POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Title

Tesi di Laurea Magistrale in
Xxxxxxx Engineering - Ingegneria Xxxxxxx

Author: **Name Surname**

Student ID: 000000
Advisor: Prof. Name Surname
Co-advisors: Name Surname, Name Surname
Academic Year: 20XX-XX

Abstract

Here goes the Abstract in English of your thesis followed by a list of keywords. The Abstract is a concise summary of the content of the thesis (single page of text) and a guide to the most important contributions included in your thesis. The Abstract is the very last thing you write. It should be a self-contained text and should be clear to someone who hasn't (yet) read the whole manuscript. The Abstract should contain the answers to the main scientific questions that have been addressed in your thesis. It needs to summarize the adopted motivations and the adopted methodological approach as well as the findings of your work and their relevance and impact. The Abstract is the part appearing in the record of your thesis inside POLITesi, the Digital Archive of PhD and Master Theses (Laurea Magistrale) of Politecnico di Milano. The Abstract will be followed by a list of four to six keywords. Keywords are a tool to help indexers and search engines to find relevant documents. To be relevant and effective, keywords must be chosen carefully. They should represent the content of your work and be specific to your field or sub-field. Keywords may be a single word or two to four words.

Keywords: here, the keywords, of your thesis

Abstract in lingua italiana

Qui va l'Abstract in lingua italiana della tesi seguito dalla lista di parole chiave.

Parole chiave: qui, vanno, le parole chiave, della tesi

Contents

Introduction

Intro

Case study

Overview

1 | Literature review

2 | Problem description and mathematical formulation

2.1. 3D Bin Packing Problem

2.2. Support

2.3. MILP Formulation

Conceptual model

A conceptual model of the problem we are trying to solve would be:

minimize	unused volume in used bins
subject to	all items assigned to one and only one bin
	all items within the bin dimensions
	no overlaps between items in the same bin
	all items with support

We can now provide the formal definition of the 3DBPP by formulating a mixed integer linear programming problem model.

Formal model

Let us consider now the standard 3DBPP problem definition and define a formal model which we'll expand with additional constraints in the following sections.

We start by defining the known sets and parameters of the problem.

Sets

$$I = \{1, \dots, n\}: \quad \text{set of items}$$
$$B = \{1, \dots, m\}: \quad \text{set of bins}$$

Parameters

$$
\begin{aligned}
W \times D \times H & \quad \text{width} \times \text{depth} \times \text{height of a bin} \\
V & \quad \text{bin volume} \\
w_i \times d_i \times h_i & \quad \text{width} \times \text{depth} \times \text{height of item } i \qquad \forall i \in I \qquad (2.1)
\end{aligned}
$$

Variables We can now introduce the following sets of integer variables

$$(x_i, y_i, z_i) \quad \text{bottom front left corner of an item} \qquad \forall i \in I \qquad (2.2)$$

$$(x'_i, y'_i) \quad \text{back right corner of an item} \qquad \forall i \in I \qquad (2.3)$$

$$r_i \quad \begin{cases} 1, \text{if item } i \text{ is rotated } 90° \text{ over its z-axis} \\ 0, \text{otherwise} \end{cases} \qquad \forall i \in I \qquad (2.4)$$

$$u_{ib} \quad \begin{cases} 1, \text{if item } i \text{ is placed in bin } b \\ 0, \text{otherwise} \end{cases} \qquad \forall i \in I, \forall b \in B$$

$$x^p_{ij} \quad \begin{cases} 1, \text{if } x_i \leq x'_j \\ 0, \text{otherwise} \end{cases} \qquad \forall i, j \in I$$

$$y^p_{ij} \quad \begin{cases} 1, \text{if } y_i \leq y'_j \\ 0, \text{otherwise} \end{cases} \qquad \forall i, j \in I$$

$$z^p_{ij} \quad \begin{cases} 1, \text{if } z_i \leq z_j + h_j \\ 0, \text{otherwise} \end{cases} \qquad \forall i, j \in I$$

$$z^{\max}_b \quad \text{maximum height of bin } b \qquad \forall b \in B$$

Given a coordinate system, each item i can be rappresented univocally in 3D space by eqs. (2.1) to (2.4) as seen in figure 2.1

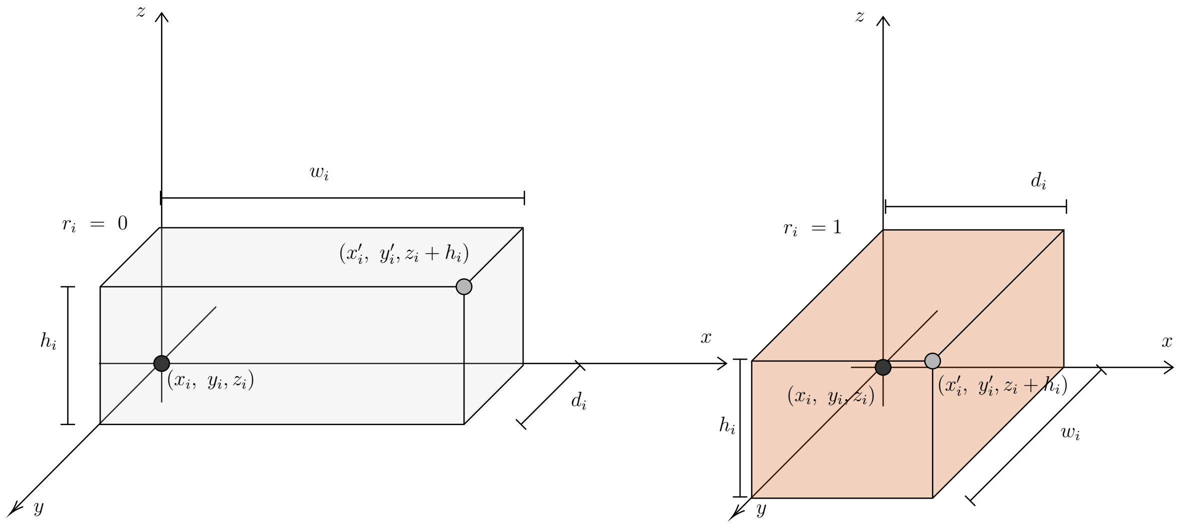

Figure 2.1: Coordinate system rappresentation for a generic item i given its rotation r_i

3 | Solution algorithms

In this chapter we describe a solution to the 3D bin packing problem with static stability. A solution candidate to the problem can be found by conducting a search over the graph of possible packings or states given an approriate rappresentation which is described in section 3.1. Since an exaustive search isn't feasible, an heuristic search is conducted by combining a beam search algorithm described in section 3.2 and constructive heuristic described in section 3.3. The proposed algorithm takes in input an initial feasible state (as defined in section 3.1.1) usually rappresented by the empty state (3.1.2) and outputs the best scoring state based on an ordering function defined in section 3.2.1.

3.1. State

States or packings are partial solutions to the 3DBPP. Given the formal definition of the problem (2.3) a few new definitions are introduced to facilitate the algorithm's definition.

Definition 3.1.1 (Unpacked item)**.** Given an item $i \in I$ we define it as unpacked **iff**

$$\sum_{b \in B} u_{ib} = 0$$

A state s can then be defined as follows

- U: the set of unpacked items
- B: the set of bins
- $(s_1, s_2, \ldots, s_b)$: the set of supporting structures for each bin $b \in B$

Observation 3.1.1. *Given two states s and s' we can have that $|s.B| \neq |s'.B|$ since the number of bins is also a variable in the proposed heuristic*

Each bin b has additional data used to facilitate the execution of the algorithm that is contained in s_b. Let us introduce the concept of placement inside a bin:

Definition 3.1.2 (Packed item)**.** Given a state s and a bin $b \in s.B$, we say that item $i \in I$ is packed in b **iff**

$$\begin{cases} u_{ib} = 1, \\ \sum_{j \in s.B, j \neq b} u_{ij} = 0 \end{cases}$$

3.1.1. Feasibility

Observation 3.1.2. *We can always define the empty state s_e where*

$$\begin{cases} s_e.U = I \\ s_e.B = \emptyset \end{cases}$$

and it is always feasible

3.2. Beam Search

Beam Search (BS) is an heuristic graph search algorithm designed for systems with limited memory where expanding every possible node is unfeasible. The idea behind BS is to conduct a iterative truncated breadth-first search where, at each iteration, expanded nodes are ranked based on an heuristic and only the best ones are further explored. To perform BS one must define the node structure, an expansion function to generate new nodes from an existing one, an evaluation function to compare nodes between eachother and a function to determine if a node is a solution to the problem.

Let s_i be a node in the graph of possible solutions of the 3DBPP, s_i can be seen as an instance of the problem where a sequence of placements has taken place. An expansion of a node s_i generates a new node s_j where a placement has occured for a given set of items. Since evaluating possible expansions can be computationally easier than computing new node data structures, a *Commit* function is defined which applies a pre-computed expansion by updating the supporting data structures in its node.

Given S_{init} the set of initial nodes to start from and k the number of best nodes to expand at each iteration, the described procedure is rappresented by algorithm 1.

Algorithm 1: Beam search

input : S_{init}, k
output: S_{best}
$S \leftarrow S_{init}$
$S_{final} \leftarrow \emptyset$
repeat
  $S_{new} \leftarrow Expand(S)$ (Algorithm 2)
  $S_{final} \leftarrow S_{final} \cup \{s_i \in S_{new} : IsFinal(s_i)\}$
  $S_{new} \leftarrow S_{new} \setminus S_{final}$
  $S_{new} \leftarrow Sort(S_{new})$
  $S \leftarrow \{\forall Commit(s_i) : s_i \in S_{new} \land i \in \mathbb{Z}^+ \land i \leq k\}$
until $S \neq \emptyset$
$S_{final} \leftarrow Sort(S_{final})$
return $s_0 \in S_{final}$

The *Expand* function computes new nodes which rappresent possible placements that can be made starting from a given packing. Each node contains a number of supporting data structures that are updated across iterations by the *Commit* function.

Let S be the set of nodes that need to be expanded, each node s is rappresented by a structure which contains

- *bins*: the set of open bins
- *unpacked*: the set of items that aren't assigned to any bin
- s_b: a substructure which cointains informations about a bin b

Let $GroupByHeight(I)$ be a function which operates on a set of items and outputs a set of tuples (t, I) where t is the family of the set I of items. A new set of nodes can be computed by using an underlying 3DSPP heuristic which evaluates the best move for each family of items for each currently opened bin. The described procedure is detailed in algorithm 2

Algorithm 2: Expand

```
input  : S
output: S_new
forall s ∈ S do
    S_new ← ∅
    I_h ← GroupByHeight(s.unpacked)
    placed ← false
    forall (h, I) ∈ I_h do
        forall b ∈ bins do
            placement ← SPBestInsertion(s_b, I) (Algorithm 3)
            if placement ≠ ∅ then
                placed ← true
                S_new ← S_new ∪ Next(s, placement)
            end
        end
    end
    if placed = false then
        S_new ← S_new ∪ OpenNewBin(s)
    end
end
return S_new
```

3.2.1. Scoring States

In order to sort nodes, a scoring function needs to be defined over the nodes. To allow the BS to explore better solutions the scoring function can't be as flat as the objective function defined in the mathematical formulation of the problem.

3.3. Support Planes

We introduce Support Planes (SP) which is an heuristic introduced in this thesis based on an underlying 2DBPP heuristic which is used to evaluate feasible expansions of a given node in the BS. The proposed heuristic ensures that the constraint of support isn't violated. The idea at the base of SP is to build a solution to the 3DSPP by filling 2D planes called support planes.

Each support plane can be characterized by the triple $S_z = (z, I_{support}, I_{upper})$ where

- z: the height of the plane
- $I_{support}$: the set of the items that can offer support to items placed on the plane
- I_{upper}: the set of items that will be obstacles to potential new items placed on the plane

Let s_b be a data structure containing

- $planes$: the set of triples S_z of support planes to evaluate, ordered in ascending z order
- $aabb$: the AABB Tree of the items placed in the evaluated bin
- (W_b, D_b, H_b): the dimensions of the bin

Let $coords$ be the set of possible coordinate changes which allow for the problem to evaluate placements starting from different corners of the bin.

Given a function $IsFeasible(i, bin, I_{support}, I_{upper}, aabb)$ which evaluates if a packing of item i in bin bin is feasible, and the function $ComparePacking(p, p')$ which defines a ranking over placements in the same plane, the SP algorithm can be written as algorithm 3.

Algorithm 3: SP Best Insertion

```
input  : s_b, I
output: placement
placement ← ∅
forall S_z ∈ planes do
    I_p ← I \ {i ∈ I : z + i.h > H_b}
    forall change ∈ coords do
        I'_upper ← CoordinateChange(change, I_upper)
        I'_p ← CoordinateChange(change, I_p)
        P' ← SPPackPlane(W_b, D_b, I'_upper, I'_p) (Algorithm 4)
        P ← CoordinateChange(change, P')
        P ← {i ∈ P : IsFeasible(i, bin, I_support, I_upper, aabb)}
        if ComparePacking(placement, P) then
            placement ← P
        end
    end
    if placement ≠ ∅ then
        return placement
    end
end
return placement
```

To evaluate a packing on a plane an heuristic to solve the 2DBPP is used with the introduction of fixed placements which rappresent items on other planes that will be obstacles in the current one.

Given the dimensions of the 2D bin (W_b, D_b), the set of obstacles I_o and the set of items to pack I_p a new placement can be computed following algorithm 4

Algorithm 4: SP Pack Plane

input : W_b, D_b, I_o, I_p

output: P

$P \leftarrow \emptyset$

$2dPacking \leftarrow \emptyset$

foreach $i \in I_o$ **do**

```
//Initialize the 2D bin packing instance with each obstable already placed
```

  $2DPlaceRect(2dPacking, i)$

end

repeat

```
//Pack untill full
```

  $p \leftarrow 2DPackRect(2dPacking, W_b, D_b, i)$

  $P \leftarrow P \cup \{p\}$

until $p \neq \emptyset$

return P

Once the k best nodes are selected the placements evaluated for each node are applied and the $Commit$ function updates every datastructure in S, including the ones used by SP. Given the instance that generated one of the placements selected and p the current set of support planes, z_{min} the minimum z coordinate for which a placement was made in the related bin starting from the current state, I the set of items placed, U the set of items unpacked. Since placements are evaluated in order starting from the lower z possible, if no placement was made in an open support plane with z lower than z_{min}, the plane can be pruned to avoid further evaluations. The algorithm which updates the structures for a given SP instance is rappresent by algorithm 5.

Algorithm 5: SP Apply and Filter

```
input  : s_b, I, z, z_min, t
output: s'_b
//Filter bad planes
P' ← planes \ {S_z ∈ planes : z ≤ z_min}
//Apply insertion
B ← placed ∪ I
U ← unpacked \ I
T ← aabb
forall i ∈ I do
    T ← InsertAABB(i, T) //If balanced O(log(n))
    generate ← true
    forall S'_z ∈ P' do
        //Based on the distance from the top of the item
        dz ← S'_z.z − i.z_max
        if 0 ≤ dz ≤ t then
            generate ← false
            S'_z.I_support ← S'_z.I_support ∪ i
        end
        else if dz < 0 then
            S'_z.I_upper ← S'_z.I_upper ∪ i
        end
    end
    if generate then
        P' ← P' ∪ (i.z_max, {i}, ∅)
    end
end
return Update(s_b, P', B, U, T)
```

3.3.1. Scoring Insertions

3.4. Max Rects

3.4.1. AABB Tree

In order to check the feasibility of a given insertion, a way of checking for intersections is needed. Since every box in a solution is axis aligned and defined by a static bounding box an Axis Aligned Bounding Box Tree (AABB Tree) is constructed and updated throughout the various nodes of the search. AABB Trees are accelleration structures which allow the computation of intersections given a bounding box with a time complexity of $O(\log n)$ where n is the number of items placed.

4 | Computational experiments

5 | Conclusions and future developments

A final chapter containing the main conclusions of your research/study and possible future developments of your work have to be inserted in this chapter. [1]

Bibliography

[1] D. E. Knuth. Computer programming as an art. *Commun. ACM*, pages 667–673, 1974.

A | Appendix A

If you need to include an appendix to support the research in your thesis, you can place it at the end of the manuscript. An appendix contains supplementary material (figures, tables, data, codes, mathematical proofs, surveys, . . .) which supplement the main results contained in the previous chapters.

B | Appendix B

It may be necessary to include another appendix to better organize the presentation of supplementary material.

List of Figures

List of Tables

List of Symbols

Variable	Description	SI unit
$\boldsymbol{u}$	solid displacement	m
$\boldsymbol{u}_f$	fluid displacement	m

Acknowledgements

Here you might want to acknowledge someone.