



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Title

TESI DI LAUREA MAGISTRALE IN
XXXXXXX ENGINEERING - INGEGNERIA XXXXXXXX

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

| | |
|---|------------|
| Abstract | i |
| Abstract in lingua italiana | iii |
| Contents | v |
| | |
| Introduction | 1 |
| Literature review | 3 |
| Problem description and mathematical formulation | 5 |
| Solution algorithms | 7 |
| 0.1 Beam Search | 7 |
| 0.1.1 Scoring States | 9 |
| 0.2 Support Plane | 9 |
| 0.2.1 Scoring Insertions | 13 |
| 0.3 Max Rects | 13 |
| 0.4 Feasibility | 13 |
| 0.4.1 AABB Tree | 14 |
| Computational experiments | 15 |
| Conclusions and future developments | 17 |
| | |
| Bibliography | 19 |
| | |
| A Appendix A | 21 |

| | |
|-------------------------|-----------|
| B Appendix B | 23 |
| List of Figures | 25 |
| List of Tables | 27 |
| List of Symbols | 29 |
| Acknowledgements | 31 |

Introduction

Literature review

Problem description and mathematical formulation

Solution algorithms

0.1. 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)
 $S_{final} \leftarrow S_{final} \cup \{s_i \in S_{new} : IsFinal(s_i)\}$
 $S_{new} \leftarrow S_{new} \setminus S_{final}$
 $S_{new} \leftarrow Sort(S_{new})$
 $S \leftarrow \{\forall Commit(s_i) : s_i \in S_{new} \wedge i \in \mathbb{Z}^+ \wedge i \leq k\}$
until $S \neq \emptyset$
 $S_{final} \leftarrow Sort(S_{final})$
return $s_0 \in S_{final}$

The *Expand* function computes new nodes which represent 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 represented 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 contains 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  $s \in S$  do
     $S_{new} \leftarrow \emptyset$ 
     $I_h \leftarrow \text{GroupByHeight}(s.\text{unpacked})$ 
     $placed \leftarrow false$ 
    forall  $(h, I) \in I_h$  do
        forall  $b \in bins$  do
             $placement \leftarrow SPBestInsertion(s_b, I)$  (Algorithm 3)
            if  $placement \neq \emptyset$  then
                 $placed \leftarrow true$ 
                 $S_{new} \leftarrow S_{new} \cup Next(s, placement)$ 
            end
        end
    end
    if  $placed = false$  then
         $S_{new} \leftarrow S_{new} \cup OpenNewBin(s)$ 
    end
end
return  $S_{new}$ 

```

0.1.1. 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.

0.2. Support Plane

We introduce Support Plane (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:  $placement$ 
 $placement \leftarrow \emptyset$ 
forall  $S_z \in planes$  do
     $I_p \leftarrow I \setminus \{i \in I : z + i.h > H_b\}$ 
    forall  $change \in coords$  do
         $I'_{upper} \leftarrow CoordinateChange(change, I_{upper})$ 
         $I'_p \leftarrow CoordinateChange(change, I_p)$ 
         $P' \leftarrow SPPackPlane(W_b, D_b, I'_{upper}, I'_p)$  (Algorithm 4)
         $P \leftarrow CoordinateChange(change, P')$ 
         $P \leftarrow \{i \in P : IsFeasible(i, bin, I_{support}, I_{upper}, aabb)\}$ 
        if  $ComparePacking(placement, P)$  then
             $placement \leftarrow P$ 
        end
    end
    if  $placement \neq \emptyset$  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 represent 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 represented by algorithm 5.

Algorithm 5: SP Apply and Filter

input : s_b, I, z, z_{min}, t **output:** s'_b

//Filter bad planes

 $P' \leftarrow planes \setminus \{S_z \in planes : z \leq z_{min}\}$

//Apply insertion

 $B \leftarrow placed \cup I$ $U \leftarrow unpacked \setminus I$ $T \leftarrow aabb$ **forall** $i \in I$ **do** $T \leftarrow InsertAABB(i, T)$ //If balanced $O(\log(n))$ $generate \leftarrow true$ **forall** $S'_z \in P'$ **do**

//Based on the distance from the top of the item

 $dz \leftarrow S'_z.z - i.z_{max}$ **if** $0 \leq dz \leq t$ **then** $generate \leftarrow false$ $S'_z.I_{support} \leftarrow S'_z.I_{support} \cup i$ **end** **else if** $dz < 0$ **then** $S'_z.I_{upper} \leftarrow S'_z.I_{upper} \cup i$ **end** **end** **if** $generate$ **then** $P' \leftarrow P' \cup (i.z_{max}, \{i\}, \emptyset)$ **end****end****return** $Update(s_b, P', B, U, T)$

0.2.1. Scoring Insertions

0.3. Max Rects

0.4. Feasibility

0.4.1. 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 acceleration 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.

Computational experiments

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.

