase 2 poiché gli oggetti non possono essere piazzati all'esterno di T_P Per cui a patto di saper fare la somma di Minkowski di poligoni con buco possiamo formulare con le seguenti notazioni il modello:

X_{T_P} dimensione orizzontale di T_P

Y_{T_P} dimensione verticale di T_P

NFP_P^{ir} no-fit polygon tra $T_P \setminus P$ e il poligono i in modalità r

m_P^{ir} numero di componenti convesse di $(NFP_P^{ir})^c$

f_P^{ik} numero di vincoli della componente convessa k di $(NFP_P^{ir})^c$

$\bar{a}_{ir}^{kf} x + \bar{b}_{ir}^{kf} y \leq \bar{w}_{ir}^{kf}$ equazione del vincolo numero f della componente convessa k di $(NFP_P^{ir})^c$

b_P^{irk} variabile binaria che vale 1 se il poligono i è posizionato nella componente convessa k di $(NFP_P^{ir})^c$ in modalità r

$$\max \quad \sum_{i \in P} \sum_{r \in R_i} a_i z_{ir}$$

$$x_i \leq (X_{T_P} - \sum_{r \in R_i} l_{ir} z_{ir}) \qquad i \in P \tag{3.22}$$

$$y_i \leq (Y_{T_P} - \sum_{r \in R_i} h_{ir} z_{ir}) \qquad i \in P \tag{3.23}$$

(3.24)

$$\bar{a}_{ir}^{kf}(x_i) + \bar{b}_{ir}^{kf}(y_i) \leq \bar{w}_{ir}^{kf} + M(1 - b_P^{irk}) \qquad i \in P \quad r \in R_i \quad k = 1 \dots m_P^{ir} \quad f = 1 \dots f_P^{ik} \tag{3.25}$$

$$\alpha_{irjr'}^{kf}(x_j - x_i) + \beta_{irjr'}^{kf}(y_j - y_i) \leq w_{irjr'}^{kf} + M(1 - b_{irjr'k}) \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i, j \in P \quad k = 1 \dots m_{irjr'} \quad f = 1 \dots f_{irjr'}^{k} \tag{3.26}$$

$$\sum_{h=1}^{m_{irjr'}} b_{irjr'h} \geq z_{ir} + z_{jr'} - 1 \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i, j \in P \tag{3.27}$$

$$\sum_{h=1}^{m_P^i} b_P^{irk} = z_{ir} \qquad i \in P \quad r \in R_i \tag{3.28}$$

$$\sum_{r \in R_i} z_{ir} \leq 1 \qquad i \in P \tag{3.29}$$

$$x_i, y_i \geq 0 \qquad i \in P \tag{3.30}$$

$$z_{ir} \in \{0, 1\} \qquad i \in P \tag{3.31}$$

$$b_{irjr'h} \in \{0, 1\} \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i, j \in P \quad h = 1 \dots m_{irjr'} \tag{3.32}$$

$$b_P^{irk} \in \{0, 1\} \qquad i \in P \quad r \in R_i \qquad k = 1 \dots m_P^{ir} \tag{3.33}$$

3.3 Modello 4 (Problema della disposizione compatta)

Si vuole studiare, alla luce delle considerazioni fatte sui problemi precedenti, il problema che , dati un insieme di poligoni, cerca di trovare una loro giustapposizione che minimizzi l'area del rettangolo di contenimento che li contiene tutti. Una tale funzione obiettivo risulta non essere lineare ma quadratica e non convessa come funzione delle posizioni dei pezzi. Questo implica che una modellizzazione diretta non è possibile nè in programmazione lineare nè in programmazione quadratica in quanto entrambe richiedono che la funzione obbiettivo sia convessa in termini delle variabili.

3.3.1 Funzioni obiettivo

In questa sezione verranno proposte diverse varianti lineari di funzioni obiettivo per il problema della disposizione compatta. Per semplicità verranno proposte nel caso in cui le disposizioni contengano due pezzi e non un numero arbitrario ma sarà facile generalizzarle nel caso in cui si vogliano disporre più di due pezzi contemporaneamente.

Distanza dei baricentri

Si vuole implementare in un modello come quelli visti precedentemente la funzione obiettivo

$$\|b_1 - b_2\|_2$$

dove i b_i sono i baricentri delle figure. Per fare questo una possibile soluzione è quella di calcolare, per ogni rotazione di entrambi i poligoni, i baricentri delle figure. Per cui sia

bar_{ir} posizione del baricentro del poligono i in modalità r (posizione relativa al punto di riferimento)

A questo punto la funzione obiettivo

$$\left\| \left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \sum_{r \in R_1} bar_{1r} z_{1r} \right) - \left(\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + \sum_{r \in R_2} bar_{2r} z_{2r} \right) \right\|_2$$

risulta quadratica e convessa.

Distanze tra i vertici dei rettangoli di contenimento

Il motivo che spinge ad analizzare una funzione obiettivo che considera le distanze tra i vertici dei rettangoli di contenimento è che affinché l'area del rettangolo che contiene entrambe le figure sia più piccola si può cercare di sovrapporre il più possibile i rettangoli di contenimento dei due pezzi. Per cui una volta calcolati i coefficienti

v_{ir}^{d} posizione del vertice d del rettangolo di contenimento di i in modalità r (posizione relativa al punto di riferimento)

I vertici sono numerati da 1 a 4 in senso antiorario a partire da quello in basso a sinistra. Si noti che

$$v_{ir}^{1} = 0 \qquad \forall i \in \{1,2\} \quad r =\in R_i$$

poiché il vertice in basso a sinistra è il punto di riferimento della figura. Ora la funzione obiettivo si scrive in modo quadratico e convesso come

$$\sum_{i=1}^{4}\sum_{j=1}^{4}\left\|\left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \sum_{r\in R_1} v_{1r}^{i} z_{1r}\right) - \left(\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + \sum_{r\in R_2} v_{2r}^{j} z_{2r}\right)\right\|_2$$

Distanza massima tra i vertici dei rettangoli di contenimento

L'idea è la stessa della variazione precedente ma questa volta si scegliere di minimizzare la massima tra le distanze dei vertici dei rettangoli di contenimento. Questo permette di formulare una funzione obiettivo quadratica e convessa nel modo seguente

$$\min u$$

con

$$\left\|\left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \sum_{r\in R_1} v_{1r}^{i} z_{1r}\right) - \left(\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + \sum_{r\in R_2} v_{2r}^{j} z_{2r}\right)\right\|_2 \leq u \qquad i,j \in \{1\ldots 4\}$$

Perimetro

Si cerca di minimizzare la funzione

$$\min\{\max\{l_{1r}, x_2 + \sum_{r\in R_2} l_{2r} z_{2r}\} - \min\{0, x_2\} + \max\{h_{1r}, y_2 + \sum_{r\in R_2} h_{2r} z_{2r}\} - \min\{0, y_2\}\}$$

Nel modello mostrato di seguito le disuguaglianze servono a delineare i ruoli delle variabili in gioco nella funzione obiettivo

$$min \quad x_M - x_m + y_M - y_m$$

$$x_2 + \sum_{r=1}^{R_2} l_{2r} z_{2r} \leq x_M \tag{3.34}$$

$$\sum_{r=1}^{R_1} l_{1r} z_{1r} \leq x_M \tag{3.35}$$

$$x_m \leq x_2 \tag{3.36}$$

$$x_m \leq 0 \tag{3.37}$$

$$y_2 + \sum_{r=1}^{R_2} h_{2r} z_{2r} \leq y_M \tag{3.38}$$

$$\sum_{r=1}^{R_1} h_{1r} z_{1r} \leq y_M \tag{3.39}$$

$$y_m \leq y_2 \tag{3.40}$$

$$y_m \leq 0 \tag{3.41}$$

$$\sum_{r=1}^{R_1} z_{1r} = 1 \tag{3.42}$$

$$\sum_{r=1}^{R_2} z_{2r} = 1 \tag{3.43}$$

$$z_{ir} \in \{0,1\} \qquad i = 1,2 \qquad r = 1 \dots R_i \tag{3.44}$$

Linearizzazione dell'area

Un'altra alternativa per approssimare al meglio la funzione obbiettivo non convessa potrebbe essere quella di osservare che le dimensioni del rettangolo di contenimento della disposizione non saranno tanto diverse da quelle relative al pezzo più grande tra i due. Perciò potrebbe valere la pena utilizzare l'approssimazione lineare della funzione

$$f(x,y) = xy \ \simeq \ h_0 x + l_0 y$$

con l_0 e h_0 rispettivamente la larghezza e l'altezza del rettangolo di contenimento del pezzo con area maggiore tra i due. Allora utilizzando la stessa notazione e gli stessi vincoli utilizzati per modellare il perimetro potremo

scrivere la funzione obiettivo

$$min \quad h_0 x_M - h_0 x_m + l_0 y_M - l_0 y_m$$

3.3.2 Vincoli di non sovrapposizione

In questo problema non vi sono vincoli di contenimento, gli unici necessari al modello sono quelli che non permettono le intersezioni tra i pezzi. Dato che nel problema contano solamente le posizioni relative dei pezzi ed inoltre tutti i pezzi devono essere disposti obbligatoriamente possiamo supporre che la posizione del primo fra questi sia fissata. A patto di restringerci ad avere le stesse rotazioni per ogni pezzo, possiamo fare lo stesso anche per la sua rotazione e questa scelta non influenza la funzione obiettivo della soluzione ottima.

I vincoli di non sovrapposizione sono grazie a questa semplificazione praticamente gli stessi del modello 3 per il caso di foglio di lavoro poligonale. Infatti il ruolo del complementare del foglio di lavoro viene preso dal pezzo con posizione e rotazione fissa e per questo utilizzando le seguenti notazioni per le variabili è possibile costruire il seguente modello:

NFP_P^{ir} no-fit polygon tra il pezzo fisso e il poligono i in modalità r

m_P^{ir} numero di componenti convesse di $(NFP_P^{ir})^c$

f_P^{ik} numero di vincoli della componente convessa k di $(NFP_P^{ir})^c$

$a_{ir}^{kf} x + b_{ir}^{kf} y \leq w_{ir}^{kf}$ equazione del vincolo numero f della componente convessa k di $(NFP_P^{ir})^c$

b_P^{irk} variabile binaria che vale 1 se il poligono i è posizionato nella componente convessa k di $(NFP_P^{ir})^c$ in modalità r

$$a_{ir}^{kf}(x_i) + b_{ir}^{kf}(y_i) \leq w_{ir}^{kf} + M(1 - b_P^{irk}) \qquad i \in P \quad r \in R_i \quad k = 1 \dots m_P^{ir} \quad f = 1 \dots f_P^{ik} \tag{3.45}$$

$$\alpha_{irjr'}^{kf}(x_j - x_i) + \beta_{irjr'}^{kf}(y_j - y_i) \leq w_{irjr'}^{kf} + M(1 - b_{irjr'k}) \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \quad k = 1 \dots m_{irjr'} \quad f = 1 \dots f_{irjr'}^{k} \tag{3.46}$$

$$\sum_{h=1}^{m_{irjr'}} b_{irjr'h} \geq z_{ir} + z_{jr'} - 1 \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{3.47}$$

$$\sum_{h=1}^{m_P^i} b_P^{irk} = z_{ir} \qquad i \in P \quad r \in R_i \tag{3.48}$$

$$\sum_{r \in R_i} z_{ir} \leq 1 \qquad i \in P \tag{3.49}$$

$$x_i, y_i \geq 0 \qquad i \in P \tag{3.50}$$

$$z_{ir} \in \{0,1\} \qquad i \in P \tag{3.51}$$

$$b_{irjr'h} \in \{0,1\} \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \quad h = 1 \dots m_{irjr'} \tag{3.52}$$

$$b_P^{irk} \in \{0,1\} \qquad i \in P \quad r \in R_i \qquad k = 1 \dots m_P^{ir} \tag{3.53}$$

3.3.3 Caso con 2 pezzi

Adesso ci restringeremo al caso in cui i pezzi da compattare siano solo due e forniremo due algoritmi per la risoluzione del problema. Il primo sarà compatibile con l'utilizzo con più di due pezzi mentre il secondo sfrutta questa restrizione per fare una correzione della soluzione finale che ha delle interessanti proprietà.

Metodo completamente modellistico

Il primo metodo segue la strada usata per i problemi visti precedentemente, e dunque considera i due poligoni ognuno con le sue rotazioni, uno dei due con la posizione fissata. Il modello diventa per cui

$$\min f$$

$$\alpha^{kf}_{1r2r'}(x_{2r'}) + \beta^{kf}_{1r2r'}(y_{2r'}) \leq w^{kf}_{1r2r'} + M(1 - b_{1r2r'k})$$
$$k = 1 \dots m_{1r2r'} \quad f = 1 \dots f^{k}_{1r2r'} \quad r \in R_i \quad r' \in R_j \tag{3.54}$$

$$\sum_{k=1}^{m_{1r2r'}} b_{1r2r'k} = 1 \tag{3.55}$$

$$\sum_{r \in R_i} z_{ir} = 1 \qquad i = 1, 2 \tag{3.56}$$

$$b_{1r2r'k} \in \{0, 1\} \qquad r = 1 \dots R_1 \quad r = 1 \dots R_2 \tag{3.57}$$

La funzione f da utilizzare può variare tra quelle viste sopra.

Metodo ibrido

Quest'altro metodo consiste in due fasi:

1. Si tiene fissa la posizione e la rotazione del primo poligono. Per ogni rotazione del secondo e per ogni componente convessa del di $NFP^{c}_{1r2r'}$ si risolve un problema di minimizzazione della funzione obiettivo.

2. Per ogni soluzione trovata si calcola il rettangolo di contenimento di area minima (non necessariamente con i lati paralleli agli assi)

In questo modo si sostituisce ad un approccio *Branch & Bound* una serie di problemi di *PL* (esattamente $\sum_{r'=1}^{R_2} m_{1r2r'}$ se per il primo pezzo è stata scelta la rotazione r). Si noti che il numero di problemi di programmazione lineare da risolvere è minore del numero di foglie dell' albero delle soluzioni del modello proposto al metodo modellistico, infatti in questo caso stiamo supponendo che il primo dei due oggetti abbia una rotazione fissa. Inoltre la seconda fase di questo metodo ibrido permette di esplorare soluzioni in cui le rotazioni non sono più discrete ma continue (calcolare il rettangolo di contenimento di costo minimo è un problema risolubile in costo lineare rispetto il numero di vertici del poligono) e questo permette di introdurre in un certo senso le rotazioni libere nel modello.

Capitolo 4

Algoritmi

In questa sezione diverse euristiche per cercare delle buone soluzioni al problema del packing bidimensionale utilizzando lo studio fatto fino a questo momento che ha portato all'implementazione di un algoritmo basato su modello MILP.

L'algoritmo scritto fino ad ora è quello più immediato, creare un modello MILP e utilizzare un solutore generico per la risoluzione di problemi di programmazione lineare-intera mista. Si tratta di un algoritmo di ricerca diretto. Sappiamo che questo metodo porta in un tempo finito alla soluzione esatta (disposizione con minor sfrido possibile), ma il tempo impiegato dalla computazione potrebbe risultare troppo elevato (infatti dai primi test è proprio così). Questa proprietà di esattezza in tempo finito non vuol dire che sia però il metodo migliore, infatti potrebbero esserci dei metodi che non hanno questa proprietà ma che a parità di tempo, anche relativamente breve, e nella maggior parte dei casi, riescono a fornire delle soluzioni migliori. In questo senso si può pensare di implementare un algoritmo ibrido esatto-euristico, che cerchi di diminuire la dimensione del problema da affidare al solutore, facendo delle scelte che promettono di essere vantaggiose e che restringono lo spazio delle soluzioni che il solutore deve visitare.

4.0.1 Modularizzazione degli algoritmi

Per facilitare la fase di sviluppo e test procederemo con la scelta e l'implementazione di diversi moduli che rappresentano altrettanto diversi approcci per la risoluzione del problema del packing. Questi potranno essere usati e combinati in sequenza per trovare una buona soluzione. Abbiamo già studiato uno di questi moduli, quello che fa uso del modello MILP. I moduli condivideranno uno schema generale che consiste in:

- Ogni modulo sarà sottoclasse di Packing2D_di_unipi_it::PackingAlgo.

- Ogni modulo prende come input un insieme di pezzi e un foglio di lavoro. Il foglio di lavoro consiste in rettangolo e in una soluzione parziale. Il primo modulo che lavorerà sull'istanza avrà in input i pezzi da disporre e la soluzione parziale in cui nessun pezzo viene piazzato. Queste informazioni sono già tutte implementate nella classe Packing2D_di_unipi_it::Sheet.

- Restituisce un oggetto della classe Packing2D_di_unipi_it::Solution che comprende informazioni sul foglio di lavoro, i pezzi disposti dal modulo e quelli disposti da i moduli precedenti.

Esempi di moduli:

MilpPackingAlgo modulo che implementa l'algoritmo basato su No-Fit Polygon e solutore MILP, cerca di piazzare più pezzi possibile tra quelli che ottiene in input. Questo modulo utilizza il modello 1 se viene lanciato su una soluzione vuota (foglio di lavoro rettangolare), altrimenti viene utilizzato il modello 3 prendendo come foglio di lavoro il complementare nel rettangolo iniziale dei pezzi già disposti.

MilpBottomLeftAlgo modulo che implementa l'algoritmo che dispone i pezzi ordinatamente uno alla volta nella posizione più bassa e a sinistra disponibile utilizzando il No-Fit Polygon e il solutore MILP. Considera anche le diverse rotazioni del pezzo (discrete) e i pezzi vengono disposti in ordine di area decrescente. Questo modulo utilizza il modulo precedente MilpPackingAlgo per disporre uno alla volta i pezzi.

Pack&StockAlgo modulo che prevede una fase di "Pack" in cui si cercano delle combinazioni di pezzi che approssimano rettangoli e una seconda fase di "Stock" che esegue un packing di rettangoli. Questo modulo necessita di lavorare con un foglio rettangolare per sfruttare le proprietà utili del packing di rettangoli.

StretchAndFillAlgo modulo che cerca di riempire i buchi lasciati da una soluzione parziale, concedendosi la facoltà di allargare i buchi per migliorare il nesting senza però modificare i posizionamenti relativi tra i pezzi già disposti.

FillHardestFirstAlgo modulo che cerca di riempire i buchi lasciati da una soluzione parziale dando priorità a quelli che possono essere utilizzati da pochi tra i pezzi a disposizione.

Ora vedremo di approfondire gli algoritmi che i moduli implementano.

4.0.2 Pack&StockAlgo

Propone una strategia in più fasi:

1. **Ricerca combinazioni promettenti** Per ogni coppia o terna (dipende da quanto risulta veloce il calcolo) di pezzi da disporre risolviamo attraverso il solutore MILP il problema di trovare la disposizione di questi pezzi in modo da minimizzare l'area del rettangolo di contenimento della dispozione meno l'area dei pezzi (funzione obiettivo quadratica non convessa). Per fare questo è sufficiente fissare la posizione di uno dei pezzi e la sua rotazione e invece lasciare agli altri pezzi libertà di posizione e rotazione (sempre discrete). Per creare questi modelli MILP non abbiamo bisogno di conoscenze diverse da quelle che abbiamo usato fin'ora (NFP, slices, creazione del modello, uso del solutore). In questa fase potrebbe essere decisivo l'ordine con cui si scorrono i pezzi per trovare le combinazioni promettenti, si dovranno fare dei test per capire come è più vantaggioso ordinarli. La funzione obbiettivo essendo non convessa deve essere approssimata con una convessa, in questo caso abbiamo provato diverse tra le varianti proposte nel capitolo 3. Inizialmente è stata provata la funzione obbiettivo che considera il perimetro sommata con quella che considera la distanza tra i baricentri moltiplicata per un coefficiente piccolo. Questo permette di dare priorità al perimetro e tra le soluzioni a parità di perimetro preferire quelle con distanza tra i baricentri più bassa. I risultati proposti utilizzando questa funzione obbiettivo ci hanno fatto pensare all'alternativa proposta in 3.3.1 che ha dimostrato di poter dare risultati significativamente migliori come in figura:

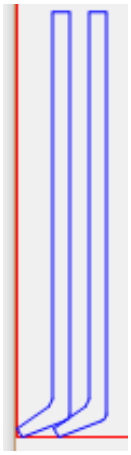


Figura 4.1: Funzione obbiettivo perimetro

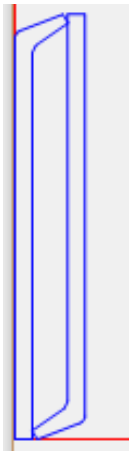


Figura 4.2: Funzione obbiettivo linearizzazione area

Un altra possibilità è quella di cercare, sempre attraverso il modello, di saturare i rettangoli di contenimento. Questa è una alternativa computazionalmente più leggera ma anche più restrittiva nelle soluzioni effettivamente esplorate. Quale tra le due idee sia migliore dipende dalle prestazioni che danno i due algoritmi.

2. **Impaccamento delle combinazioni** Considero le combinazioni più promettenti come un unico pezzo rettangolare (il suo rettangolo di contenimento) e provo al risolvere il problema usando un algoritmo di impaccamento per rettangoli.

Durante la fase 1 è possibile che alcuni pezzi non vengano inseriti in nessun accoppiamento, questo può essere dovuto o al fatto che non vi sono altri pezzi con cui accoppiarli, oppure al fatto che hanno già una forma vicina a quella di un rettangolo e quindi considerarli come pezzi a se stanti nell'impaccamento di rettangoli non è evidentemente svantaggioso.

Il modello che risolve la fase 1 risulta essere:

$$\min \qquad h_p(\hat{X}_{\max} - \hat{X}_{\min}) + l_p(\hat{Y}_{\max} - \hat{Y}_{\min})$$

$$a_{ir}^{kf}(x_i) + b_{ir}^{kf}(y_i) \leq w_{ir}^{kf} + M(1 - b_P^{irk}) \qquad i \in P \quad r \in R_i \quad k = 1 \dots m_P^i \quad f = 1 \dots f_P^{ik} \tag{4.1}$$

$$\alpha_{irjr'}^{kf}(x_j - x_i) + \beta_{irjr'}^{kf}(y_j - y_i) \leq M(1 - b_{irjr'jh}) \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \quad k = 1 \dots m_{irjr'} \quad f = 1 \dots f_{irjr'}^{k} \tag{4.2}$$

$$\sum_{h=1}^{m_{irjr'}} b_{irjr'h} \geq z_{ir} + z_{jr'} - 1 \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \tag{4.3}$$

$$\sum_{h=1}^{m_P^i} b_P^{irh} = z_{ir} \qquad i \in P \quad r \in R_i \tag{4.4}$$

$$\sum_{r \in R_i} z_{ir} \leq 1 \qquad i \in P \tag{4.5}$$

$$\sum_{i \in P}(x_i + \sum_{r \in R_i} z_{ir} l_{ir}) \leq \hat{X}_{\max} \tag{4.6}$$

$$\sum_{i \in P}(y_i + \sum_{r \in R_i} z_{ir} h_{ir}) \leq \hat{Y}_{\max} \tag{4.7}$$

$$\sum_{i \in P} x_i \geq \hat{X}_{\min} \tag{4.8}$$

$$\sum_{i \in P} y_i \geq \hat{Y}_{\min} \tag{4.9}$$

$$z_{ir} \in \{0,1\} \qquad i \in P \tag{4.10}$$

$$b_{irjr'h} \in \{0,1\} \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P \quad h = 1 \dots m_{irjr'} \tag{4.11}$$

$$\hat{X} \geq l_p \tag{4.12}$$

$$\hat{Y} \geq h_p \tag{4.13}$$

$$\hat{X}_{\min}, \hat{Y}_{\min} \leq 0 \tag{4.14}$$

Nel modello descritto qua sopra si può notare la somiglianza con quello relativo al caso con foglio poligonale visto precedentemente solo che il ruolo dello spazio occupato nel foglio è preso dal pezzo che ha posizione e rotazio-

ne fissa attorno al quale possono "orbitare" gli altri pezzi. Nel modello le costanti l_p e h_p sono rispettivamente la larghezza e l'altezza del pezzo fisso, mentre le variabili $\hat{X}_{\max}$, $\hat{Y}_{\max}$, $\hat{X}_{\min}$ e $\hat{Y}_{\min}$ rappresentano larghezza ed altezza del rettangolo di contenimento della disposizione infatti nei vincoli (57), (58), (59) e (60) tali variabili sono costrette a contenere nei loro intervalli l'ingombro dei pezzi disposti. Nei vincoli (63), (64) e (65) si impone invece che tali variabili contengano quantomeno le dimensioni del pezzo fisso.

4.0.3 StretchAndFillAlgo

L'implementazione di questo modulo prevede l'utilizzo del solutore MILP a cui viene assegnato il modello simile quello di MilpPackingAlgo, tranne per il fatto che le posizioni dei pezzi già disposti (quelli della soluzione parziale data in input) non sono fisse ma possono variare, a patto di non modificare la soluzione intera relativa (ossia le variabili relative alla disposizione in una certa rotazione, e quelle che controllano le posizioni relative tra i pezzi). Per fare questo occorre calcolare, a partire dalla soluzione parziale, le posizione relative tra i pezzi. Quindi per ogni coppia di pezzi già disposti bisogna capire in quale slice giace uno rispetto al No-Fit Polygon con l'altro (si può scegliere se ricalcolare gli NFP e le slices relative oppure utilizzare quelli già computati dai moduli che hanno disposto in precedenza i pezzi). Dopodichè bisogna costruire il modello che per ogni pezzo disposto contiene le variabili di posizionamento e per ogni coppia di questi, contiene le equazioni di contenimento nella slice scelta dalla soluzione data in input. Il modello conterrà inoltre, per ogni nuovo pezzo, le variabili di posizione, scelta della rotazione,scelta della slice, e i vincoli di non sovrapposizione non solo con tutti gli altri nuovi pezzi ma anche relativi a quelli già disposti dalla soluzione data in input. Perciò questo modulo si riduce a creare tale modello ed utilizzare il solutore MILP per risolverlo.

$$\max \quad \sum_{i \in P} \sum_{r \in R_i} a_i z_{ir}$$

$$x_i \leq (X_F - \sum_{r \in R_i} l_{ir} z_{ir}) \qquad i \in P_{new} \quad (4.15)$$

$$y_i \leq (Y_F - \sum_{r \in R_i} h_{ir} z_{ir}) \qquad i \in P_{new} \quad (4.16)$$

$$x_i \leq (X_F - l_{ir_c}) \qquad i \in P_{old} \quad (4.17)$$

$$y_i \leq (Y_F - h_{ir_c}) \qquad i \in P_{new} \quad (4.18)$$

$$\alpha^{kf}_{irjr'} (x_j - x_i) + \beta^{kf}_{irjr'} (y_j - y_i) \leq w^{kf}_{irjr'} + M(1 - b_{irjr'h})$$
$$i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P_{new} \quad k = 1 \dots m_{irjr'} \quad f = 1 \dots f^k_{irjr'} \quad (4.19)$$

$$\alpha^{kf}_{irjr_c} (x_j - x_i) + \beta^{kf}_{irjr_c} (y_j - y_i) \leq w^{kf}_{irjr_c} + M(1 - b_{irjr_ch})$$
$$r \in R_i \quad i \in P_{new} \quad j \in P_{old} \quad k = 1 \dots m_{irjr'} \quad f = 1 \dots f^k_{irjr_c} \quad (4.20)$$

$$\alpha^{k_cf}_{irjr'} (x_j - x_i) + \beta^{k_cf}_{irjr'} (y_j - y_i) \leq w^{k_cf}_{irjr'}$$
$$i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P_{old} \quad f = 1 \dots f^{k_c}_{irjr'} \quad (4.21)$$

$$\sum_{h=1}^{m_{irjr'}} b_{irjr'h} \geq z_{ir} + z_{jr'} - 1 \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P_{new} \quad (4.22)$$

$$\sum_{h=1}^{m_{irjr_c}} b_{irjr_ch} \geq z_{ir} \qquad r \in R_i \quad i \in P_{new} \quad j \in P_{old} \quad (4.23)$$

$$\sum_{r \in R_i} z_{ir} \leq 1 \qquad i \in P_{new} \quad (4.24)$$

$$x_i, y_i \geq 0 \qquad i \in P \quad (4.25)$$

$$z_{ir} \in \{0,1\} \qquad i \in P_{new} \quad (4.26)$$

$$b_{irjr'h} \in \{0,1\} \qquad i < j \quad r \in R_i \quad r' \in R_j \quad i,j \in P_{new} \quad h = 1 \dots m_{irjr'} \quad (4.27)$$

Nel modello appena descritto gli insiemi P_{new} e P_{old} rappresentano rispettivamente i pezzi da disporre e quelli già posizionati nella soluzione data in input, mentre P è l'unione di questi ultimi. l_{ir_c} h_{ir_c} sono larghezza ed altezza

del pezzo i nella modalità fornita dalla soluzione in input, mentre il numero k_c rappresenta l'indice della slice scelta dai moduli precedenti.

4.0.4 FillHardestFirstAlgo

In questo modulo i buchi vengono saturati uno alla volta, attraverso lo studio del No-Fit Polygon tra ogni pezzo e il complementare del buco. Viene costruita una matrice binaria in cui le colonne rappresentano i buchi della disposizione e le righe rappresentano i pezzi da disporre. L'elemeno (i, j) della matrice è 1 se il pezzo i può essere disposto nel buco j e 0 altrimenti. Si sceglie la colonna che ha maggior numero di zeri e si cerca di riempire il buco relativo a tale colonna. Il modo in cui si sceglie di riempirlo è arbitrario e sarà implementato in modo parametrico, può essere usato uno qualsiasi degli altri moduli come ad esempio il MilpPackingAlgo. Nel caso i pezzi che possono essere disposti in un buco siano troppi si sceglieranno quelli di area maggiore. Dopo che si è disposto un pezzo si può aggiornare la matrice levando la colonna relativa al buco riempito e la riga relativa al pezzo disposto ed inoltre aggiungere le colonne relative ai buchi che si possono essere creati una volta aggiunto il pezzo. Queste colonne devono essere aggiornate controllando quali dei pezzi, che prima potevano essere disposti nel buco eliminato, posso ora essere disposti nel buco relativo.

Capitolo 5

Implementazione

5.1 La libreria Boost

Il linguaggio scelto per l'implementazione è stato il C++ e le sue librerie standard insieme alla libreria Boost. Quest'ultima ha fornito importanti strumenti tra cui principalmente l'implementazione di classi per poligoni e insiemi poligonali. Questa doppia rappresentazione ha permesso da una parte di trattare i poligoni come insiemi ordinati di vertici e dall'altra come insiemi di punti nel piano con le operazioni di unione, sottrazione insiemistica e calcolo di aree. Approcciarsi con la libreria Boost ha richiesto però l'utilizzo di coordinate intere per l'implementazione dei poligoni e questo ha portato a lavorare su due unità di misura diverse. Infatti l'uso di coordinate intere per l'intera implementazione sarebbe stato inappropriato poiché si sarebbe persa l'importante caratteristica di prevedere traslazioni continue dei pezzi ed inoltre avrebbe causato un ulteriore approssimazione nella rappresentazione delle figure da disporre. Per cui è stato utilizzato un tipo intero chiamato "coordinate" utilizzato per gli oggetti delle classi di Boost, tra cui i poligoni, ed un tipo double per il resto degli oggetti come ad esempio i poligoni con archi, i fogli di lavoro e le variabili relative ai modelli. La conversione tra le unità di misura è gestita dalle due funzioni

- const coordinate **Packing2D_di_unipi_it::dtc** (const double &a)
- const double **Packing2D_di_unipi_it::ctd** (const coordinate &a)

Per avere una maggiore precisione nel approcciarsi con i poligoni di Boost si è scelto di variare la scala del'unità "coordinate" infatti il numero 1 in double corrisponde al numero 10 in "coordinate" cosicché se l'unità "double" rappresenta il millimetro , quella "coordinate" riesce a descrivere anche

variazioni dell ordine del decimo di millimetro mantenendo l'importante caratteristica di interezza richiesta da Boost. Questa libreria ha inoltre messo a disposizione una implementazione della somma di Minkowski tra due poligoni, strumento fondamentale, come visto nei capitoli precedenti, per calcolare gli NFP e quindi i dati per la creazione dei modelli MILP.

Nelle prossime sezioni riporteremo poche tra le principali classi che sono state implementate ed utilizzate nel progetto. Per ogni classe verrà riportata una descrizione e la sua interfaccia in modo da mostrarne le caratteristiche principali.

5.2 Packing2D_di_unipi_it::ArcPolygon

5.2.1 Descrizione

Classe ArcPolygon: implementa il concetto di poligono con archi. Il poligono con archi viene salvato sia come vettore di lati (istanze della classe Lato) ma anche attraverso una sua approssimazione come poligono di Boost.

Metodi publici

- **ArcPolygon** (const vector< **Lato** > &nlati)

  Costruttore principale della classe ArcPolygon: l'input deve essere un vettore di lati tali che il punto iniziale del (i+1)-esimo lato sia il punto finale del (i)-esimo. Il punto finale dell' ultimo lato deve coincidere con il punto iniziale del primo. I lati devono essere forniti con i dati in scala normale, quella di input

- const vector< **Lato** > & **Readlati** (void) const

  Lettura vettore di lati.

- const Polygon * **Readpolygon** (void) const

  Lettura poligono di approssimazione.

- double **Readarea** (void) const

  Lettura area.

- Polygon * **Approximate** (double alpha)

  Restituisce un poligono che approssima l'oggetto. Il parametro "alpha" decide la massima lunghezza che possono avere i segmenti che approssimano gli archi. Il poligono viene restituito con coordinate in scala 'coordinate'.

5.2.2 Attributi

amount
`int Packing2D_di_unipi_it::ArcPolygon::amount [protected]`
indica quante copie del poligono sono da disporre.

area
`double Packing2D_di_unipi_it::ArcPolygon::area [protected]`
valore dell'area del poligono con archi.

lati
`vector<Lato> Packing2D_di_unipi_it::ArcPolygon::lati [protected]`
vettore di oggetti "Lato" che descrivono la frontiera del poligono con archi. I lati sono disposti in ordine di percorrenza orario e il punto finale di un lato deve corrispondere con il punto iniziale del successivo.

polygon
`const Polygon* Packing2D_di_unipi_it::ArcPolygon::polygon [protected]`
contiene un poligono che approssima il poligono con archi.

5.3 Packing2D_di_unipi_it::NFP

5.3.1 Descrizione

Classe NFP: implementa il concetto di No-Fit Polygon tra due poligoni. Memorizza sia il poligono dell'NFP ma anche le slice della decomposizione in convessi del suo complementare e le equazioni dei loro lati.

Metodi Pubblici

- **NFP** (const Polygon &a, const Polygon &B)

  descrizione nel paragrafo successivo.
- **NFP** (const Sheet &a, const Polygon &B)

  descrizione nel paragrafo successivo.
- **NFP** (const Sheet &a, const Polygon &B, string stringa)

  descrizione nel paragrafo successivo.
- vector< Slice > const * **Readslices** (void) const

  Lettura slices.
- vector< Polygon > * **ReadPolygon** (void) const

Lettura poligono dell'NFP.

5.3.2 Costruttori e distruttori

NFP() [1/3]

```
Packing2D_di_unipi_it::NFP::NFP (
            const Polygon & a,
            const Polygon & B )
```

costruttore principale per la classe NFP, prende come input i due poligoni.

Parameters

a	primo poligono (fisso)
B	secondo poligono (orbitante)

NFP() [2/3]

```
Packing2D_di_unipi_it::NFP::NFP (
            const Sheet & a,
            const Polygon & B )
```

costruttore per la classe NFP per il caso di foglio poligonale , crea l'NFP tra il pezzo e il complementare degli spazi liberi nel foglio.

Parameters

a	foglio che identifica gli spazi liberi
B	poligono

NFP() [3/3]

```
NFP::NFP (
            const Sheet & a,
            const Polygon & B,
            string tstring )
```

costruttore per la classe NFP per il caso di foglio poligonale , crea l'NFP tra il pezzo e il complementare degli spazi liberi nel foglio (non si interessa di creare le decomposizioni in convessi del complementare o di trovare le equazioni utili al modello). Utilizzato perchè più veloce rispetto al precedente costruttore.

Parameters

a	foglio che identifica gli spazi liberi

Parameters

B	poligono
tstring	stringa per evitare overriding

5.3.3 Attributi

nfp_polygon

`vector<Polygon >* Packing2D_di_unipi_it::NFP::nfp_polygon =new vector<Polygon >() [protected]`

contiene il poligono che descrive l NFP.

slices

`vector< Slice > * Packing2D_di_unipi_it::NFP::slices [protected]`

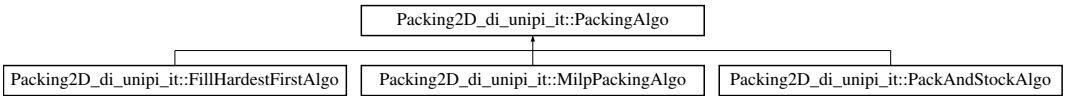
vettore che contiene tutte le slices del No-Fit Polygon in questione.

5.4 Packing2D_di_unipi_it::PackingAlgo

5.4.1 Descrizione

Classe PackingAlgo: implementa la classe astratta PackingAlgo, che definisce l'interfaccia di un generico solutore di Packing2D. Questa classe non implementa la maggior parte dei metodi fondamentali come l'Input() e il Solve() che sono dichiarati "pure virtual". Infatti questa classe non verrá mai utilizzata direttamente ma solo attraverso le sue sottoclassi che implementeranno i diversi algoritmi di packing.

Gerarchia delle sottoclassi di Packing2D_di_unipi_it::PackingAlgo:



Tipi pubblici

- enum PAParam { kMaxTime = 0, kLastParam }

Metodi pubblici

Costruttori

- **PackingAlgo** ()

  costruttore principale.

Altre inizializzazioni

- virtual void **Input** (vector< ArcPolygon const *> *pezzi, Sheet *const foglio)=0
- void **SetPar** (int par, int val)
- void **SetPar** (int par, double val)

Risolvere il problema

- virtual void **Solve** (void)=0

Leggere la soluzione

- virtual const Solution * **ReadSolution** ()=0

Leggere i parametri impostati

- const vector< ArcPolygon const * > * **ReadPieces** (void)
- Sheet const *const **ReadSheet** (void)
- virtual void **GetPar** (int par, int &val)
- virtual void **GetPar** (int par, double &val)

Distruttori

- virtual ~**PackingAlgo** ()

5.4.2 Descrizione tipi enumerati

PAParam

`enum Packing2D_di_unipi_it::PackingAlgo::PAParam`

enumerate pubblico che descrive i possibili parametri impostabili. SetPar() e GetPar().

Enumerate

kMaxTime	Tempo massimo.
kLastParam	parametro test. Serve per permettere alle sottoclassi di estendere l'elenco dei parametri

5.4.3 Descrizione Metodi

GetPar() [1/2]

```
void Packing2D_di_unipi_it::PackingAlgo::GetPar (
            int par,
            int & val ) [inline], [virtual]
```

restituisce uno dei parametri interi impostati nel modulo.

Parameters

par	è il parametro da restituire;
val	variabile su cui viene salvato il parametro richiesto.

GetPar() [2/2]

```
void Packing2D_di_unipi_it::PackingAlgo::GetPar (
            int par,
            double & val ) [inline], [virtual]
```

restituisce uno dei parametri double impostati nel modulo.

Parameters

par	è il parametro da restituire;
val	variabile su cui viene salvato il parametro richiesto.

Input()

```
virtual void Packing2D_di_unipi_it::PackingAlgo::Input (
            vector< ArcPolygon const *> * pezzi,
            Sheet *const foglio ) [pure virtual]
```

permette di impostare i dati dell'istanza, ossia i pezzi da piazzare ed il foglio su cui farlo. Il metodo prende *puntatori* all'informazione, che puo' essere solamente letta ma non modificata dall'oggetto. La memoria per tali oggetti deve quindi essere allocata prima della creazione dell'oggetto, non essere deallocata finché l'oggetto non è deallocato, e poi deve essere deallocata dal chiamante.

Parameters

pezzi	parametro che identifica il vettore di pezzi da disporre.
foglio	parametro che identifica il foglio su cui disporre i pezzi.

ReadPieces()

```
const vector<ArcPolygon const *>* Packing2D_di_unipi_it::PackingAlgo::Read↩
Pieces (
            void  )  [inline]
```

restituisce i pezzi che descrivono l'istanza del problema.

ReadSheet()

```
Sheet const* const Packing2D_di_unipi_it::PackingAlgo::ReadSheet (
            void  )  [inline]
```

restituisce il foglio di lavoro.

ReadSolution()

```
virtual const Solution* Packing2D_di_unipi_it::PackingAlgo::ReadSolution ( )
[pure virtual]
```

restitruisce la miglior soluzione trovata.

SetPar() [1/2]

```
void Packing2D_di_unipi_it::PackingAlgo::SetPar (
            int par,
            int val )  [inline]
```

imposta i parametri interi del modulo.

Parameters

par	Nome del parametro da impostare; Puó essere usato il PAParam per identificare il parametro.
value	il parametro intero da impostare.

questa classe non implementa parametri interi ma la definizione di questa funzione serve da scheletro per gli override delle sottoclassi.

SetPar() [2/2]

```
void Packing2D_di_unipi_it::PackingAlgo::SetPar (
            int par,
            double val )  [inline]
```

imposta i parametri double del modulo.

Parameters

par	Nome del parametro da impostare; Puó essere usato il tipo enumerato PAParam per identificare il parametro.

Parameters

value	parametro intero da impostare.

-kMaxTime: parametro che imposta il tempo massimo che il solutore ha per trovare una soluzione.

Solve()

```
virtual void Packing2D_di_unipi_it::PackingAlgo::Solve (
            void  )  [pure virtual]
```

effettua una certa quantitá di computazione (controllata in genere dai parametri) nella ricerca di una soluzione. I suoi parametri dipendono dallo specifico algoritmo (sottoclasse). Il metodo solve() puó essere lanciato piú volte per migliorare la soluzione trovata.

5.4.4 Attributi

MaxTime

```
double Packing2D_di_unipi_it::PackingAlgo::MaxTime  [protected]
```

memorizza il parametro che identifica il tempo massimo di computazione.

pieces

```
const vector<ArcPolygon const *>* Packing2D_di_unipi_it::PackingAlgo::pieces
[protected]
```

vettore che memorizza i pezzi da disporre.

sheet

```
Sheet const* Packing2D_di_unipi_it::PackingAlgo::sheet  [protected]
```

memorizza il foglio di lavoro dell'istanza

Per trovare maggiori informazioni sull'implementazione devo rimandare alla documentazione del software. La parte implementativa ha rappresentato la maggior parte dello sviluppo del progetto ed è difficile far trasparire il lavoro svolto in una tesi di Laurea Triennale in Matematica senza tediare irrimediabilmente il lettore.

5.5 Interfacciarsi con il solutore

Per la risoluzione dei modelli misti lineari-interi abbiamo bisogno di utilizzare un solutore generico di problemi MILP. Tra quelli disponibili al momento abbiamo scelto di utlizzarne 2 in particolare:

Cbc (Coin-or branch and cut) solutore Open-Source che fa parte del Coin-or Project. Si tratta di un software che implementa un algoritmo Branch & Bound per istanze di programmazione lineare intera mista. Cbc utilizza le librerie di Clp per la risoluzione dei rilassamenti continui, Cgl per la generazione dei tagli utili per restringere le soluzioni continue, e Coin-Utilis per gestire in modo semplice i dati in input e in output.

Gurobi motore sotto licenza utilizzato da più di 1400 compagnie nel mondo per ottimizzare decisioni nell'industria. Gurobi non risolve solo modelli MILP ma anche problemi che si posso ricondurre a MIQP (mixed-integer quadratic programming) e MIQCP (mixed-integer quadratically constrained programming).

La decisione di provare entrambi i software è stata presa per poter testare il differente comportamento dei due solutori. Per interfacciarsi con Gurobi si è utilizzata l'interfaccia OsiSolverInterface, sviluppata all'interno del progetto Coin-or, e compatibile ovviamente anche con Cbc. Questo ha permesso di interfacciarsi con la stessa sintassi con entrambi i solutori e quindi di poter intercambiare in modo facile i due software.

Le interazioni che abbiamo dovuto implementare con il solutore sono state principalmente quelle che hanno permesso di creare il modello (dichiarazioni di variabili, vincoli e funzione obiettivo), impostare alcuni parametri (tempo massimo di calcolo e massimo numero di nodi da visitare durante la ricerca di una soluzione) e leggere la migliore soluzione. Il software restituisce la soluzione attraverso un vettore di "double" che rappresenta gli assegnamenti migliori per le variabili. Quindi è stato necessario implementare una traduzione dei dati riportati nel vettore per creare una istanza della classe "Solution" che presenta una visualizzazione più intuitiva e accessibile delle caratteristiche della soluzione trovata.

Capitolo 6

Sperimentazione

Gli algoritmi che sono stati implementati fino a questo momento sono il MilpPackingAlgo, il Pack&StockAlgo e il FillHardestFirstAlgo. In questo capitolo mostreremo come lavorano questi moduli provando a concatenarne l'utilizzo per avere un algoritmo computazionalmente efficiente.

6.1 MilpPackingAlgo

In questa sezione riporteremo i risultati di alcuni test svolti sull'algoritmo MilpPackingAlgo. Come evidenzia il grafico qui riportato la strategia di utilizzare il modulo MilpPackingAlgo su tutti i pezzi dati in input risulta essere decisamente computazionalmente troppo costosa. Questo è dovuto al fatto

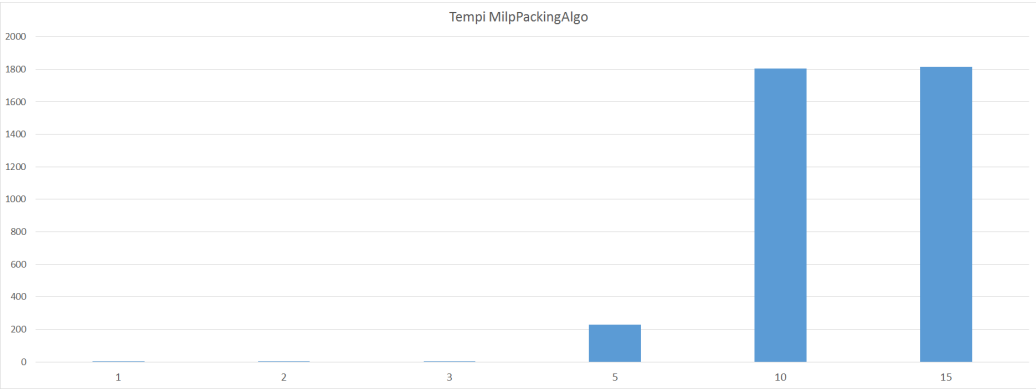


Figura 6.1: tempi in funzione del numero di pezzi dai in input

che il numero di variabili intere, indice significativo della complessità di un modello, crescono come (numero di pezzi $\cdot$ numero di rotazioni)^2. Proveremo

a testare due diverse strategie, la prima quella di utilizzare il modulo MilpPackingAlgo lanciato su più pezzi contemporaneamente, contro quella che invece lancia più volte il MilpPackingAlgo su meno pezzi. La prima volta il modulo lavorerà utilizzando il modello 1 per il foglio rettangolare, le successive volte calcolerà gli spazi liberi della disposizione appena creata per lanciare il riempimento con il modello 3. Le posizioni dei pezzi non potranno essere modificate dai moduli che agiranno successivamente. In questi casi saranno possibili 4 rotazioni per ogni pezzo (0,90,180 e 270 gradi) e i risultati sono i seguenti:

n° pezzi	n° pezzi contemporanei	n° rotazioni	secondi
51	1	4	434

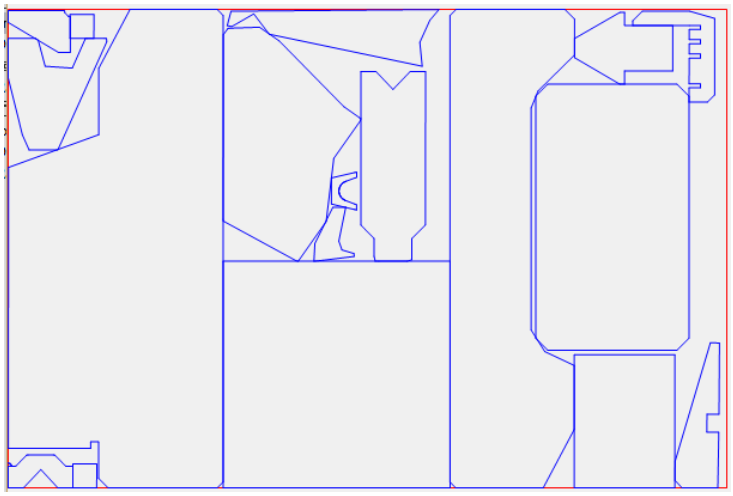


Figura 6.2: MilpPackingAlgo Test 1

n° pezzi	n° pezzi contemporanei	n° rotazioni	secondi
51	5	4	614

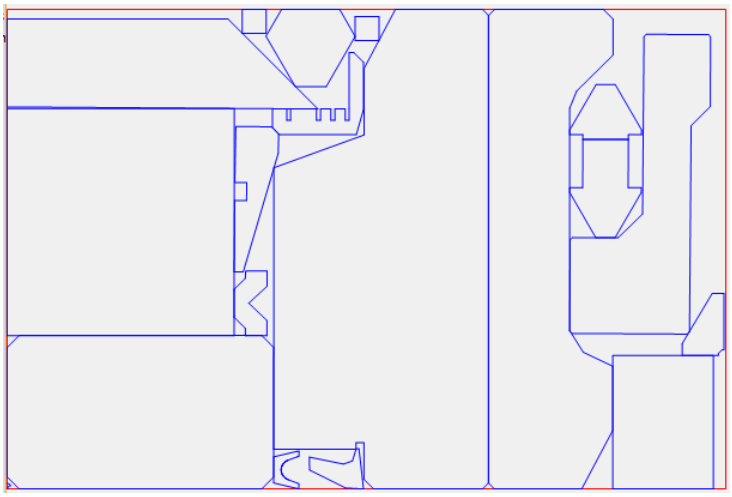


Figura 6.3: MilpPackingAlgo Test 2

Da questi due esempi si nota come le disposizioni dei pezzi di area maggiore sia più riuscita nel secondo caso, ma la qualità della soluzione non è sensibilmente diversa, i pezzi piccoli aiutano a compensare l'approssimazione nel posizionamento dei primi pezzi.

Infine un ultimo esempio per mostrare come lavora l'algoritmo con pezzi di dimensioni diverse e tempi più prolungati.

n° pezzi	n° pezzi contemporanei	n° rotazioni	secondi
51	5	4	813

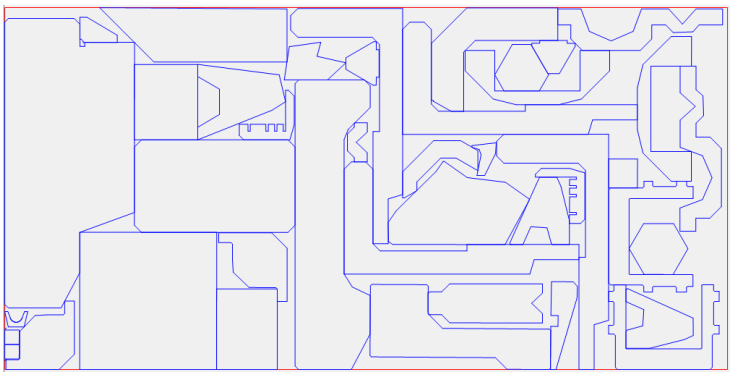


Figura 6.4: MilpPackingAlgo Test 3

6.2 Risultati test

A seguire sono raccolti i dati che descrivono i test effettuati su un campione di 20 istanze proposte da Intersystems, azienda informatica italiana che ha proposto il lavoro di ricerca di questa tesi.

Tabella 6.1: MilpPackingAlgo - 3 pezzi per volta, 4 rotazioni

nome test	pezzi in input	pezzi disposti	percentuale area totale	tempo totale
Test1	89	22	91,792375	241,963
Test2	43	28	70,20875	322,226
Test3	51	36	88,77872222	750,57
Test4	54	54	57,48232143	3186,773
Test5	56	35	86,19885714	327,011
Test6	113	28	92,2024	636,568
Test7	50	50	75,99333333	2036,448
Test8	66	31	86,58048	588,513
Test9	97	33	88,78386667	515,773
Test10	104	22	67,64336	1730,694
Test11	100	28	93,89448	517,294
Test12	100	45	87,16872	2279,072
Test13	100	42	82,01496	931,6
Test14	117	45	87,75536	714,517
Test15	112	40	85,53824	192,622
Test16	108	35	91,78456	289,787
Test17	112	45	86,33592	1087,037
Test18	117	22	87,38304	86,091
Test19	90	20	79,63088	1006,94
Test20	90	46	77,09688	1074,343

Tabella 6.2: FillHardestFirstAlgo

nome test	pezzi in input	pezzi disposti	percentuale area totale	tempo totale
Test1	89	23	90,0345	243,369
Test2	43	27	66,864375	91,828
Test3	51	28	90,35738889	52,519
Test4	54	49	56,66367857	621,645
Test5	56	37	89,32102041	326,459
Test6	113	28	93,2029	214,615
Test7	50	50	75,99333333	406,997
Test8	66	26	86,00944	559,699
Test9	97	27	86,21333333	194,82
Test10	104	26	70,04976	791
Test11	100	27	93,64096	259,477
Test12	100	36	84,5068	243,151
Test13	100	28	80,10536	179,432
Test14	117	42	89,03584	86,036
Test15	112	37	83,9884	130,327
Test16	108	32	91,27704	74,841
Test17	112	31	83,73936	182,393
Test18	117	22	87,57552	67,194
Test19	90	22	80,70416	198,382
Test20	90	41	75,7336	525,601

Qua invece vengono proposti i risultati del modulo FillHardestFirstAlgo sulle stesse istanze.

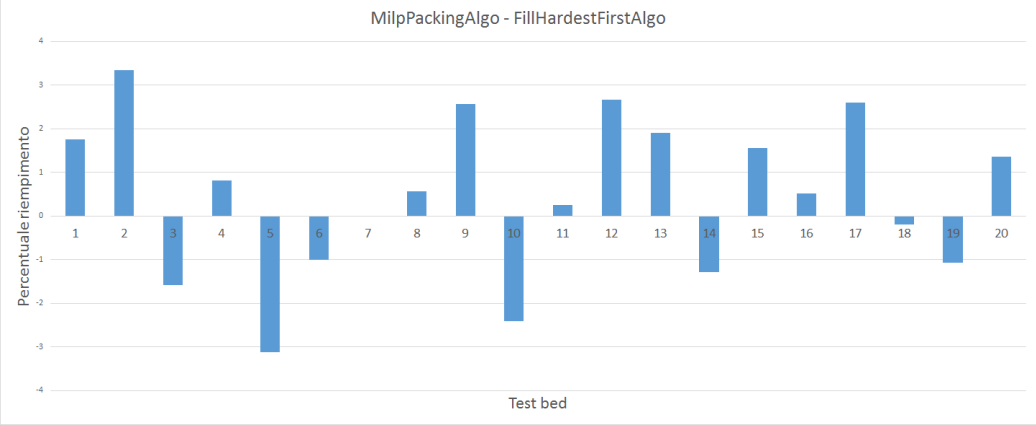


Figura 6.5: differenza tra le percentuali di riempimento di MilpPackingAlgo e FillHardestFirstAlgo

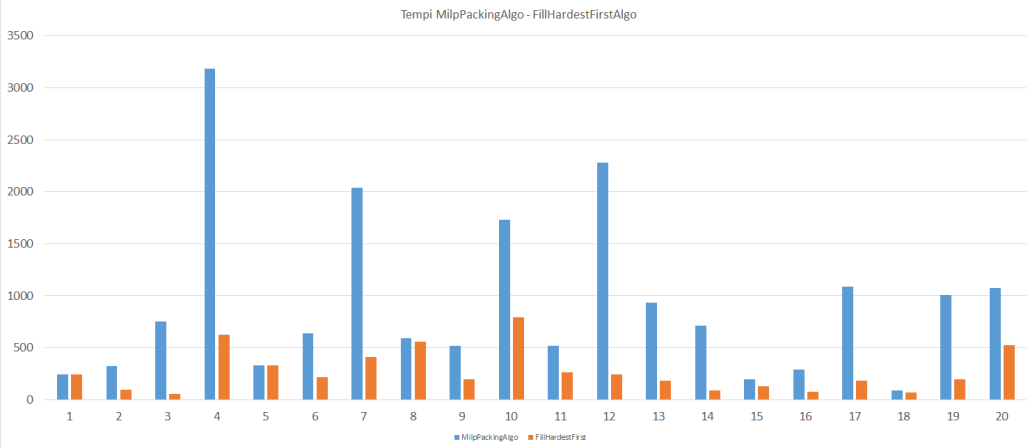


Figura 6.6: tempi MilpPackingAlgo-FillHardestFirstAlgo

I due grafici riportati propongono un confronto tra il modulo MilpPackingAlgo e il FillHardestFirstAlgo che evidenzia il fatto che, da una parte la percentuale di riempimento raggiunta è statisticamente leggermente maggiore per il primo, ma per quanto riguarda i tempi è netto il distacco del secondo modulo.

6.3 Pack&StockAlgo-FillHardestFirstAlgo

In questa sezione verrà proposta una sperimentazione che deciderà la strategia ottimale per il bilanciamento nell'utilizzo dei moduli Pack&StockAlgo e FillHardestFirstAlgo. Si tratta di scegliere quanti dei pezzi in input fornire

al modulo Pack&StockAlgo per la prima fase di riempimento. Proveremo con diverse percentuali e confronteremo i risultati.

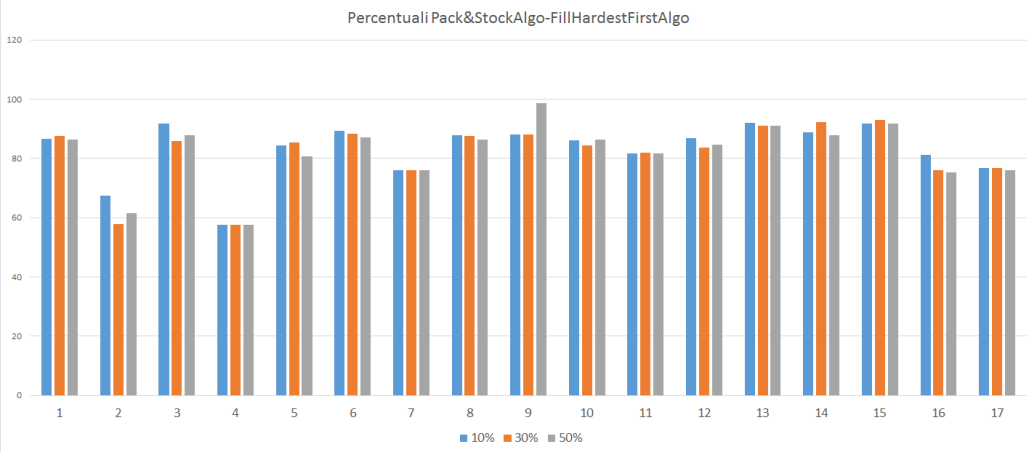


Figura 6.7: percentuali delle diverse varianti dell'algoritmo

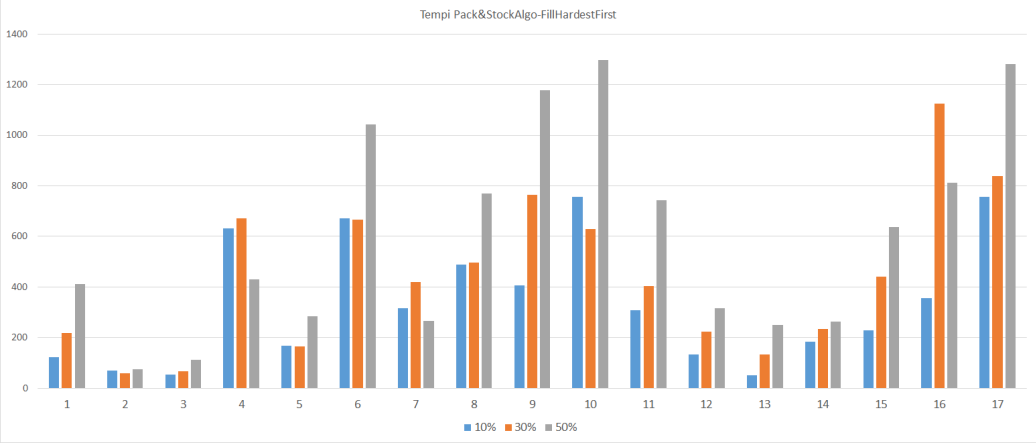


Figura 6.8: tempi delle diverse varianti dell'algoritmo

Nei grafici proposti si nota come la percentuale di pezzi fornita in input al modulo Pack&StockAlgo non cambi sostanzialmente la qualità della soluzione finale ma vada ad inficiare negativamente sulle tempistiche dell'algoritmo completo. Per questo nei successivi test verrà scelto di utilizzare Pack&StockAlgo sul 10% dei pezzi di area maggiore.

Tabella 6.3: Test Pack&StockAlgo sul primo 10% dei pezzi

nome test	pezzi input	% area pack	tempo pack	pezzi disposti	% area totale	tempo totale
Test1	89	78,362	3,732	20	86,515875	122,569
Test2	43	19,206	0,485	26	67,31825	68,776
Test3	51	53,77777778	0,531	32	91,76377778	54,231
Test4	54	34,88403571	5,318	54	57,48232143	630,78
Test5	56	55,03338776	1,235	39	84,34428571	166,499
Test6	113	63,1004	3,953	32	89,3777	670,17
Test7	50	27,76777778	0,482	50	75,99333333	314,855
Test8	66	55,4512	255,815	34	87,85912	488,881
Test9	97	62,5128	242,093	30	88,1134	405,008
Test12	100	53,27704	66,97	41	86,1904	755,315
Test13	100	49,60784	50,58	43	81,69872	308,389
Test15	112	62,80208	2,416	36	86,90352	133,679
Test16	108	67,74112	1,131	36	92,00328	50,838
Test17	112	66,74128	30,597	46	88,80824	182,351
Test18	117	61,62688	130,332	27	91,70256	229,162
Test19	90	49,2096	9,565	30	81,1704	354,798
Test20	90	41,48392	18,789	44	76,84992	755,228

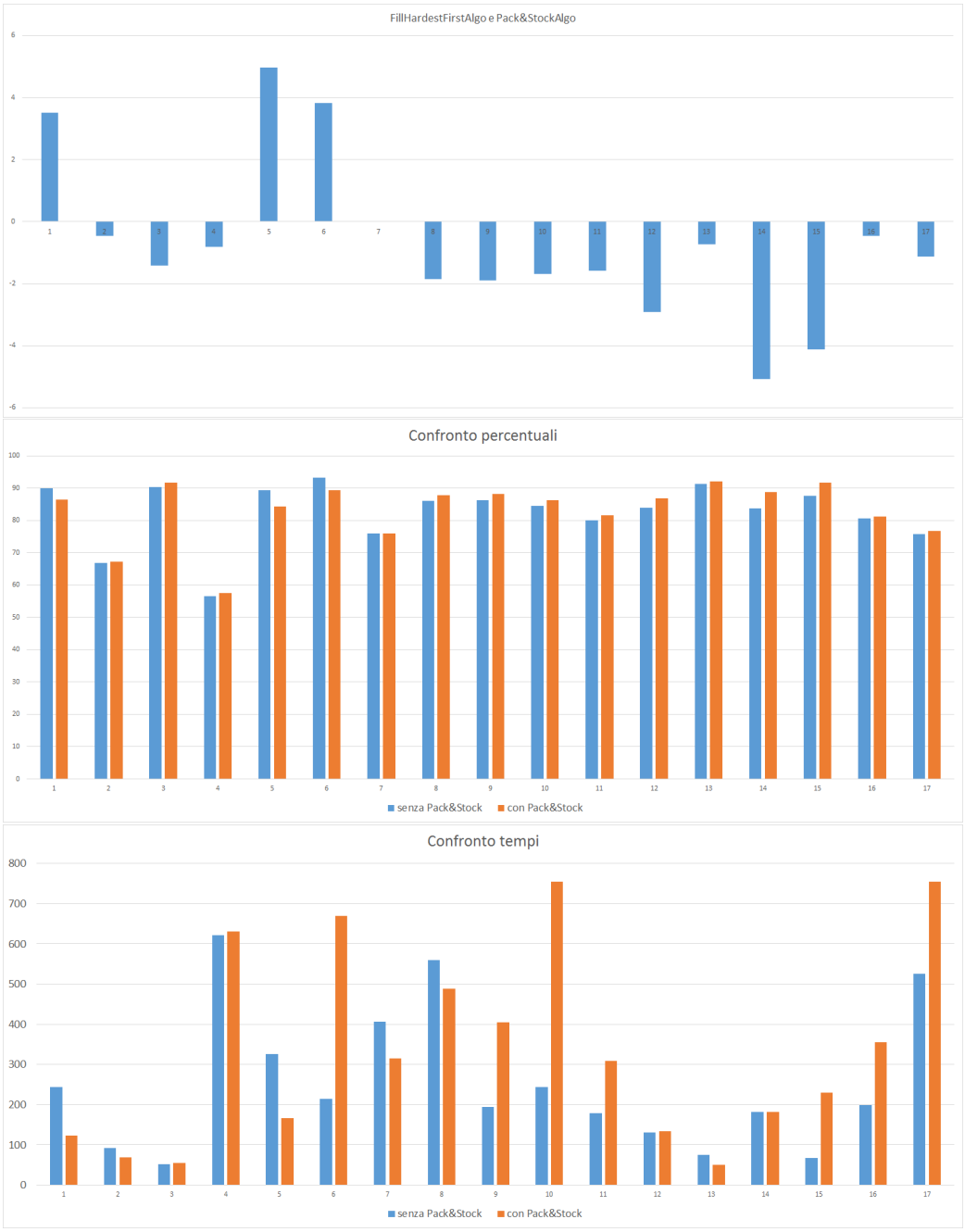
Tabella 6.4: Test Pack&StockAlgo sul primo 30% dei pezzi

nome test	pezzi input	% area pack	tempo pack	pezzi disposti	% area totale	tempo totale
Test1	89	75,62775	25,125	21	87,53775	217,951
Test2	43	38,412	2,783	22	57,94625	59,122
Test3	51	58,70822222	23,115	26	85,91722222	67,783
Test4	54	47,90314286	34,72	54	57,48232143	670,97
Test5	56	72,588	38,564	39	85,4384898	164,199
Test6	113	73,0667	65,937	32	88,4327	664,686
Test7	50	60,09222222	1,371	50	75,99333333	418,597
Test8	66	62,14336	306,03	35	87,48816	496,124
Test9	97	62,5128	607,306	30	88,1134	763,213
Test12	100	72,20504	138,424	45	84,472	629,564
Test13	100	46,60568	134,286	43	81,99472	404,176
Test15	112	61,988	19,198	43	83,7456	222,593
Test16	108	77,1728	30,7	34	91,06	133,133
Test17	112	72,83688	80,47	46	92,16304	234,719
Test18	117	74,64416	346,463	31	93,11328	440,515
Test19	90	52,9588	217,84	31	76,00448	1123,782
Test20	90	60,05696	138,167	44	76,8692	839,178

Tabella 6.5: Test Pack&StockAlgo sul primo 50% dei pezzi

nome test	pezzi input	% area pack	tempo pack	pezzi disposti	% area totale	tempo totale
Test1	89	77,962625	201,227	23	86,36325	412,381
Test2	43	38,412	8,918	24	61,5135	73,431
Test3	51	73,97166667	52,793	33	87,94511111	112,521
Test4	54	53,21621429	64,861	54	57,48232143	429,844
Test5	56	69,90702041	99,995	40	80,60983673	284,974
Test6	113	78,855	381,179	33	87,1298	1042,536
Test7	50	69,08333333	3,039	50	75,99333333	264,985
Test8	66	72,1256	500,205	36	86,3848	769,523
Test9	97	88,47146667	1003,876	31	98,60153333	1177,207
Test12	100	77,0528	727,621	47	86,31592	1296,632
Test13	100	59,45416	275,463	43	81,82248	743,438
Test15	112	73,41496	116,718	40	84,71792	316,276
Test16	108	83,95776	97,952	41	91,1444	250,727
Test17	112	67,97824	137,105	53	87,78488	263,41
Test18	117	77,72256	537,962	29	91,86672	636,516
Test19	90	64,66736	294,452	31	75,24088	810,866
Test20	90	56,83592	379,813	45	76,00992	1280,452

Vediamo ora gli effetti dell'utilizzo di Pack&StockAlgo confrontando la prima delle tre varianti proposta nel grafico precedente nell'algoritmo che usa solamente il modulo FillHardestFirstAlgo.



Il primo grafico evidenzia la differenza percentuale tra il riempimento senza e con l'utilizzo di Pack&StockAlgo. In effetti si nota un miglioramento

percentuale dello $0,69\%$ in media mentre il confronto dei tempi presenta dei risultati oscillanti. In alcune istanze l'utilizzo del modulo Pack&StockAlgo ha velocizzato il packing mentre in altri casi ha avuto l'effetto contrario.

Infine riporto qualche immagine di nesting per mostrare come lavorano i diversi moduli.

nome test	% area pack	tempo pack	% area totale	tempo totale
Test16	67,74112	1,366	92	169,752

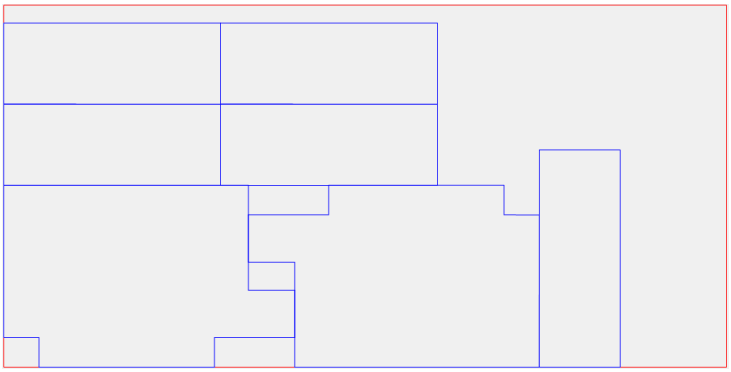


Figura 6.9: disposizione dei pezzi dopo l'utilizzo del modulo Pack&StockAlgo

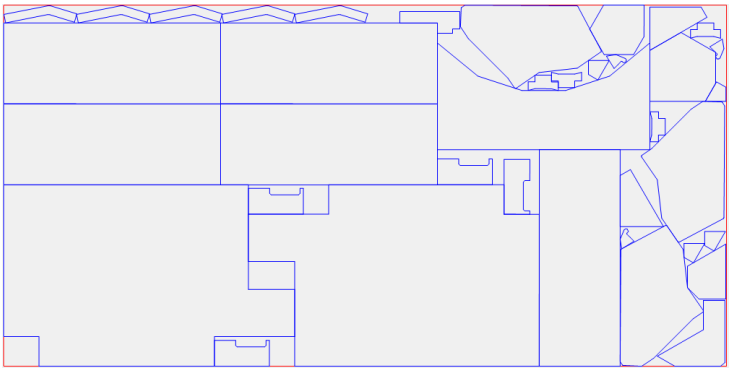


Figura 6.10: disposizione finale

nome test	% area pack	tempo pack	% area totale	tempo totale
Test8	55,4512	255,573	87,85	473,583

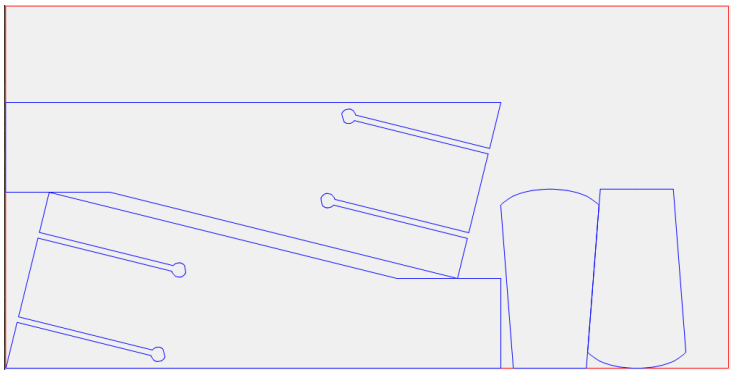


Figura 6.11: disposizione dei pezzi dopo l'utilizzo del modulo Pack&StockAlgo

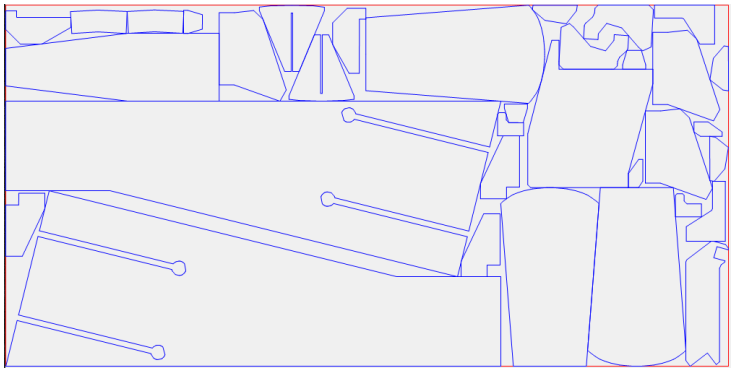


Figura 6.12: disposizione finale

Bibliografia

[AB01] N.R. Babu A.R. Babu. A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms. *Computer-Aided Design*, 33:879–891, 2001.

[JO93] J.S. Ferreira. J.F. Oliveira. Algorithms for nesting problems. *Applied Simulated Annealing, Lecture Notes in Economics and Mathematical Systems*, 396:255–273, 1993.

[MF09] I. Luzzi M. Fischetti. Mixed-integer programming models for nesting problems. *Journal of Heuristics*, 15(3):201–226, 2009.

[Moo02] A. Moore. The circle tree – a hierarchical structure for efficient storage, access and multi-scale representation of spatial data. *Proceedings of the 14th Annual Colloquium of the spatial Information Research Centre*, 2002.

[RAVn13] J.M. Tamarit R. Alvarez-Valdes n, A. Martinez. A branch & bound algorithm for cutting and packing irregularly shaped pieces. *Int. J. Production Economics*, 145:463–477, 2013.

[Roc14] P.F. Monteiro Rocha. Geometrical models and algorithms for irregular shapes placement problems. *Tesi di dottorato presso l'Università di Porto*, 2014.

[SS86] L.M. Braga S.A. Sagenreic. Optimal nesting of general plane figures: A monte carlo heuristical approach. *Computers & Graphics*, 10(3):229–284, 1986.