Università degli studi di
Milano-Bicocca

Decision Models

Final Project

Packing Santa's Sleigh: risoluzione tramite Algoritmo Genetico

Authors:
Bagnalasta Matteo - 833349 - m.bagnalasta@campus.unimib.it
Biondi Stefano - 839358 - s.biondi7@campus.unimib.it
Pierri Luca - 846597 - l.pierri1@campus.unimib.it

7 settembre 2020

Sommario

I problemi di confezionamento (packaging problems) sono stati studiati a fondo negli ultimi decenni, sia per la loro varietà di applicazione in aspetti pratici sia per interessi puramente teorici. L'oggetto di studio di questa relazione si basa su una particolare variante tridimensionale del problema di confezionamento. Sarà inizialmente presentato il dataset oggetto di analisi e la misura di valutazione seguita per portare a termine l'analisi stessa. Seguirà la descrizione del modello implementato per risolvere il problema, così come i risultati ottenuti al termine dei test.

1 Introduzione

L'obiettivo del progetto è risolvere il problema *Packing Santa's Sleigh* proposto da MathWorks, consultabile all'indirizzo [4]. Il problema consiste nel voler inserire i regali nella slitta di Babbo Natale seguendo un preciso ordine di arrivo e minimizzando lo spazio occupato in altezza. La combinazione di questi due vincoli rende particolarmente interessante il problema dal punto di vista pratico per le sue innumerevoli applicazioni. Allo stesso modo complica il ben noto e più generale problema di packing.
Per minimizzare la misura d'errore proposta nella competizione, si è deciso di implementare un algoritmo genetico che evolvesse le combinazioni di rotazioni dei pacchi inseriti. Sono stati inoltre impiegati diversi ordinamenti di inserimento dei pacchi per cercare di esplorare differenti configurazioni di minimo.

2 Dataset

Data l l'unità di misura, la base della slitta misura $1000 \,\mathrm{x}\, 1000\, l^2$, mentre l'altezza si estende infinitamente. I regali, di dimensioni randomiche, sono rappresentati da quattro proprietà:

- *PresentId*: l'ID del regalo;
- *Dimension1*: la dimensione x del regalo;
- *Dimension2*: la dimensione y del regalo;
- *Dimension3*: la dimensione z del regalo.

Il file csv, rappresentante la lista dei pacchetti, viene inizialmente caricato in memoria tramite *Pandas*, libreria Python di data processing. La libreria permette una veloce computazione delle statistiche descrittive del Dataframe.

	Dimension1	Dimension2	Dimension3
count	1000000.000000	1000000.000000	1000000.000000
mean	49.230199	49.260762	51.901871
std	61.875137	61.949176	61.981952
min	2.000000	2.000000	2.000000
25%	7.000000	7.000000	7.000000
50%	24.000000	24.000000	28.000000
75%	61.000000	61.000000	67.000000
max	250.000000	250.000000	250.000000

Figura 1: Statistiche descrittive dimensioni

Dalla tabella in Figura 1 si nota una chiara eterogeneità tra i pacchi:

- ognuna delle tre dimensioni varia da un minimo di $2\,l$ ad un massimo di $250\,l$;
- la media e la varianza si posizionano intorno a $50\,l$ e a $61\,l$ per ogni dimensione;

Per la manipolazione dei dati nell'algoritmo implementato, descritto nella sezione successiva, si è scelto di creare una classe che rappresentasse il singolo pacco (Pack). Si è quindi convertito il Dataframe, caricato precedentemente, in una lista di Pack:

```
> # Modello dati rappresentante il singolo pacchetto
class Pack:
    def __init__(self, x, y, z, pack_id):
        self.x = x
        self.y = y
        self.z = z
        self.pack_id = pack_id
```

3 Approccio Metodologico

3.1 Misura di Errore

I pacchi possono essere ruotati di multipli di 90 gradi in qualsiasi direzione. Dato N_p, numero di pacchi da inserire, il posizionamento dei pacchi è valutato utilizzando la seguente metrica:

$$M = 2 \max_i(z_i) + \theta(\Gamma) \quad con\, i = 1, ..., N_p \tag{1}$$

Dove z_i è la coordinata z dell'i-esimo regalo più l'offset in cui si trova, mentre in

$$\theta(\Gamma) = \sum_{i=1}^{N_p} |i - \Gamma_i| \tag{2}$$

Γ_i è l'ID del regalo nell'i-esima posizione. Quindi se $\Gamma_i = i \; \forall \, i = 1, ..., N_p$

$$\theta(\Gamma) = 0 \tag{3}$$

La misura da minimizzare è composta da due addendi $s_1 = 2 \max_i(z_i)$ e $s_2 = \theta(\Gamma)$. Analizzando quelli che sono i *worst cases* si può notare che gli addendi contribuiscono all'errore totale con ordini di grandezza significativamente differenti.
L'altezza sarà massima nel caso in cui i pacchi vengano impilati uno sull'altro. Considerando che i pacchi possono ruotare è possibile effettuare l'inserimento avendo come altezza del pacco i-esimo la dimensione minima dello stesso. Quindi

$$s_1 = 2 \cdot \sum_{i=1}^{N_p} \min_{j=1,2,3} Dimension(j) = 65.068.208 \simeq 6.5 \cdot 10^7 \tag{4}$$

L'addendo che verifica l'ordinamento, invece, sarà massimo nel caso in cui i pacchi vengano inseriti in ordine inverso a quello richiesto, in particolare si avrà

$$\begin{aligned} s_2 = \theta(\Gamma) = |\, 1 - 1.000.000 \,| + |\, 2 - 999.999 \,| + ... + |\, 500.001 - 500.000 \,| + \\ + |\, 500.000 - 500.001 \,| + ... + |\, 999.999 - 2 \,| + |\, 1.000.000 - 1 \,| = \end{aligned}$$

$$= 2 \cdot \sum_{i=1}^{500.000} (2\,i - 1) = 2 \cdot 500.000^2 = 5 \cdot 10^{11} \tag{5}$$

Risulta quindi prioritario focalizzarsi sulla minimizzazione della parte di ordinamento s_2.

3.2 Strutturazione del Problema

La priorità è quindi l'inserimento ordinato in modo da avere la componente $\theta(\Gamma)$ minore possibile.
Essendo i pacchi dei parallelepipedi, ogni singolo pacco possiede 3 superfici differenti. Il problema di riempimento del primo livello della slitta si riduce quindi a quello di trovare quale delle totali 3^{n_1} combinazioni minimizzi la misura di errore, con n_1 il numero di pacchi inseriti nel primo livello. Una volta concluso l'inserimento del primo strato, quello in cui tutti i pacchi hanno un lato a contatto con il fondo della slitta, il problema si complica perché la base su cui inserire i pacchi non sarà infatti più

piatta e l'inserimento dei pacchi di ogni nuovo livello dipenderà dalla configurazione del piano precedente.

Per diminuire il grado di complessità, mantenendo prioritario l'ordinamento, si è deciso di considerare il problema come un problema a layer. Ogni layer è un piano su cui verranno inseriti i pacchi mantenendo la seguente relazione sempre valida:

$$id_i > id_j \qquad \forall\, i = 1, .., n_k \;\forall\, j = 1, .., n_{k+1} \;\forall\, k = 1...N_{t-1} \tag{6}$$

con n_k e n_{k+1} rispettivamente il numero di pacchi inseriti al livello k e $k+1$ e N_t il numero di layer utilizzati per inserire tutti i pacchi. Il problema di bin packing 3D diventa quindi un problema di bin packing 2D iterato fino all'inserimento di tutti i pacchi. Considerando il singolo layer infatti il problema si riduce nel trovare quale delle 3 superfici rettangolari di ogni pacco mettere a contatto con la base del layer stesso e in che modo posizionare tutte queste superficie all'interno della base.

3.3 Scelta ed Implementazione del Modello

Il problema da risolvere per ogni layer, fino ad esaurimento dei pacchi, è quindi trovare quale delle 3^{n_i} combinazioni possibili di rotazioni dei pacchi minimizzi la misura d'errore, con n_i numero di pacchi inseriti nell'i-esimo livello. Fissato il livello, quindi, agirà un algoritmo genetico che è descritto nella 3.3.2. Questo evolverà le configurazioni degli n_i pacchetti, in particolare deciderà per ognuno di essi quale delle tre misure è l'altezza (rotazione verticale) e lascerà il compito delle rotazioni orizzontali ad un algoritmo di inserimento bidimensionale, il MaxRects che è descritto nella 3.3.1. La funzione di fitness scelta per l'evoluzione delle popolazioni in ogni layer i è data dalla misura di errore (1) normalizzata per il numero di pacchi n_i.

$$FitnessFunction_i = M_i/n_i \tag{7}$$

La scelta di normalizzare l'errore per il numero di pacchi è dovuta al fatto che, per ogni layer, si vuole trovare un ottimo tra il minimo dell'errore M_i e il massimo n_i. Questo perché si vogliono evitare i casi limite in cui venga favorita una configurazione che abbia basso errore a causa di un basso numero di pacchi inseriti. In questo caso si avrebbe un vantaggio sul singolo layer a discapito dell'obiettivo dello studio: minimizzare l'errore totale

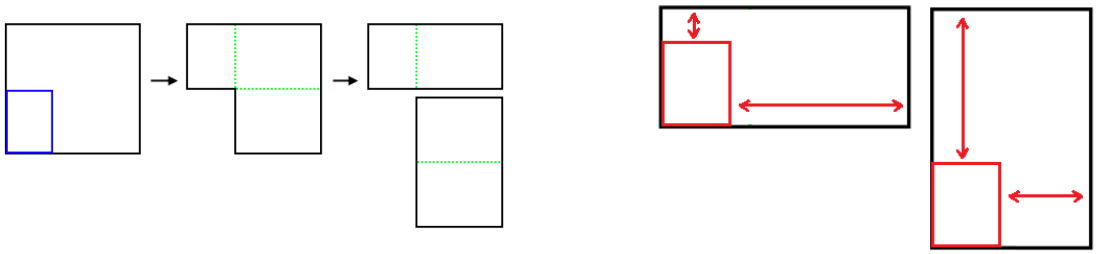
$$M = \sum_{i=1}^{N_t} M_i \tag{8}$$

ricordando che N_t è il numero di layer utilizzati per l'inserimento di tutti i pacchi.

3.3.1 Posizionamento dei rettangoli nel singolo layer

Per la soluzione del problema di posizionamento 2D la letteratura è vasta. In particolare si è utilizzato il metodo *MaxRectsBssf (Maximum Rectangle - Best short side fit)*

descritto in [3]. Questo algoritmo inserisce il primo rettangolo nell'angolo in basso a sinistra e memorizza tutti i rettangoli liberi che rappresentano l'area rimanente del layer, come mostrato in figura 2a.



(a) *Rettangoli liberi dopo il primo inserimento*

(b) *Scelta del rettangolo libero per l'inseriemtno del secondo*

Figura 2: Inserimento rettangoli *MaxRectsBssf*

Quindi inserirà il secondo rettangolo in basso a sinistra nei rettangoli liberi e, dopo aver calcolato le distanze libere rimanenti, andrà a posizionarlo dove questa è minore.

Per la sua implementazione, è stata selezionata la libreria Python *RecktPack* [2] che raccoglie una serie di euristiche, tra cui MaxRectsBssf, per la risoluzione del problema di packing 2D.
Per rispettare il vincolo (6) si è deciso di modificare la libreria in modo tale da interrompere l'inserimento dei pacchi nel momento in cui l'algoritmo non avesse spazio libero sufficiente per inserire il pacco successivo. Questa metodologia di interruzione evita che si modifichi l'ordine di inserimento dei pacchi. L'algoritmo originale, una volta trovato il primo pacco non inseribile, provava ad inserire in sequenza i pacchi successivi dando priorità al riempimento piuttosto che all'ordine.

3.3.2 Algoritmo genetico per rotazione dei pacchi

In accordo con quanto detto finora è stato implementato un *Algoritmo Genetico* in cui un *cromosoma* è definito da una sequenza di pacchi da ruotare e dall'errore dato dall'inserimento sequenziale delle superfici di base nel riempimento del layer.

```
class Chromosome:
 def __init__(self, packs, error=0):
     self.packs = packs
     self.error = error
```

I parametri di controllo dell'algoritmo sono:

- numero di **generazioni** (o iterazioni) g;

- numero di **cromosomi della popolazione iniziale** m;
- numero di **cromosomi elitari** e che passano direttamente alla generazione successiva;
- probabilità di **selezione** $pSel$;
- probabilità di **crossover** $pCross$;
- probabilità di **mutazione** $pMut$.

L'algoritmo genetico implementato si struttura nel seguente modo:

- **Fase Preliminare**. I rettangoli, superficie dei pacchi a contatto con la base del layer, sono ordinati per id decrescente. Viene effettuato un primo inserimento dei rettangoli seguendo l'ordine fino a raggiungere la condizione di interruzione descritta nella sezione 3.3.1. Questa fase preliminare serve a stimare l'ordine di grandezza del numero n di rettangoli che si riusciranno ad inserire nel layer. In ogni cromosoma si inseriscono $3 \cdot n$ pacchi perché, come verrà a breve descritto, modificando tramite algoritmo genetico il loro orientamento si modificheranno le superfici bidimensionali (rettangoli) che dovranno essere inserite nel layer e, conseguentemente, cambierà il numero n di pacchi inseriti per ogni cromosoma. Si è deciso di non inserire in ogni cromosoma la lista completa dei pacchi perché si ha un significativo risparmio dal punto di vista computazionale.
- **Fase di Inizializzazione**. Per ogni generazione, sono presenti 2 cromosomi **custom** e $m - 2$ cromosomi **random**. Il primo cromosoma custom è rappresentato dai rettangoli (xy) dei pacchi di arrivo senza modifiche; il secondo dai rettangoli $(xy)'$ degli stessi pacchi ruotati in modo che l'altezza z' sia la dimensione minore delle tre. I cromosomi random invece sono rappresentati dai rettangoli $(xy)''$ della medesima sequenza di pacchi ruotati in modo casuale per multipli di 90 gradi (in breve, scambiando randomicamente le coordinate del pacco).
- **Fase di Inserimento nel Layer**. I pacchi di ogni cromosoma, dopo essere stati ordinati per id decrescente, vengono inseriti nel layer con il metodo MaxRectsBssf descritto in 3.3.1. Per ricavare la fitness, M/n, di tale riempimento, si considerano il numero di rettangoli inseriti n e il valore della misura d'errore M, ottenuto ricostruendo il cubo su ogni rettangolo inserito grazie alla sua altezza.
- **Fase di Riproduzione**. Gli e cromosomi, con i migliori valori di fitness, vengono rimossi dalla popolazione m per essere tramandati alla popolazione successiva. La fase di selezione si affida al cosiddetto metodo a *torneo binario*, ovvero, si scelgono due individui dalla popolazione di cromosomi restanti $m-e$,

l'individuo con fitness migliore viene scelto con probabilità $pSel$, mentre l'altro viene scelto con probabilità $1 - pSel$. Si itera il processo fintanto che non si sono scelti $m - e$ parenti.
Scegliendo casualmente coppie di parenti, con probabilità $pCross$ viene effettuato il crossover e si generano coppie di figli, con probabilità $1 - pCross$ non si effettua nessun tipo di crossover e i due parenti diventano i figli. La fase di crossover è mostrata in figura 3. Infine, con probabilità $pMut$, viene effettuata una mutazione su un cromosoma figlio: si prendono casualmente due pacchi e si mischiano casualmente le dimensioni. La nuova popolazione di cromosomi su cui rifare la fase di inserimento layer sarà così formata dagli e elitari più gli $m - e$ figli.

- Il processo viene ripetuto per un numero di iterazioni g. Dell'ultima popolazione viene selezionato il cromosoma con *fitness* minore.

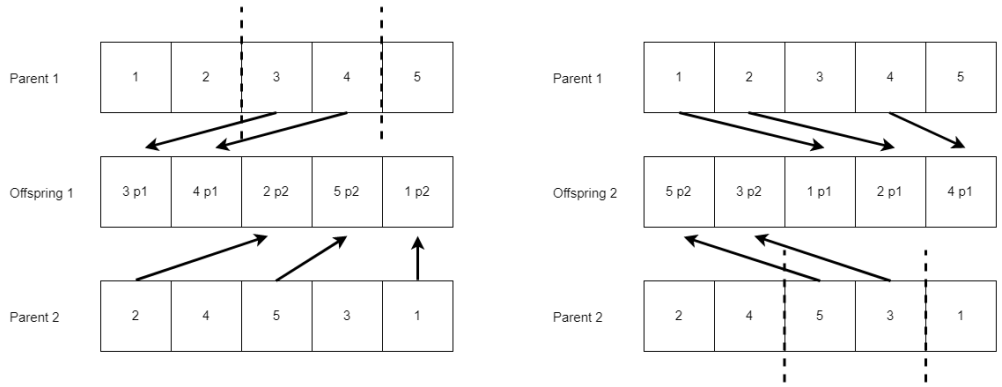


Figura 3: Crossover algoritmo genetico

4 Risultati e Valutazione

L'algoritmo genetico è stato implementato con i parametri di controllo descritti nella Tabella 1

Iterazioni	m	e	$pSel$	$pCross$	$pMut$
3	20	5	0.9	0.6	0.3

Tabella 1: Parametri algoritmo

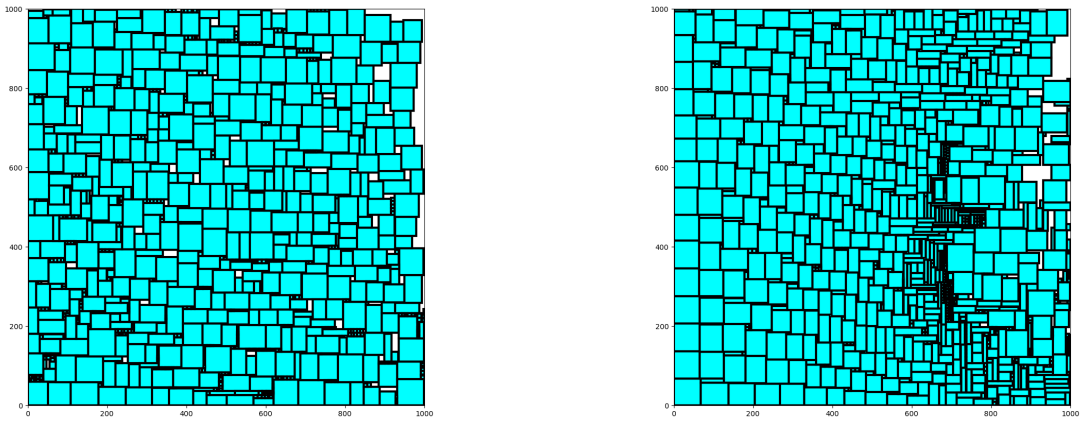
A causa della limitata capacità hardware a nostra disposizione si è testato l'algoritmo su soli 10 layer consecutivi, non procedendo fino all'esaurimento dei pacchi.

4.1 Inserimento senza Algoritmo Genetico

Utilizzando unicamente l'algoritmo di inserimento, senza utilizzare l'algoritmo genetico e ordinando per id decrescente si ha un numero medio di inserimento di pacchi per layer di 1329.4, un errore (1) medio per layer dell'ordine di $576 \cdot 10^3$ ed un tempo di computazione medio per layer di $33.8s$.

4.2 Inserimento con Algoritmo Genetico

Utilizzando l'algoritmo genetico per la rotazione dei pacchi prima dell'inserimento si ha un numero medio di pacchi inseriti per layer di 867.7, un errore (1) finale medio per layer di $244 \cdot 10^3$ ed un tempo di computazione medio per layer di $2100.7s$. Il risultato del riempimento del primo layer è riportato graficamente in figura 4a.



(a) *Ordinamento per Id*

(b) *Ordinamento per Area*

Figura 4: Inserimento dei pacchi 2D

Per confrontare i risultati dell'inserimento con e senza algoritmo genetico si è considerato il rapporto ρ tra errore medio e numero medio di pacchi inseriti per layer:

$$\rho_{genetico} = 281.81 \quad \rho_{nongenetico} = 433.53 \tag{9}$$

5 Discussione

I risultati dell'algoritmo genetico sono significativamente migliori. Tuttavia al procedere delle 3 iterazioni la best fitness è rimasta costante. Questa deriva dal secondo

cromosoma custom della *fase di inizializzazione* della 3.3.2 in cui si ordinano a priori le dimensioni per avere come dimensione minima l'altezza. Quindi l'algoritmo genetico non ha alcun ruolo nel miglioramento dei risultati. Per cercare di sfruttare le potenzialità dell'algoritmo genetico si è deciso quindi di aumentare la probabilità di crossover $pCross = 0.75$, la probabilità di mutazione $pMut = 0.4$, la popolazione iniziale $m = 30$ e le iterazioni a 5. Queste modifiche, tuttavia, non hanno permesso di trovare una soluzione di minimo alternativa a quella subottimale già trovata.

Alla luce di queste considerazioni si è proceduto all'implementazione di un diverso tipo ordinamento che non violasse l'equazione (6) e permettesse di inserire più pacchi per ogni layer. In particolare, nella *fase di inserimento nel layer* della sezione 3.3.2, oltre all'ordinamento per id, i primi $0.8 \cdot n$ rettangoli vengono ulteriormente ordinati per area decrescente. Per mantenere valida l'equazione (6) si suppone che la frazione ordinata per area dei rettangoli venga sempre inserita nel layer, in caso contrario, per il cromosoma in questione, si effettuerà l'inserimento con ordinamento unicamente per id. Con l'implementazione dell'ordinamento per area si cerca di ottimizzare l'utilizzo della superficie disponibile da parte dell'algoritmo di inserimento bidimensionale: inizialmente, quando si ha molta area a disposizione l'algoritmo inserisce i rettangoli con superficie maggiore, passando poi ad inserire rettangoli più piccoli man mano che la superficie disponibile al bordo diminuisce.
Il risultato di riempimento del primo layer con questa implementazione è riportato graficamente in figura 4b.
Valutando i risultati, non si osserva un miglioramento della precedente best fitness generata dal cromosoma custom con altezze come dimensione minima. In particolare questo cromosoma è l'unico in cui si ha un numero di rettangoli inseriti sempre minore a $0.8 \cdot n$, quindi l'ordinamento avviene per id e non per area. Questo spiega perché i risultati siano identici all'implementazione con l'ordinamento originale.

5.1 Scelta del modello più efficace per la risoluzione finale

Alla luce del fatto che l'algoritmo genetico, per i primi 10 layer, non ha alcun ruolo nel miglioramento del risultato già ottenuto in fase di inizializzazione, si è deciso di computare il risultato dell'inserimento completo dei pacchi utilizzando solo il metodo euristico *MaxRectBssf*, dopo aver ordinato i pacchi in modo tale da avere come massima dimensione l'altezza. In tal modo in sole 10 ore di computazione l'algoritmo ha inserito la totalità dei pacchi con un errore totale :

$$M_{tot} = 110163524 \simeq 1 \cdot 10^8 \tag{10}$$

6 Conclusioni

In questa relazione è stata affrontata una particolare versione del 3D Bin Packing Problem. Nello specifico è stato implementato un algoritmo genetico che lavorasse sulle rotazioni tridimensionali dei pacchi. Inoltre sono state impiegate diverse strategie di inserimento nei layer: ordinamenti per id e per area. Dalle analisi emerge che la combinazione di rotazione dei pacchi migliore è quella che utilizza come altezza la dimensione del pacco minima. Questo è confermato da entrambi gli approcci di ordinamento, quindi l'ottimo ottenuto deriva da una condizione imposta a priori e non dall'utilizzo dell'algoritmo genetico in sé.
Alla luce di ciò, utilizzando il solo metodo euristico *MaxRectBssf*, dopo aver ordinato i pacchi in modo tale da avere come massima dimensione l'altezza, si ottiene un errore totale pari a:

$$M_{tot} = 110163524 \simeq 1 \cdot 10^8 \tag{11}$$

L'errore totale è quindi meno di 2 volte l'errore commesso nel *worst case* delle altezze, quello in cui i pacchi vengono impilati uno sull'altro, portando a 0 la componente d'errore dovuta all'ordinamento e avendo massima la componente dovuta all'altezza.
Un possibile miglioramento dei risultati può derivare dall'utilizzo dell'algoritmo genetico con ordinamento per area con un tuning della percentuale dei rettangoli ordinati per area rispetto al totale, attualmente $0.8 \cdot n$. Una seconda possibilità di miglioramento dei risultati, grazie all'uso dell'algoritmo genetico, può derivare dal tuning dei parametri così da permettere una maggiore esplorazione delle possibili soluzioni di ottimo.

Considerando i risultati della competizione, che spaziano tra $2 \cdot 10^6$ e $5 \cdot 10^7$, e considerando le indicazioni precedenti, non si attende un miglioramento soddisfacente per la risoluzione della competizione. Il ragionamento per layer è risultato essere un approccio troppo grezzo. Le future analisi dovrebbero essere volte ad implementare degli algoritmi ad hoc che tengano conto dello skyline generato dall'inserimento dei pacchi.

Riferimenti bibliografici

[1] *BoxPacking*. Ihor Kovalyshyn, in: `https://github.com/delta1epsilon/BoxPacking`, 2019;

[2] *RectPack*. in: `https://github.com/secnot/rectpack`, 2017;

[3] *A Thousand Ways to Pack the Bin - A Practical Approach to Two-Dimensional Rectangle Bin Packing*, Jukka Jylang, 2010;

[4] *Packing Santa's Sleigh*. MathWorks, in: `https://www.kaggle.com/c/packing-santas-sleigh`, 2014.