Containment Algorithms for Nonconvex Polygons with Applications to Layout

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:34310069">http://nrs.harvard.edu/urn-3:HUL.InstRepos:34310069</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
Containment Algorithms for Nonconvex Polygons with Applications to Layout

Karen McIntosh Daniels

TR-12-95

May 1995

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts
Containment Algorithms for Nonconvex Polygons
with Applications to Layout

A thesis presented
by
Karen McIntosh Daniels
to
The Division of Applied Sciences
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy
in the subject of

Computer Science

Harvard University
Cambridge, Massachusetts

May, 1995
# Table of Contents

List of Figures 7

List of Tables 10

1 Introduction 13
   1.1 Background ......................................................... 13
       1.1.1 Containment, Packing, and Layout .......................... 14
       1.1.2 Thesis Statement and Overview of Chapter .................. 16
   1.2 Overview of Containment Work ................................. 17
       1.2.1 Related Work ................................................. 17
       1.2.2 Our Containment Work ...................................... 20
   1.3 Overview of Layout Work ....................................... 22
       1.3.1 Related Work ................................................. 23
       1.3.2 Convex Items .................................................. 24
       1.3.3 Nonconvex Items ............................................. 25
       1.3.4 Assignment .................................................. 29
       1.3.5 Our Layout Work ............................................ 29
   1.4 Project Background ............................................. 32
       1.4.1 Compaction .................................................. 32
       1.4.2 Panel Placement ............................................. 34
       1.4.3 Trim Placement .............................................. 34
   1.5 Benchmarks ....................................................... 34
   1.6 Contribution ..................................................... 34
   1.7 Overview ......................................................... 35

I Translational Containment 37

2 Preliminaries 38
   2.1 Introduction ..................................................... 38
       2.1.1 Overview of Chapter ....................................... 41
5 Algorithms 100
5.1 Introduction ................................................. 100
5.1.1 Overview ................................................ 101
5.2 An Approximate Containment Algorithm ................. 101
5.2.1 Introduction .............................................. 101
5.2.2 Analysis .................................................. 104
5.2.3 Results ................................................... 105
5.3 A Hybrid Containment Algorithm ......................... 113
5.3.1 Introduction .............................................. 113
5.3.2 Results ................................................... 115
5.3.3 Conclusion ............................................... 119
5.4 Summary and Conclusions ................................... 119

II Applications of Containment to Layout 122
6 Preliminaries 123
6.1 Introduction .................................................. 123
6.1.1 Overview of Chapter .................................. 124
6.2 Overview of Container Transformation Work ............ 124
6.3 Overview of Minimal Container Work .................... 124
6.3.1 Related Work ............................................ 125
6.3.2 Our Minimal Container Work ......................... 126
6.4 Overview of Maximal Item Work .......................... 126
6.4.1 Our Maximal Item Work .............................. 126
6.5 Overview of Part Two ...................................... 127

7 Limited Gaps 128
7.1 Introduction .................................................. 128
7.1.1 Decomposing the Container into Gaps ................. 129
7.1.2 Overview ................................................ 130
7.2 The Challenge of Constructing Gap Sets .................. 130
7.2.1 Compaction ............................................. 130
7.2.2 Partitioning Squares .................................. 130
7.2.3 Other Methods ......................................... 131
7.3 Limited Gaps ................................................. 131
7.3.1 Reachable Regions .................................. 131
7.3.2 Defining Limited Gaps using Probability of Interference .... 132
7.4 Efficient Computation of Limited Gaps ................... 133
7.4.1 Breaking Polygons .................................. 133
7.4.2 Inking Polygons .................................... 133
7.5 MAAPRs ..................................................... 136
List of Figures

1.1 Translational containment problem for six items .......................... 14
1.2 The marker-making task in the apparel industry ............................ 33

2.1 \((B + t) \cap A\) if and only if \(t \in A \oplus -B\) ............................. 42
2.2 \((B + t) \subseteq A\) if and only if \(t \in A \oplus -B\) ............................. 42

3.1 Boundary intersection example 1: the containment problem ................. 53
3.2 Boundary intersection example 1: two one-component \(U_{0j}s\), with \(U_{0j}^+\) .... 53
3.3 Boundary intersection example 1: two two-component \(U_{ij}s\), with \(U_{ij}^+\) .... 54
3.4 Boundary intersection example 2: the containment problem ................. 54
3.5 Boundary intersection example 2: a one-component \(U_{0j}\), with \(U_{0j}^+\) .... 55
3.6 Boundary intersection example 2: a two-component \(U_{ij}\), with \(U_{ij}^+\) .... 55
3.7 An annular configuration space .............................................. 57
3.8 Geometric restriction .......................................................... 59
3.9 SSGR containment problem: place 3 convex items ............................ 62
3.10 A one-component \(U_{0j}\), with \(U_{0j}^+\) for the SSGR of \(U\) .................... 62
3.11 A two-component \(U_{ij}\), with \(U_{ij}^+\) for the SSGR of \(U\) .................... 63
3.12 A 6NN example with 1.8% SSGR restriction ................................ 65
3.13 “L”-shaped container and 3 items .......................................... 65
3.14 A 5NN example with 52% average SSGR restriction .......................... 66
3.15 An infeasible 3CN example ..................................................... 67
3.16 A 3CC example with an unrestrictive container .............................. 67
3.17 A 6NC example with 53% average SSGR restriction .......................... 68
3.18 A 5CC example with 48% average SSGR restriction .......................... 69
3.19 \(U_{23} \cap (U_{21} \oplus U_{13})\) ....................................................... 69
3.20 A 9CC example with strong SSGR restriction ................................ 70
3.21 A highly restrictive 6NN example ............................................ 70
3.22 A 6NN example: effect of SSGR on two one-component \(U_{0j}s\) ................. 71
3.23 Counterexample for \(A \subseteq A \oplus -A\) ....................................... 72
3.24 Selected \(U_{ij}s\) for 6NN example ............................................. 73
3.25 SSLPR containment problem: place 3 nonconvex items 74
3.26 Two two-component $U_{0j}$s, with $U^*_0$ for SSLPR of $U$ 74
3.27 A one-component $U_{ij}$, with $U^*_i$ for SSLPR of $U$ 75
3.28 Effect of SSLPR on $U_{ij}$ when a single component of a $U_{ij}$ is selected 76
3.29 Example for which SSLPR is more powerful than SSGR 77
3.30 Regularization example 80
3.31 Two rectangles with degenerate intersection 80

4.1 3CN example which does not yield a common intersection of translated $U_{0j}$s 85
4.2 No common intersection of translated $U_{0j}$s 86
4.3 No common intersection after maximizing intersection 90
4.4 Common intersection after maximizing convexity of the union 91
4.5 Distance-based subdivision 96
4.6 Common situation for $p = t_j - t_i$ 98

5.1 3NN example: time = 6 seconds 105
5.2 3NN example: time = 5 seconds 106
5.3 4NN example: time = 90 seconds 107
5.4 5NN example: time = 60 seconds 107
5.5 4NN feasible example solved using partial characterization 109
5.6 4NN example: time = 15 seconds 118
5.7 7NN example: time = 176 seconds 118

7.1 Trim placement task 129
7.2 A partitioning square which fragments the container 131
7.3 A neck which is usable although items cannot pass through it 131
7.4 Algorithmic results for a variety of polygon types 136
7.5 Edge contact types for a determining set 138
7.6 Slope lemma 139
7.7 1-Parameter problems with two and three dependent sliding contacts 140
7.8 Determining sets of contacts for the MAAPR 141
7.9 Limited gaps example 144

8.1 Incremental layout task 147
8.2 Loose layout 148
8.3 Final layout 148
8.4 Compaction solution 149
8.5 Minimal square enclosure for 3 nonconvex items 150
8.6 Minimal square enclosure for 4 items 151
8.7 Minimal square enclosure for 5 nonconvex items 152
8.8 Tight rectangular enclosure for 3 convex items 154
8.9 Tight rectangular enclosure for 3 nearly identical nonconvex items 155
8.10 Tight rectangular enclosure for 3 nonconvex items 156
8.11 Tight nonconvex enclosure for 3 items 157

9.1 Scaling circles 161
9.2 Scaling up items in a greedy fashion 164
9.3 Scaling up items in an enhanced balanced fashion 165
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.4</td>
<td>Scaling up two items</td>
<td>166</td>
</tr>
<tr>
<td>9.5</td>
<td>Find a “large” subset which fits</td>
<td>167</td>
</tr>
<tr>
<td>9.6</td>
<td>A “large” subset which fits</td>
<td>168</td>
</tr>
<tr>
<td>9.7</td>
<td>Items and container for randomized construction task</td>
<td>169</td>
</tr>
<tr>
<td>9.8</td>
<td>Randomly constructed 3-item group</td>
<td>169</td>
</tr>
<tr>
<td>9.9</td>
<td>Randomly constructed 5-item group</td>
<td>169</td>
</tr>
<tr>
<td>9.10</td>
<td>Trim placement task for 37188c.mrk</td>
<td>175</td>
</tr>
<tr>
<td>9.11</td>
<td>First-Fit for 37188c.mrk</td>
<td>176</td>
</tr>
<tr>
<td>9.12</td>
<td>Assignment for 37188c.mrk</td>
<td>176</td>
</tr>
<tr>
<td>9.13</td>
<td>Assignment plus First-Fit for 37188c.mrk</td>
<td>177</td>
</tr>
<tr>
<td>9.14</td>
<td>Trim placement task for 41132b.mrk</td>
<td>177</td>
</tr>
<tr>
<td>9.15</td>
<td>First-Fit for 41132b.mrk</td>
<td>178</td>
</tr>
<tr>
<td>9.16</td>
<td>Assignment for 41132b.mrk</td>
<td>178</td>
</tr>
<tr>
<td>9.17</td>
<td>Assignment plus First-Fit for 41132b.mrk</td>
<td>179</td>
</tr>
</tbody>
</table>
3.1 SSGR applied to infeasible examples .............................................. 64
4.1 Number of solutions found without subdivision ................................. 89
5.1 Running times in seconds ((*) denotes infeasible example) ................. 106
5.2 Running time, in seconds, and number of hypotheses examined by approximate algorithm ................................................................. 108
5.3 Number of hypotheses examined by sequential algorithm ((*) denotes infeasible example) ................................................................. 110
5.4 Number of hypotheses examined for identical items ((*) denotes infeasible example) ................................................................. 111
5.5 Exact vs. approximate running times for 3CN examples (100 calls each) .. 113
5.6 Running times for feasible examples in seconds .................................. 116
5.7 Running times for infeasible examples in seconds ............................... 120
5.8 Number of hypotheses examined ((*) denotes infeasible example) ........ 120
5.9 Number of hypotheses evaluated for restriction options ((*) denotes infeasible example) ................................................................. 121
5.10 Number of hypotheses evaluated for infeasible problems (no hypothesis restriction) ................................................................. 121
9.1 Pre-packing results .............................................................................. 174
Abstract

Layout and packing are NP-hard geometric optimization problems of practical importance for which finding a globally optimal solution is intractable if $P \neq NP$. Such problems appear in industries such as aerospace, ship building, apparel and shoe manufacturing, furniture production, and steel construction. At their core, layout and packing problems have the common geometric feasibility problem of containment: find a way of placing a set of items into a container. In this thesis, we focus on containment and its applications to layout and packing problems. We demonstrate that, although containment is NP-hard, it is fruitful to: 1) develop algorithms for containment, as opposed to heuristics, 2) design containment algorithms so that they say “no” almost as fast as they say “yes”, 3) use geometric techniques, not just mathematical programming techniques, and 4) maximize the number of items for which the algorithms are practical.

Our approach to containment is based on a new restrict/evaluate/subdivide paradigm. We develop new theory and practical techniques for the operations within the paradigm. The techniques are appropriate for two-dimensional containment problems in which the items and container may be irregular (i.e. nonconvex) polygons and have multiple components, and in which the items may be translated, but not rotated. Our techniques can be combined to form a variety of two-dimensional translational containment algorithms. The paradigm is designed so that, unlike existing iteration-based algorithms, containment algorithms based on the paradigm are adept at saying “no”, even for slightly infeasible problems. Infeasible problems occur frequently in practice. We present two algorithms based on our paradigm. We obtain the first practical running times for NP-complete two-dimensional translational containment problems for up to ten nonconvex items in a nonconvex container. Most of our examples are from apparel manufacturing. Typically, each item has from 4 to 100 vertices and the container has from 100 to 300 vertices.

We demonstrate that viewing containment as a feasibility problem has many benefits for packing and layout problems. For example, we present an effective method for finding minimal enclosures which uses containment to perform binary search on a parameter. Compaction techniques can accelerate the search. We also use containment to develop the first practical pre-packing strategy for a multi-stage pattern layout problem in apparel manufacturing. Pre-packing is a layout method which packs items into a collection of containers by first generating groups of items which fit into each container and then assigning groups to containers.
Acknowledgments

My research has been carried out under the supervision of my advisor, Victor Milenkovic, Assistant Professor of Computer Science at Harvard, who is now an Associate Professor at the University of Miami. His guidance and support of my research have been superb. I gratefully acknowledge the advice provided by my entire thesis committee: Professor Victor Milenkovic, Professor Fred Abernathy and Professor Harry Lewis. In addition, I thank Professor Les Valiant, who served on my qualifier committee.

This thesis research is part of a project funded by the Sloan Foundation within the Harvard Center for Textile and Apparel Research. The principal investigator is Professor Fred Abernathy, who has helped us forge relationships with industry. Don Walker of the Harvard Office of Technology and Licensing has helped us formalize our relationships with industry. Microdynamics, Inc. and Gerber Garment Technologies have helped us transfer our work to the apparel manufacturing industry.

Special thanks are due to two former Harvard graduate students: Dr. Zhenyu Li of GTE Labs and Dr. Dan Roth. Zhenyu Li was an integral member of our research team while working on his dissertation at Harvard. His comments on drafts of the thesis were very helpful. I am grateful to Dan Roth for his collaboration on finding the maximum-area axis-parallel rectangle in a polygon.

Our research team has involved a number of Harvard undergraduates over the years: Rajarshi Bhattacharya, Lee Wexler, Jacqueline Huang, Jackie Chang, Sanjoy Dasgupta, Venkatesh Reddy, Shivashish Chatterjee, Kirat Singh, Eric Wilfrid, Sanjay Madan, Matt LaMantia, John Immel, Ben Westbrook, Ed Han, and Iraklis Kourtidis. I also thank future Harvard student Umesh Shankar for his work while at the MIT Summer Research Institute.

Administrative assistance from Carol Harlow, Sheila Toomey, Baiba Menke, Pauline Mitchell, and the rest of the staff has been wonderful. The constant support of the Aiken computer systems staff has kept our machines up and running through years of research and many demonstrations. I convey long-distance thanks to Irina Kaliman at the University of Miami for her support of our software and for her translations of Russian layout work.

On a personal note, I thank my family for their encouragement and patience: my daughters, Theresa and Michelle, and my husband Murray Daniels. My mother, Jane McIntosh has also provided constant support. I dedicate this thesis to my husband, Murray, whose user interface code is an integral part of our project’s software, and whose love and support have made it possible for me to pursue this research.
1.1 Background

Many industrial problems involve placing objects into containers so that no two objects overlap each other. The general goal is to either minimize the size of the container or find a maximal\(^1\) collection of placed objects. These problems are known by a variety of names, such as layout, packing, loading, placement, and nesting. A number of industries apply layout techniques when cutting new parts from stock material. In apparel manufacturing, for instance, pattern pieces are arranged on cloth. The goal is to find a non-overlapping arrangement which uses the least amount of cloth. In sheet metal layout, objects are cut from metal stock sheets. The goal here is typically to cut as many objects as possible from a given sheet. In shoe manufacturing, each hide is a different shape, and the goal is to cut as many objects from each hide as possible.

Layout applications are not limited to cutting operations, nor to processes where the goal is to minimize the waste of material. In VLSI layout, for example, rectangular modules are arranged on a chip in such a way as to best meet the two competing goals of minimizing chip area and minimizing the length of interconnections between modules on the chip. In furniture layout an arrangement of furniture usually must satisfy a set of aesthetic criteria. When putting together a jigsaw puzzle, the goal is simply to find a feasible solution.

Placement problems also appear in robotics. The simplest problem is to find a position for a robot amidst a collection of obstacles. This is generalized to planning a motion for the robot through the obstacles. This can be further generalized to problems in assembly planning in which motions must be planned not only for the robot, but also for parts which the robot manipulates.

People typically do not think of all of these problems as similar, as they correspond to entirely different fields of research. However, some work in robot motion planning and computational geometry has noted the correspondence between placement and motion planning problems [Cha83, CK89, AB88, Avn89]. For example, Avnaim [Avn89] solves placement and displacement (motion planning) problems, noting that the former type is useful in layout applications and the latter in robotics. The relationship between cutting, packing,

\(^1\)One possible maximality criterion is the number of objects. Another is the total area of the objects.
and motion planning was also observed by Cuninghame-Green in a survey paper [CG89]. At their core, all these problems have the common problem of containment: find a way of placing a set of objects into a container.

In this thesis, we focus on containment and its applications to layout and packing problems. In Section 1.1.1 we define containment, packing and layout, and show how these form a hierarchy of problems. We also briefly discuss the state-of-the-art with respect to these problems. Section 1.1.2 states our main thesis and provides an overview of the remainder of this chapter.

1.1.1 Containment, Packing, and Layout

In the first part of Section 1.1.1 we define containment and show how it fits into a hierarchy of layout and packing problems. We then discuss how difficult these problems are to solve (their “hardness”) and the current state-of-the-art in solving them in the second part of Section 1.1.1.

Hierarchy of Problems

We define containment as follows: given a container, a set of items, and a set of allowable transformations of the items, find transformations of the items which result in a non-overlapping arrangement of the items within the container. Figure 1.1 shows an example of a two-dimensional containment problem for six nonconvex polygonal items and a nonconvex polygonal container consisting of the complement of some other, already placed, items within the rectangle. The items are pattern pieces from a layout problem in apparel manufacturing. The pieces are allowed to translate but not rotate, and so this is a translational containment problem.

Figure 1.1: Translational containment problem for six items

A packing problem is an extension of a containment problem. Its solution is a non-
overlapping arrangement that also optimizes some criteria applied to either the container or the items. Typically the goal is to either find the “minimal”\(^2\) container into which a given set of items fit, or to find the “maximal” collection of items which fit into a given container. A packing problem is therefore an optimization problem over a range of containers or collections of items. A containment problem considers only the feasibility of placing a given collection into a given container. Thus, containment is the \textit{feasibility problem} corresponding to the packing optimization problem.

One example of a “minimal container” type of packing problem is the classical one-dimensional bin-packing problem. People often think of this as a specialized two-dimensional packing problem for which a set of rectangular (axis-parallel) bins and a set of rectangular items are given. The set of bins is infinite; the set of items is finite. Bins and items all have unit width. Each item has height \(\leq c\), where \(c\) is the common bin height. The task is to minimize the number of bins required to pack all the items using only translation. An example of the “maximal collection of items” type of packing problem is the three-dimensional task of cargo load planning for aircraft, where the goal is to efficiently utilize the available three-dimensional cargo space.

The terms packing and layout are sometimes used interchangeably in the literature. Often, however, a \textit{layout} problem refers to a packing problem which either has additional constraints or multiple goals. For example, as mentioned earlier, a VLSI layout problem has the two competing goals of minimizing chip area and minimizing the length of interconnections between modules. This type of problem also has constraints such as a requirement to maintain certain minimum separation between modules. For other types of layout problems, the constraints are sometimes in the form of aesthetic criteria, as in the case of laying out furniture in a room.

Definitions vary, but we consider containment, packing, and layout to be a hierarchy of problems. \textit{Loading} generally refers to a specific type of three-dimensional layout problem that arises in shipping. In VLSI design, \textit{placement} is that aspect of the layout task concerned with the positions of modules, whereas routing deals with interconnections between modules. The term \textit{nesting} is associated with packing tasks for irregularly shaped items, in particular those which can be placed inside of others.

\textbf{State-of-the-Art}

Work on containment problems for small numbers of items (three or fewer) appears occasionally in the literature on computational geometry and has primarily been motivated by problems in robotics. Section 1.2.1 gives a detailed account of this work.

By comparison, the literature on packing and layout problems is vast. This work appears in the literature on management sciences, engineering sciences, information and computer science, mathematics, and operations research. Section 1.3.1 reviews this work, which employs a broad range of techniques.

For rectangular shapes, the layout and packing literature and algorithms are well-developed. For example, some special types of rectangular problems lend themselves to linear and/or dynamic programming; in particular, when guillotine cuts\(^3\) are used exclusively [DD92]. Ground-breaking work in this area was done in the 1960's by Gilmore and

\(^2\)One possible minimality criterion is area.

\(^3\)A guillotine cut is a horizontal or vertical cut which completely partitions the remaining material.
Gomory [GG65]. There also exists a rich literature on bin-packing for rectangles that contains much theoretical work on the effectiveness of simple heuristics [JGJ84].

For irregular shapes, packing and cutting algorithms have yet to match human performance. In industry, such problems are usually solved either by humans or by computer-based heuristics. Expert humans construct nearly optimal layouts in some domains. One very large clothing company has humans create 1500 layouts per week, averaging about 1.5 hours of labor for a layout of 150 items. The automatic generation of production-quality layouts of irregular items is an active area of research.

What is primarily missing from work on packing and layout is the ability to quickly detect an infeasible problem, that is, to say “no” to the containment question. Infeasible problems arise frequently in practice and this shortcoming significantly limits the types of techniques which can be used in layout and packing.

Saying “no” is difficult. Except in the simplest of cases, heuristics and even humans cannot detect infeasibility. For instance, the example in Figure 1.1 is (not obviously) infeasible. A human could try for a long time to place the items but would never succeed. Furthermore, this failure would not constitute a proof of infeasibility until every possible position had been tried.

Containment in its most general form is NP-hard and therefore computationally intractable if P≠NP. Current containment algorithms have worst-case running times which are exponential in the number of items being placed. Furthermore, current containment algorithms attain their worst-case running times when faced with any infeasible problem. Multiple resolution techniques for approximating polygonal boundaries can be used to improve the ability to say “no”. However, algorithms using these techniques still perform slowly when faced with problems which are just barely infeasible. Figure 1.1 is an example of such a problem. This type of problem arises frequently in practice because a tight packing is the goal of a packing problem.

Another shortcoming of current practical containment algorithms is that they place at most three items. Why has the computational geometry community not produced practical containment algorithms for larger numbers of items? Since containment is NP-hard, perhaps their view is that computational geometry techniques are not powerful enough for such problems and that mathematical programming techniques offer the only chance for success.

1.1.2 Thesis Statement and Overview of Chapter

This thesis provides two-dimensional translational containment algorithms which overcome both of the disadvantages of existing containment algorithms. Our algorithms are designed to place an arbitrary number of items. The algorithms can say both “no” and “yes” quickly. They do not necessarily perform more slowly on “slightly” infeasible inputs. One of our algorithms is practical for up to ten items. The implementation of the other algorithm is currently fast for up to four items. The slowness of our current implementation of polygon set operations makes this algorithm slow for more than four items. The ability to solve containment for up to ten items is powerful. Previously unsolved packing and layout problems for hundreds of items can now be solved using containment as the feasibility

---

4In this case, the problem is infeasible and the global overlap minimum is small but nonzero.
5Our examples are drawn primarily from the apparel industry. Each item is represented as a polygon which typically has from 4 to 100 vertices. The container is also represented as a polygon; it typically has from 100 to 300 vertices. The items and container may be nonconvex.
oracle inside other methods.

Our ability to say “no” well comes primarily from our new geometric techniques. Our ability to say “yes” for so many items is partly due to our geometric techniques and partly due to our application of and improvements to existing linear programming-based techniques. Thus, we demonstrate that a computational geometry approach can contribute significantly to practical algorithms for NP-hard geometric problems.

Our main thesis is that, although containment is NP-hard, it is fruitful to: 1) develop algorithms for containment, as opposed to heuristics, 2) design containment algorithms so that they say “no” almost as fast as they say “yes”, 3) use geometric techniques, not just mathematical programming techniques, and 4) maximize the number of items for which the algorithms are practical.

The remainder of the thesis is divided into two main parts. Part I describes our work on containment algorithms. Part II presents applications of containment to layout. The applications validate our approach to containment.

The remainder of this chapter is organized as follows. Section 1.2 discusses work from the containment literature and then gives an overview of our own work on containment. Section 1.3 describes work from the layout literature and discusses our applications of containment to layout problems. Section 1.4 describes the research project which has motivated the work in this thesis. Section 1.5 discusses benchmarks. Section 1.6 highlights the main contributions of the thesis. Finally, Section 1.7 provides a road map for the remainder of the thesis.

1.2 Overview of Containment Work

Containment problems are, in general, NP-hard. As we show in Section 2.3, containment is NP-complete if the items are allowed to translate but not rotate. Part I of the thesis discusses our work on the NP-complete two-dimensional translational containment problem for polygons. The polygons and container may be nonconvex. We present new theory leading to practical algorithms for up to ten polygons.

The following two subsections provide an overview of related work on containment and of our work on containment. Section 1.2.1 surveys the literature on containment. Section 1.2.2 summarizes our work on containment.

1.2.1 Related Work

The term containment appears as early as 1983 in a seminal paper by Chazelle on two-dimensional polygon containment [Cha83]. Subsequent work on containment has appeared in literature on robot motion planning, computational geometry, and operations research. The layout and packing communities do not seem to use this containment work. Either they are unaware of it, or they have encountered difficulties when implementing it. In layout, the equivalent problem is sometimes called “arrangement”.

The next two parts of this subsection survey related work on containment. The first part discusses work on rotational containment. The second part treats work on translational

---

6Some of the containment research has focused more on combinatorial issues than on implementation details, and so this is not surprising.

7The term arrangement has a very different meaning in the computational geometry community.
containment. For these discussions, we assume that the container is a polygon with \( n \) vertices and that a polygonal item to be placed has at most \( m \) vertices.

**Rotational Containment**

Chazelle [Cha83] considers fitting a single polygon \( P \) into another polygon \( C \), where both translation and rotation of \( P \) are allowed. He observes that, if one only seeks a single solution, it suffices to examine (stable) placements of \( P \) in \( C \) for which \( P \)'s three degrees of freedom are removed. He gives an \( O(mn^2) \) time algorithm for solving the problem when \( C \) is convex, and an \( O(m+n) \) time algorithm if \( C \) is convex and only translations are allowed. He gives a naive algorithm for the general case requiring \( O(m^3n^2(m+n)\log(m+n)) \) time.

Avnaim [Avn89] and Avnaim et al. [AB88] give an \( O(m^3n^3\log mn) \) time algorithm for a single nonconvex polygon in a nonconvex container. Their algorithm yields all the solutions.

Grinde and Cavalier [GC93] use mathematical programming to solve the rotational containment problem for a convex polygon in a convex container. Their algorithm requires \( O(bmn) \) time, where \( b \) is the number of two-contact sets obtained as \( P \) rotates through \( 2\pi \). As \( b = O(m+n) \) when the polygon and container are convex, their algorithm requires \( O(m^2n + mn^2) \) time.

**Translational Containment**

In order to discuss related work on translational containment, we use the *configuration space* concept from robotics. In robotics, a point in the configuration space is a configuration. The components of the configuration are the values of the joint angles and any other parameters necessary to specify the robot position and orientation. For our approach (as in [AB87, Avn89, Dev90]), an essential type of configuration space is the set of *relative positions* for a pair of polygons. For \( k \) polygons and one container, there are \( \binom{k+1}{2} \) such configuration spaces. A configuration in this type of configuration space is a \( (k+1) \)-tuple \( (t_0, t_1, \ldots, t_k) \), where \( t_i \) is a translation of the \( i \)th polygon and \( t_0 \) is a translation of the container, usually set to \((0,0)\). If applying these translations to the polygons yields a non-overlapping arrangement, the \( (k+1) \)-tuple is a *valid* configuration.

In the following discussion, we use a shorthand notation to describe various types of translational containment problems. The shorthand is based on the two definitions below.

**Definition 1.1** The kNN problem is that of finding a valid configuration of \( k \) nonconvex polygons in a nonconvex container. Variations replace \( N \) with either \( C \) (convex), \( R \) (rectangle), or \( P \) (parallelogram).

**Definition 1.2** The \((r,k)\)NN problem is that of finding all subsets of size \( k \) out of a set of \( r \) nonconvex polygons such that the \( k \) polygons can be simultaneously placed in a nonconvex container. The variations of Definition 1.1 apply.

We know of at least one industrial solution to translational containment, and it is used in an interactive setting. That solution appears to be based on inner and outer approximations of polygonal boundaries, and the running time increases with the tightness of the fit. Published algorithms for translational containment for small \( k \) have asymptotic running times which tend to depend on a high power of \( n \). For arbitrary \( k \) and nonconvex polygons
and container, the only known algorithms have running time which is exponential in $k$. This is not surprising, since $k$-NN is NP-complete.

A “naive” $k$-NN algorithm requires $O((mn)^{2k} \log mn)$ time. It searches for a valid configuration by iterating over all choices of $2k$ contacting vertex/edge pairs among the polygons and container. More details appear in Section 2.6. The naive algorithm has running time in $\Omega((mn)^{2k})$ because it must iterate over all possible choices of contacts whenever the problem is infeasible.

Fortune [For85] gives a solution to 1CN by computing the Minkowski sum using a generalized Voronoi diagram. This $O(mn \log mn)$ result is an improvement over the $O((n^2 + mn) \log mn)$ time algorithm for 1CN given by Baker et al. [BFM86]. Avnaim and Boissonnat [AB87, Avn89] use the Minkowski sum and convex decomposition to solve 1NN and 2NN, and Avnaim [Avn89] gives an algorithm for 3NN which iterates over the edges of the configuration space polygons. Devillers [Dev90] gives faster algorithms for 2CN and 3CN. These running times are summarized below.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$k$-CN</th>
<th>$k$-NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$O(mn \log mn)$ [For85]</td>
<td>$O(m^2 n^2 \log mn)$ [AB87, Avn89]</td>
</tr>
<tr>
<td>2</td>
<td>$O(m^2 n^2 \log m)$ [Dev90]</td>
<td>$O(m^4 n^4 \log mn)$ [AB87, Avn89]</td>
</tr>
<tr>
<td>3</td>
<td>$O(m^3 n^3 \log m)$ [Dev90]</td>
<td>$O(m^{14} n^6 \log mn)$ [Avn89]</td>
</tr>
</tbody>
</table>

Avnaim and Boissonnat also give a solution to the 3NP problem, three nonconvex polygons in a parallelogram container, using time in $O(m^{60} \log m)$ [AB87, Avn89]. Their 3NP algorithm is based on polygon unions, intersections, and Minkowski sum operations. Even though the asymptotic running time of the 3NP algorithm contains a polynomial whose degree is high compared to the others, in practice the 3NP algorithm is faster than the 3NN algorithm, which is, in turn, faster than the naive algorithm. (We defer further discussion of this until Section 2.6.) Unfortunately, Avnaim shows that there is no formula for a solution to 3NN purely based on polygon union, intersection, and Minkowski sums [Avn89]. This means that, for $k > 2$, any $k$-NN algorithm must use iteration over edges, or subdivision, or some technique other than set operations on polygonal regions. The fact that Avnaim’s algorithm for 3NN uses iteration gives it the $\Omega(m^{14} n^6)$ lower bound.

Milenkovic, et al. [DML94] offer three approaches to translational containment. The first method operates on convex polygons. It iterates over combinations of orientations for the lines which separate the convex polygons. This approach yields an algorithm for 3CN with running time in $O(m^3 n \log mn)$, an improvement of a factor of $O(n^2)$ over previous algorithms (in particular, [Dev90]). The second method uses a MIP (mixed integer programming) model for $k$-NN. This approach is described in detail in [Li94]. It takes one or two minutes on a typical workstation for two or three polygons, but is slow for four or more polygons (it can easily take more than an hour to detect an infeasible 4CN problem). The third method is an algorithm for $k$-NN which finds an $\epsilon$-approximate configuration using time in $O\left(\frac{1}{\epsilon^k} \log \left(\frac{1}{\epsilon^k}\right) k^6 s \log s\right)$, where $s$ is the maximum number of vertices in a polygon.

---

8For polygons $A$ and $B$, the Minkowski sum is defined as $A \oplus B = \{a + b \mid a \in A, b \in B\}$. See Section 2.2 for more details.

9For an $\epsilon$-approximate configuration, no point of any polygon is more than $2\epsilon$ inside of the boundary of any other polygon when the polygons are laid out according to the configuration. For some cutting applications, CAD vendors round polygon vertex coordinates to integers. In such cases, an overlap of up to one unit (corresponding to .01 inches) is acceptable.
which is created through successive applications of a certain collection of polygon set operations (see the discussion of size analysis in Section 3.2.4). This algorithm is introduced in [DM, DM95], and appears in Chapter 5 of this thesis.

Milenkovic [Mil, DM95] improves upon the 3CN algorithm of [DML94] by giving an algorithm which has the same asymptotic running time but is simpler to implement. He also introduces an exact $k$NN algorithm based on linear programming, and approximation and subdivision of the configuration spaces, which runs in $O((mn)^{2k+1}LP(2k, O(kmn + k^2m^2)))$ time, where LP($a, b$) is the time to solve a linear program with $a$ variables and $b$ constraints. This algorithm is practical for $k \leq 3$.

1.2.2 Our Containment Work

Existing two-dimensional translational containment algorithms developed by other researchers are practical only for $k \leq 3$. We present new theory leading to algorithms which handle any $k$. One of our algorithms is practical for $k \leq 10$; the implementation of the other algorithm is currently fast for $k \leq 4$. As stated in Section 1.1.2, our examples are drawn primarily from the apparel industry. Each item is represented as a polygon which typically has from 4 to 100 vertices. The container is also represented as a polygon; it typically has from 100 to 300 vertices. The items and container may be nonconvex. Under these conditions, we can solve containment problems quickly, typically in several minutes or less on a 50 MHz SPARCstation\textsuperscript{10}. Our work is therefore practical for solving a number of layout-related problems for nonconvex polygons.

Although containment is NP-hard in general, we believe that it is valuable to make the practical value of $k$ as large as possible. Thus, we continue the “bottom-up” progression (1CC, 1NN, 2CN, 2NN, etc.) of containment work which has taken place in robotics and computational geometry. Our ability to say “no” well comes primarily from the use of our new geometric techniques. Our ability to say “yes” for so many items is partly due to our geometric techniques and partly due to our application of and improvements to existing linear programming-based\textsuperscript{11} techniques. Thus, we demonstrate that a computational geometry approach can contribute significantly to practical algorithms for NP-hard geometric problems.

Saying “no” quickly is important. Iteration-based algorithms for containment attain their worst-case running time when the containment problem is infeasible. This situation occurs frequently in practice. We replace iteration with subdivision. Our algorithms are specifically designed to detect infeasibility quickly, and we have some theoretical justification for their behavior, which we discuss in more detail later in the thesis.

We present a new paradigm for containment algorithms. Our containment algorithms apply three operations, in order, to the configuration spaces: 1) restriction, which prunes the spaces and helps detect infeasibility, 2) evaluation, which tries to find a solution, and 3) subdivision, which partitions one of the configuration spaces into two parts. The algorithm then recurses on each part while keeping the other configuration spaces constant.

We introduce new theory and practical techniques for each of these types of operations for the case of two-dimensional containment problems in which the items and container may be irregular (i.e., nonconvex) and have multiple components, and in which

\textsuperscript{10}SPARCstation is a trademark of SPARC, International, Inc., licensed exclusively to Sun Microsystems, Inc.

\textsuperscript{11}We use the CPLEX 2.1 linear programming system (CPLEX Optimization, Inc.).
the items may be translated, but not rotated. Our techniques can be combined in various ways to form two-dimensional translational containment algorithms which follow the restrict/evaluate/subdivide paradigm. The paradigm is designed so that, unlike existing iteration-based algorithms, containment algorithms based on the paradigm are adept at saying “no”, even for slightly infeasible problems.

At present there appear to be two approaches to containment within the restrict/evaluate/subdivide framework: direct and indirect. Direct methods act on the configuration spaces directly, and these methods tend to be based on computational geometry. Indirect methods act on convex approximations to the configuration spaces: either inner or outer approximations. Indirect methods use linear programming or other types of mathematical programming, which is why they can only act on convex approximations. Direct methods have a more accurate view of the “world”, but they can only operate on a few (three, in our case) configuration spaces at a time. Indirect methods can act on all the configuration spaces simultaneously in a single operation, but they use only approximations to the configuration spaces.

The thesis presents two new algorithms based on our paradigm. They are briefly described below. The first is a direct algorithm. We call the second a hybrid algorithm because it uses an indirect evaluation method, and both direct and indirect methods for restriction.

Direct Algorithm

This thesis presents a new direct algorithm for containment. It is an approximate algorithm which produces an $\epsilon$-approximate configuration whose overlap is below a given tolerance. As mentioned earlier, for some cutting applications, CAD vendors round polygon vertex coordinates to integers. In such cases, an overlap of up to one unit, corresponding to 0.01 inches, is acceptable. Our approximate algorithm uses our new direct techniques for restriction, evaluation, and subdivision. It appears in [DM95, DM] as well as in Section 5.2.

Our direct restrictions are based on polygon set operations, and so we call them geometric restrictions. They are powerful tools for detecting infeasibility. We define two general types of restrictions: valid restrictions, which preserve all solutions to a containment problem, and semi-valid restrictions, which preserve a single solution, if one exists. We present a variety of restrictions and several general properties of valid restrictions. We examine, in detail, the effectiveness of our geometric restrictions, which are valid restrictions. Our characterization of the types of containment problems for which these restrictions are effective is based on theoretical as well as experimental results.

The evaluation method uses restriction together with a geometric method for maximizing the intersection of translations of the configuration spaces. The evaluation method often finds a containment solution without any subdivision. The subdivision method is a size-based one. Its goal is to partition a configuration space into two parts, each smaller than the unsubdivided configuration space.

Our approximate algorithm has an extra layer of structure on top of the restrict/evaluate/subdivide paradigm. This layer has a set of containment subproblems to which coarse-grained parallelism could be applied in the future. We create the subproblems by characterizing a valid configuration for translational containment.

The current implementation of this algorithm is fast for up to four items. For $k > 4$, the current implementation of our geometric restrictions, based on polygon set operations, slows the algorithm down considerably. Increasing the speed of our polygon set operations
is a subject of future work (see Section 10.2.1). In our experiments, our approximate algorithm performs fewer subdivisions than the faster, hybrid algorithm, and so we believe the approximate algorithm has the potential to be competitive with the hybrid algorithm.

**Hybrid Algorithm**

In addition to our direct algorithm, we also present significant improvements to the restriction, evaluation, and subdivision techniques of Milenkovic's exact \( k \)NN algorithm [Mil], which is an indirect algorithm. Milenkovic's indirect algorithm, in its original form, was slower than our direct algorithm for infeasible problems, and its practical upper bound on \( k \) was \( k \leq 3 \). The improved indirect algorithm is currently our fastest containment algorithm for \( 4 \leq k < 10 \).

Our first improvement to the indirect algorithm is a proof that restrictions, when used in the indirect algorithm, do not increase the algorithm's worst-case running time if the algorithm bases certain decisions on the *unrestricted* configuration spaces. This is important because restrictions can add vertices to the configuration spaces, and so they might have increased the worst-case running time. The proof allows us to add our new direct (geometric) restrictions to the algorithm without increasing the worst-case number of subdivisions. It also allows us to apply a new linear programming-based restriction method, developed by Milenkovic.

The second improvement to the indirect algorithm is related to the evaluation method. We apply the work of Li and Milenkovic on overlap resolution [Li94, ML, LM93c, LM93a] to improve the evaluation step. Their overlap resolution algorithm is an indirect algorithm. It uses inner convex approximations, an approach which complements the outer approximations of the indirect containment algorithm. Our third improvement is our new practical subdivision technique which attempts to maximize the distance from the current evaluation to the convex approximations of the subdivided configuration spaces.

We call the resulting algorithm a *hybrid* algorithm because it uses indirect methods for evaluation, developed by Li and Milenkovic, and both direct and indirect methods for restriction. The hybrid algorithm is described in [MD95] as well as in Section 5.3.

The hybrid algorithm, in addition to being a formidable containment algorithm, has the property that a partial containment solution from it is valuable. This is because each evaluation step produces either a non-overlapping solution or an overlapping configuration for which a linear program has reached a local minimum. As a result, the hybrid algorithm, when terminated early, is the most powerful heuristic known for generating a local overlap minimum.

### 1.3 Overview of Layout Work

Part II of the thesis discusses our application of containment to layout problems. Layout and packing are NP-hard problems for which finding a globally optimal solution is generally intractable. The more restricted problem of packing squares onto a rectangle without rotation (as in VLSI applications) is NP-complete. Rectangle packing is also NP-complete when rotation is not allowed, even for the one-dimensional bin-packing problem [JGJ84]. In general, translational packing is NP-complete.

In this section we survey related work in layout and then describe our applications of containment to layout. Section 1.3.1 discusses related work at a high level. Section 1.3.2
reviews work on layout of convex items. Section 1.3.3 treats work on nonconvex items.

1.3.1 Related Work

Work on layout is spread across a broad range of literature. Relevant areas include [Dyc90]: management sciences, engineering sciences, information and computer science, mathematics, and operations research.

Work on layout, cutting, packing, loading and nesting has been surveyed in [Hin80, SP92, Dyc90, DF92, DD92, DD]. Sweeney and Paternoster give an application-oriented bibliography for cutting and packing problems in [SP92]. Dowsland and Dowsland survey research on packing problems in [DD92]. In [DD], the same authors review work on irregular nesting problems, emphasizing the different types of approaches which have been taken. Dyckhoff analyses the structure of cutting and packing problems and presents a classification in [Dyc90]; this forms the basis for Dyckhoff and Finke's meta-survey for cutting and packing literature [DF92].

In [DF92], Dyckhoff and Finke distinguish four major types of cutting and packing problems based on the way items are assigned to containers and on the assortment of items. They observe that, in the layout literature, problems of the same type tend to be solved using the same technique. The four types are: 1) bin-packing, in which heterogeneous items are assigned to a selection of containers, 2) cutting stock, which is similar to bin-packing, but the items can be partitioned into a small set of equivalence classes, based on shape, 3) knapsack, in which all the containers must be used and the items are heterogeneous, and 4) pallet loading, in which all the containers must be used but the items are homogeneous. These bin-packing and knapsack definitions are more general than typical usage in the literature. Because of their difficulty, bin-packing problems are typically solved sub-optimally using fast, simple heuristics such as First-Fit. The small number of distinct shapes involved in cutting stock problems often allows them to be solved using repeating patterns. Knapsack problems are frequently solved using branch-and-bound or dynamic programming. For a pallet loading problem, the containers are often homogeneous. In this case, a solution for one container can be applied to all the others.

Layout techniques cover a wide range of methods. In some cases exact techniques from mathematical programming, such as linear or dynamic programming, are appropriate for all or part of the problem. Integer and mixed integer programming are still considered too slow for even moderate sized problems. Some researchers are experimenting with AI approaches, such as heuristic search, knowledge-based systems, case-based reasoning, and neural networks. Other researchers use improvement strategies based on "meta-heuristics" such as simulated annealing, genetic algorithms and tabu search\[12\]. Other techniques include: database/substitution, greedy heuristics, Monte Carlo placement, lattice packings, and clustering methods.

Our discussion of layout work treats only two-dimensional layout problems because the containment problem we solve is two-dimensional. We begin in the first part of Section 1.3.2 with the simplest of situations in which the items are rectangular. Next, we consider arbitrary convex items in the second part of Section 1.3.2. Section 1.3.3 discusses nonconvex items. The amount of published research on nonconvex items in nonconvex containers is quite small compared with the amount of research on convex items. Finally, we discuss in

\[12\]Tabu search is a (usually deterministic) neighborhood search technique with mechanisms for managing restrictions on "taboo" regions of the search space and for keeping historical information [Ree93].
Section 1.3.4 how the layout literature treats the assignment problem, that is, the problem of assigning items to different components of a container. The assignment problem is a combinatorial optimization problem which can arise as a subproblem of containment when the container consists of multiple (non-identical) components.

1.3.2 Convex Items

Here we review layout literature related to convex items. The first part of Section 1.3.2 surveys work on rectangular items, and the second part of Section 1.3.2 discusses work on convex items which need not be rectangular.

Rectangular Items

Dyckhoff [Dyc90] observes that most problems in the cutting and packing literature deal with shapes which can be specified using a small number of parameters, such as the rectangle. Research into problems for which both the container and items are rectangular is reviewed in [DF92, DD92, JGJ84].

Coffman, et al. [JGJ84] survey approximation algorithms for bin-packing. The survey focuses on approximation algorithms which are effective in practice, and simple enough so that their worst-case or average-case performance has been analyzed in the literature. These include, for example, First-Fit, Next-Fit, Best-Fit, and their variations. One-dimensional bin-packing was defined above in Section 1.1. In two-dimensional strip-packing, a semi-infinite vertical strip of unit width is given, along with a set of rectangular items of width \leq 1. The goal is to pack items on the strip while minimizing the height of the strip. The classical two-dimensional bin-packing problem [JL91] is a variant on strip-packing in which horizontal boundaries are added to the strip at integer heights. Few published analyses exist for the case where the bins are not identical.

Israni and Sanders [IS82] present heuristics for packing non-identical rectangles. One uses decreasing length, perpendicular strip placement (DLPER), and another uses decreasing height, perpendicular strip placement (DHPER). These heuristics essentially place the rectangles in nested “L”-shaped strips. In [IS85], Israni and Sanders compare the performance of DLPER and DHPER with other bin-packing heuristics.

Some special cases of rectangular two-dimensional packing for cutting stock problems lend themselves to linear and/or dynamic programming; for example, when guillotine cuts are used exclusively [DD92]. Gilmore and Gomory [GG65] solve some restricted types of guillotine cutting problems using a linear programming formulation whose speed relies on solving knapsack problems (the knapsack problems can be solved using dynamic programming). Dagli and Tatoglu [DT87] observe that if each item has a “weight” and if at least one dimension of the items and containers is equal, the two-dimensional problem reduces to a one-dimensional single container problem which can be solved with dynamic programming.

\footnote{An approximation algorithm for an optimization (minimization) problem has the property that the value of its output is } \(1 + \varepsilon\) times the optimal (minimum) value. This is different from the definition of an approximate algorithm, in which the solution generated by the algorithm is accurate to within a tolerance \(\varepsilon\). For example, a containment algorithm which produces an \(\varepsilon\)-approximate configuration is an approximate algorithm because the configuration may contain small overlaps.

\footnote{We remind the reader that a guillotine cut is a horizontal or vertical cut which completely partitions the remaining material.}
In [AA76], Adamowicz and Albano lay out rectangles by first forming candidate strips, and then selecting a subset of candidate strips using dynamic programming. Haims and Freeman [HF70] also use dynamic programming to place rectangles; they use linear programming and two-dimensional dynamic programming and take a multi-stage approach.

**Convex Items**

Non-rectangular packing results have appeared primarily for problems that pack identical copies of a convex polygon. It is well known that the densest packing by translates of a convex polygon is given by a lattice packing [Rog51]. In a lattice packing, the polygons are placed at points on a lattice. A lattice is determined by two linearly independent vectors; a point on the lattice is an integral linear combination of the two vectors. The densest lattice packing of an \( n \)-vertex convex polygon can be found in \( O(n) \) time [MS90b]. A double-lattice packing is the union of two lattice packings, such that a rotation of either of the two lattice packings by \( \pi \) produces the other lattice packing. The densest double-lattice packing can be found in \( O(n) \) time for any convex polygon that is allowed to rotate by \( \pi \) [Mou91]. The density\(^\text{15}\) of such a packing is \( \leq \sqrt{3}/2 \) [KK90]. Doré and Ben-Bassat [DBB84] show how to find an appropriate circumscribing hexagon for a convex polygon in order to find a lattice packing.

\[1.3.3 \text{ Nonconvex Items}\]

Dowsland and Dowsland [DD] survey work on irregular nesting problems, highlighting the different types of approaches which have been taken. No fast, high quality solutions have emerged to date for layout problems involving a variety of nonconvex items. Many algorithms for packing nonconvex objects are unpublished; [QS87] suggests this may be due to “commercial confidentiality”. Published work with nonconvex items appears to be narrowly focused on the particular application under consideration.

Published methods for packing nonconvex objects can be classified (as in [AS80]) as: 1) indirect, (or two-stage) where nonconvex items are first enclosed in convex polygons, and techniques for packing convex objects are then applied, and 2) direct, where the nonconvex polygons themselves are manipulated. In the following discussion, we restate (1) more generally in terms of clustering; an item or collection of items is replaced by an approximation, and placement is performed on the approximations. We further subdivide (1) as follows: (a) preprocessing: clustering occurs only as a preprocessing step, and (b) integration: clustering is integrated into the layout algorithm.

**Clustering**

**Preprocessing:** One popular preprocessing method encloses polygons in rectangles. The advantage of this is the abundance of literature on packing rectangles; the disadvantage is that the resultant waste typically makes this approach unacceptable in practice. For rectangular containers, Adamowicz and Albano [AA76], Haims and Freeman [HF70], Amaral, et al. [ABJ90], Dağlı and Tatoglu [DT87], and Bailléul et al. [BTS83] pack nonconvex items into rectangles to form composite items, and then place the rectangles. In [AA76], Adamowicz and Albano give an algorithm for pairwise clustering of irregular shapes in a

\(^{15}\)Packing density is the ratio of used area to total area.
rectangle, and describe limited clustering options for more than two items. They assume the shapes can rotate. The rectangles are placed in strips using dynamic programming. (Linear and/or dynamic programming are not perceived as feasible options unless nonconvex items are enclosed first within rectangles.) Amaral, et al. [ABJ90] adopt the approach of [AA76] for clustering items.

Böhme and Graham [BG79] enclose each item within another polygon, and then use Monte Carlo methods to place each polygon onto rectangular metal plates. It has been suggested that a nonconvex polygon be approximated by its convex hull, if the waste is not too large [CG89, DBB84]. If the waste is large, Dori and Ben-Bassat [DBB84] suggest that small groups of nonconvex items be combined into a convex object and then convex packing techniques can be applied. This approach is also followed by Karoupi and Loftus [KL91]. However, this leaves open the problem of how to combine nonconvex items automatically into a convex (or nearly convex) one. Heistermann and Lengauer [HL92] observe that it is not known what conditions a composite part must satisfy in order to be useful in nesting. If the composite convex objects are different from each other, is not clear that algorithms for packing non-identical convex objects have advanced enough to make such an approach practical [DD92].

If a large set of nonconvex items to be placed on a rectangle (or rectangular strip) contains many identical items, an efficient layout can often be created using repeating patterns. For some item shapes, lattice or double-lattice packings are appropriate. However, it is not necessarily true that the densest packing of identical nonconvex polygons is lattice-based [BK87].

Prasad and Somasundaram [PS91] and Babaev [Bab82] follow the repeating pattern approach for nesting sheet-metal blanks. They form an efficient cluster of items, then build a repeating pattern. Ismail and Hon [IH92] use a genetic algorithm to cluster two similar shapes together.

To reduce waste, an indirect method can approximate each polygon using a collection of polygons. For example, Qu and Sanders [QS87] approximate nonconvex items by non-overlapping rectangles, then use a greedy layout algorithm to arrange the rectilinear items into layers of descending staircase patterns. They assume the container is a set of rectangular stock sheets.

Tanaka and Wachi [TW73] place horizontally convex approximations of garment pieces on a rectangular strip using heuristic search. They report material waste of 28% to 39%. Nakajima and Hayashi [NH84] improve upon the algorithm of [TW73] using several techniques; they obtain slightly less waste than [TW73].

Heistermann and Lengauer [HL92] use a flexible approximate representation for nonconvex items and a nonconvex container. To compute the approximation, they first construct a set of triangles along the boundary, then iteratively smooth the contour, selecting, at each step, the smallest area triangle. This technique could be used in our containment work.

**Integration:** Some work rests on the observation that good local nesting often produces good global nesting. Dağlı and Tatoğlu [DT87] cluster pairs of items in a greedy fashion. They first cluster two items, make them a unit, cluster this unit with another item, and so on. Babaev [Bab82] also places non-identical nonconvex items this way. Dighe and Jakiela [DJ95] present a hierarchical genetic algorithm. The algorithm uses a binary tree in which each item (together with values for its three parameters: x, y, and θ) constitutes a leaf, and a node represents a search for the minimal enclosing rectangle of its two children. Thus, the genetic algorithm seeks the best binary tree for a set of items. They note that “genetic
algorithms rely on the hypothesis that combining two good building blocks leads to another good building block”.

Milenkovic, et al. [MDL92] take advantage of the large/small item dichotomy and natural equivalence classes in parts layout problems. To place large items, they build column clusters of four items and form a grid-like layout consisting of a set of columns. An algorithm based on dynamic programming (and allowing backtracking) adds one column at a time to the layout. The technique of clustering items into columns also appeared in the work of Gurel at IBM in the 1960’s [Gur68b, Gur68a, Gur69].

Heckmann and Lengauer [HL95] use dynamic approximations for items. As temperature decreases in their simulated annealing process, they use tighter approximations. They report that this approach can reduce running time by up to 15%. This idea of using tighter approximations later in the process is similar to the use of convex hulls in [Mil] and in our hybrid containment algorithm; as subdivision proceeds, the convex hulls become better approximations to the subdivided portions of the configuration spaces.

Direct

Dealing directly with an assortment of nonconvex polygons is an extremely difficult problem. If many items are similar in shape and size, it can be useful to partition the set of items into equivalence classes. For example, if a large/small item dichotomy exists, it is common to place all the large items first and then place the small ones [Bab82, MDL92, ABJ90].

In some applications the problem definition specifies equivalence classes. For wood [OF93] and sheet-metal cutting [PS91, BG79, Bab82], there is often more than one copy of each item, and so an equivalence class might contain all identical items. Items in apparel manufacturing are usually similar in shape and size because a layout typically contains a mixture of sizes for the same garment style. Heistermann and Lengauer [HL92] use topology classes of nonconvex polygons to help select the next item to be placed on a leather hide using a greedy strategy. One of the ways Nakajima and Hayashi [NH84] improve upon the algorithm of Tanaka and Wachi [TW73] is to apply the Group Technology manufacturing principle to classify items according to a criterion such as size. Group Technology is defined by Moon and Kao [MK93] as “a manufacturing principle and practice of identifying similarities among diverse parts, forming part families, classifying new parts into part families, and efficiently storing and retrieving information about part families.” See [HW89] for more details. We note that grouping together similar items can have benefits for containment; see the discussion of symmetry breaking and subset substitution restrictions in Section 3.8.

Some of the earliest work in direct placement of nonconvex items was done by IBM and used a greedy approach to placement (see, for example [MHI69]). An early nongreedy approach was taken by Adamowicz [Ada69]. Albano and Sappupo [AS80] note that the experimental system in [Ada69] involves iterative application of linear programming. For problems of nontrivial size, techniques of mathematical programming have been viewed as too slow even in cases where they can be applied. Integer and mixed integer formulations (for exact solutions) have, to date, led to programs whose running time is unacceptably slow. AI and meta-heuristic techniques can produce layouts with low waste, but, again, at the cost of much computation time (e.g. several hours even for 20 or 30 items).

Some researchers are experimenting with AI approaches, such as heuristic search, case-based reasoning, knowledge-based systems, and neural networks. For example, the Clavier
system, described in [HT94], uses cased-based reasoning to arrange composite parts in a convection autoclave (oven) for a curing process. Yuzu et al. [YW87] present an expert system for placing nonconvex polygons in the garment industry. Albano and Sappupo [AS80] pack assortments of nonconvex polygons using heuristic search techniques. They allow nonconvex items to rotate by $\pi$ and they pack them onto a rectangle. They control the number of nodes expanded in their search tree by, for example, introducing a waste parameter. They present results for the value of 20% waste for some very small apparel examples containing no more than 30 items. They assume the container is a single rectangle large enough to contain all the items.

Other researchers use improvement strategies for nonconvex polygons based on meta-heuristics such as simulated annealing, genetic algorithms, and tabu search. A central question for such methods is how to reconcile the two competing goals of minimizing waste and eliminating overlaps. Some researchers explicitly forbid overlap; others allow it but assign a penalty for it. For example, the Dowsland survey discusses the tabu search algorithm of Blazewicz et al. [BH93]; this algorithm forbids overlaps. Both approaches rely on fast overlap detection. The latter also requires powerful overlap reduction which, until the recent work of Li and Milenkovic [Li94, ML93c, LM93a], has not been available. This thesis offers an even more powerful overlap reduction heuristic (recall our discussion of the overlap reduction evaluation method within the hybrid containment algorithm in Section 1.2.2).

The Dowsland survey [DD] discusses some simulated annealing work, such as that of Oliveira and Ferreira [Of93]. We augment the Dowsland list with two more examples. Mangen and Lasudry use simulated annealing for marker-making in the garment industry [ML91]. They forbid overlaps. Heckmann and Lengauer [HL95] apply simulated annealing to textile manufacturing. They allow overlaps and use a penalty term. In both of these cases, items are nonconvex and the container is a rectangular strip. (In the latter case, the nonconvex polygons which are laid out are nonconvex approximations of the original items.) Heckmann and Lengauer obtain waste levels comparable to human levels for small numbers of items (15-25). Achieving this requires multiple runs of their algorithm and from one to five hours of computing time. Heckmann and Lengauer also use simulated annealing to solve a fifteen piece puzzle in seven hours; in this case the container is a rectangle. Heckmann and Lengauer compare their simulated annealing results to results they obtain using search techniques closely related to simulated annealing, such as Threshold Accepting (TA). The primary differences between TA and simulated annealing are: 1) TA has only one parameter, and 2) "it accepts every new configuration which is not much worse than the old one" [Due93].

Dighe and Jakiela [DJ95] survey the published work on genetic algorithms for layout and introduce genetic algorithms capable of placing irregular shapes. They forbid overlaps between items. In addition to the hierarchical clustering algorithm described earlier, Dighe and Jakiela give an alternate genetic algorithm which is based on ordering the items and adding them one at a time to the layout without overlap, using a leftmost heuristic. In this case, the genetic algorithm seeks the best ordering of the items.

Li and Milenkovic [Li94, ML93c, LM93a] introduce a substitution-based approach to pants marker-making for the garment industry. Given a database of markers, a new marker can be created by finding a template marker in the database whose items are similar.

\footnote{Marker-making is the process of laying out pattern pieces on cloth.}
to the new ones, arranging the new items in the same pattern as those of the template, and then eliminating overlaps. This method is appropriate for applications in which there is a large collection of previous layouts, and the items and containers are sufficiently similar from one layout to another. It is not appropriate, in general, for sheet metal work, where the layouts are different enough from each other that humans find it difficult to develop expertise. It is also not appropriate for leather nesting problems, because each hide is different with respect to both shape and the location of defects [HI92].

1.3.4 Assignment

The assignment problem is a combinatorial optimization problem which can arise as a subproblem of containment when the container consists of multiple components which are not identical and are considered separately. If the layout algorithm treats components individually, it must decide which items to assign to each component of the container. Arbel [Arb93] identifies two different assignment strategies: 1) pre-packing, where feasible groups for each container are generated, and then assignment is performed, and 2) post-packing, where groups are assigned based on some estimate of efficiency, without determining their feasibility.

There is a rich literature on combinatorial optimization. See [CMTS79, PS82, Law81, LK88, SLS85] for discussions of exact methods, and [Ree93] for a treatment of modern heuristics. In spite of this, discussions of assignment in the literature on nonconvex layout are rare. Often the issue is either ignored, or else resolved using a greedy strategy. Daboli and Tatoglu [DT87] observe that most researchers do not handle allocation of items among multiple containers. Arbel [Arb93] dismisses pre-packing as prohibitively expensive, and therefore uses post-packing to pack nonconvex items on rectangular stock sheets. His post-packing method uses integer programming. Böhme and Graham [BG79], who use Monte Carlo methods to place items onto metal plates, state that the set of candidate plates is selected by a preprocessor; neither this algorithm nor the method of selecting the next plate to pack is described. Qu and Sanders [QS87], who approximate nonconvex items by non-overlapping rectangles, do not indicate how their algorithm selects the next sheet to be packed. Heistermann and Lengauer [HL92], for whom a nonconvex container is partitioned into quality zones, use a greedy algorithm which treats one region at a time.

To our knowledge, no successful pre-packing strategies appear in the literature.

1.3.5 Our Layout Work

We claim that the ability to quickly solve containment problems leads to improved methods for solving a variety of layout problems. Recall that our containment approach is a "bottom-up" one in which we increase the practical value of $k$. Clustering techniques for layout also use a bottom-up approach. Some of them even allow the items to rotate. However, the methods which cluster more than two arbitrary nonconvex polygons are greedy heuristics. Our containment-based solutions to minimal container problems do not allow the items to rotate, but, in contrast to the heuristics, we establish an actual lower bound on the size of the container for multiple items.

In addition, we believe that some potentially promising layout techniques have been abandoned by the layout community because they lack strong bottom-up methods. The primary example of this is pre-packing. If the layout world had a practical method for
generating groups of items which fit into each container, this, plus an appropriate assignment strategy, would make pre-packing viable. We demonstrate, through our containment-based solutions to maximal item problems, that pre-packing can work.

Before describing our applications in more detail, we pause to argue that, in general, feasibility problems can help to solve optimization problems. We also show how to construct geometric optimization problems from a containment problem.

One need only look as far as the Russian ellipsoid method in linear programming [Kha79] to find a powerful example of how solving feasibility problems can facilitate solving optimization problems\(^{17}\). In addition, a feasibility problem is a decision problem which can be used as an oracle in order to solve problems involving a parameter. For example, one can perform binary search on the value of the parameter, or, in some cases, one can use Meggido's parametric search technique [Meg83]. The binary search approach has been suggested by Chazelle [Cha83] in the computational geometry literature as a way to find the maximum scaling factor which allows a polygon to fit inside another polygon. In the literature on combinatorial optimization, Reeves [Ree93] discusses binary search and comments: “However, we would not usually try to solve an instance of an optimization problem by this means!”. Our belief is consistent with both of these views. Binary search is valuable, but pure binary search alone is insufficient. The best approach, in practice, accelerates the binary search by using an improvement strategy (typically a heuristic) to help update the upper bound.

This method provides one way to establish a lower bound for a solution to a minimization problem. The bound can be compared with a value obtained by a heuristic. The ability to solve a feasibility problem thus makes it possible to evaluate the effectiveness of a heuristic, whether or not the heuristic is automated. Evaluating the effectiveness of heuristics is a challenging and important task. Other techniques exist for establishing lower bounds for some types of optimization problems. For example, in the area of combinatorial optimization, lower bounds for minimization problems can be found using relaxation techniques\(^{18}\) [Ree93].

We construct two different types of geometric optimization problems from a containment problem. The first type is a minimal container problem; the second is a maximal item problem. Given a collection of items and a particular type of container, a minimal container problem asks for the minimal\(^{19}\) container of that type which contains the given items. Similarly, given a container and a particular collection of item characteristics, a maximal item problem asks for the maximal collection of items which possesses those characteristics and which also fits into the container.

The thesis gives examples of layout problems of each type which can be solved using two-dimensional translational containment for irregular polygons. We give four examples of minimal container problems: 1) strip packing, 2) finding minimal square enclosures for items, 3) finding minimal rectangular enclosures, and 4) finding tight convex and nonconvex enclosures.

From the discussion of related work in layout in Section 1.3.3, we see that solutions to

---

\(^{17}\)The Russian ellipsoid method is not currently regarded as practical, but it was the first polynomial-time algorithm for linear programming.

\(^{18}\)Reeves et al. [Ree93] discuss two types of relaxation methods: 1) linear programming relaxation for integer or mixed-integer programming, which can either be solved heuristically using dual ascent or exactly, and 2) Lagrangean relaxation.

\(^{19}\)One possible minimality criterion is area.
minimal enclosure problems are often required for the first (clustering) stage of a multi-stage layout problem. Minimal container problems also arise when finding the best stock sheet size for a given set of items, or the best cloth width in garment manufacturing. Section 6.3.1 discusses computational geometry work on various types of enclosure problems for single items. Current solutions to minimal enclosure problems for more than two items are rare and heuristic. This work appears in the layout literature (see Section 1.3.3 and Section 6.3.1). The problem is typically solved either incrementally by adding one item at a time [DT87], or by placing small items in gaps inside the bounding box of two large items [AA76].

Our general approach to finding minimal enclosures is to find an optimal value for a parameter via binary search combined with containment. The binary search can be accelerated by using compaction techniques, such as those developed by Li and Milenkovic [Li94, ML, LM93c, LM93a], to help update the upper bound (i.e. to improve a feasible solution found by containment). Our hybrid containment algorithm is appropriate for minimal enclosure problems. It is extremely fast for loose fits, and so the initial stages of the search are fast.

For the problem of finding tight convex and nonconvex enclosures, we show that removing unreachable portions of a tight rectangular enclosure can yield tight convex and nonconvex enclosures which preserve all the containment solutions for the rectangle.

We also give four examples of maximal item problems. For this discussion, let $k_{\text{practical}}$ be the largest number of polygons for which current containment algorithms are practical. Our problems are: 1) for a feasible containment problem, finding maximally scaled copies of items which fit in the container, 2) given a container and a collection of $\leq k_{\text{practical}}$ items for which the sum of the item areas does not exceed the area of the container, finding a “large” sub-collection which fits in the container, 3) given a container and a collection of items for which either the sum of the item areas exceeds the area of the container or the number of items is $> k_{\text{practical}}$, finding a “large” sub-collection of $\leq k_{\text{practical}}$ items which fits in the container, and 4) problem (3) for a collection of containers.

For problem (1) we use nested binary search combined with containment. The key to solving problem (2) is to carefully choose which item to eliminate from an infeasible subset. We reject the most “troublesome” item, which is identified using the results of the overlap elimination evaluation method of the hybrid containment algorithm. For problem (3), we randomly construct a set of items and solve problem (2). This yields a rich collection of sets which fit into the container. For each set of size $n$ which fits, we automatically have $2^n$ subsets which fit and which need not be individually represented. This work provides the first practical method for constructing sets of up to ten polygons which fit in a container.

For problem (4), we solve problem (3) for each container, and then apply a look-ahead assignment strategy. This is a pre-packing strategy. Containment thus allows us to transform a geometric optimization problem into a combinatorial optimization problem. Our results challenge the existing belief that pre-packing methods are impractical for layout [Arb93].

In a multi-stage layout problem, problem (3) might involve a large, highly nonconvex container. Highly nonconvex containers also appear in industries in which defects in material cause the material to be partitioned into “quality zones” (see Heistermann and Lengauer’s work with leather [HL92]). We show how to decompose such a container in a natural fashion, transforming this problem into problem (4). The transformation removes portions of the container which are clearly not reachable by any item. Our decomposition method uses an algorithm we developed for finding the maximum-area axis-parallel rectangle (MAAPR)
in a polygon. Our theoretical work on this problem gives the fastest known asymptotic running time for a MAAPR algorithm for polygons.

1.4 Project Background

The work in this thesis is part of a project whose goal is to automate the marker-making process in the U.S. apparel industry. Marker-making is the process of laying out pattern pieces on cloth so as to minimize cloth waste. As an example, Figure 1.2 depicts a multi-stage marker-making problem from the apparel industry. At the top is a collection of large pants panels and a set of smaller trim pieces. The marker-making task is a strip-packing problem: arrange these items on a semi-infinite strip of material while minimizing the length of the strip.

Our marker-making research has been carried out under the direction of Victor J. Milenkovic, Assistant Professor of Computer Science at Harvard University. The research is part of a larger project, funded by the Sloan Foundation. The principal investigator of this project is Frederick H. Abernathy, Professor of Mechanical Engineering at Harvard University and director of the Harvard Center for Apparel and Textile Research.

Humans generally place the large panels first and then place the trim in the gaps formed by neighboring panels. Thus, they first solve a panel placement problem followed by a trim placement problem. A layout of panels is shown in the figure, followed by the complete layout produced by a human expert. In this example, the items may translate and may also rotate by a discrete set of angles.

The goal of our project is to meet or exceed human performance for the pants marker-making problem. One way to do this is to apply an improvement strategy to an existing layout produced by a human. Another way is to generate a new layout from scratch using, for example, a two-stage approach such as the one adopted by human marker-makers. Accordingly, we initially partitioned our project into three subprojects: 1) compaction, 2) panel placement, and 3) trim placement. Below we briefly describe these three parts of our project.

1.4.1 Compaction

Li and Milenkovic's work on compaction has been a significant part of our project [Li94, ML, LM93c, LM93a]. Given a layout, their algorithms use linear programming to improve the layout through continuous motion of the pieces. A position-based linear programming model is at the heart of the algorithms. The algorithms accomplish a variety of tasks. One task is to reduce the length of a marker. This type of compaction can be used to accelerate binary search in our work on minimal enclosures. Another task is to reduce overlaps among pieces. This is an integral part of the substitution-based approach to pants marker-making described in Section 1.3.3. Thus, compaction can serve both as an improvement method for existing layouts and as the basis of a method for constructing layouts. The overlap reduction evaluation method we apply within our hybrid containment algorithm is an extension, by Milenkovic, of the overlap reduction work.

---

Victor Milenkovic is now an Associate Professor at the University of Miami.
Set of polygonal parts

Strip of fixed width and unknown length

Placement of large pieces

Human layout with 180 degree rotations and xy-flips allowed

Figure 1.2: The marker-making task in the apparel industry
1.4.2 Panel Placement

Section 1.3.3 summarized the work of Milenkovic [MDL92] on placing large items by building column clusters. To place large items, he builds column clusters of four items and forms a grid-like layout consisting of a set of columns. An algorithm based on dynamic programming adds one column at a time to the layout. Backtracking is allowed.

1.4.3 Trim Placement

The trim placement task falls within the two-dimensional bin-packing category, according to Dyckhoff and Finke’s typology [DF92]. One might argue that it also belongs in the cutting stock category because the items can be partitioned into equivalence classes. However, the items in each class are not congruent; they are graded copies of each other. Trim placement is not solved using repeating patterns, and so it does not fit Dyckhoff and Finke’s cutting stock type. However, approaches to trim placement can certainly take advantage of the similarity between pieces in the same equivalence class.

Trim placement can be viewed as a containment problem for which the container is the complement of the placed panels within the original container. The container is highly nonconvex and has multiple components. The number of items far exceeds the number which can currently be handled by any practical containment algorithm. However, practical algorithms for solving the associated packing problem (i.e., place as many pieces as possible) can be developed if we first decompose the container into a collection of containers, where each container’s capacity is appropriate for a containment algorithm.

This trim placement problem provided the original motivation for our work on containment. Most of the containment examples in this thesis are drawn from various marker-making problems in the apparel industry.

Surprisingly, we found that the problem of decomposing a complex container is a challenge in its own right. We therefore discuss container decomposition in a separate chapter, Chapter 7, within the applications part of the thesis.

1.5 Benchmarks

As we have already discussed, our test data comes primarily from apparel manufacturing. We welcome data from other sources. We propose that a set of benchmarks be constructed so that researchers may compare the performance of their translational containment algorithms. We offer to provide data for our containment examples to interested researchers, providing that the data is not used for commercial benefit. To facilitate this exchange while protecting the source of the data, we offer our data using a simple file format, which is given in Appendix A.

1.6 Contribution

The primary contribution of this thesis is our demonstration that, although containment is NP-hard, it is fruitful to: 1) develop algorithms for containment, as opposed to heuristics, 2) design containment algorithms so that they say “no” almost as fast as they say “yes”, 3)
use geometric techniques, not just mathematical programming techniques, and 4) maximize the number of items for which the algorithms are practical.

Our approach to containment has a number of important features. First, we introduce a restrict/evaluate/subdivide paradigm. Second, we develop new theory and practical techniques for the operations within the paradigm. The techniques are appropriate for two-dimensional containment problems in which the items and container may be irregular (i.e., nonconvex) and have multiple components, and in which the items may be translated, but not rotated. Our techniques can be combined to form a variety of two-dimensional translational containment algorithms which follow the restrict/evaluate/subdivide paradigm. The paradigm is designed so that, unlike existing iteration-based algorithms, containment algorithms based on the paradigm are adept at saying “no”, even for slightly infeasible problems. Infeasible problems occur frequently in practice.

We develop geometric restrictions which are powerful configuration space pruning techniques. They often detect infeasibility without any evaluation or subdivision. We give two algorithms which use geometric restrictions and are based on the restrict/evaluate/subdivide paradigm. We obtain the first practical running times for NP-complete two-dimensional translational containment problems for up to ten nonconvex items in a nonconvex container. Our examples are drawn primarily from the apparel industry. Each item is represented as a polygon which typically has from 4 to 100 vertices. The container is also represented as a polygon; it typically has from 100 to 300 vertices.

We demonstrate that viewing containment as a feasibility problem has many benefits for layout optimization problems. We present an effective method for finding minimal enclosures which uses containment to perform binary search on a parameter. We note that compaction can be used to accelerate the search. This work represents the first practical approach to finding minimal enclosures for multiple nonconvex items. Clustering is used often in layout, but current clustering techniques are weak. Although they often allow items to rotate, they typically use greedy heuristics, and therefore cannot provide a lower bound on the size of the container. Our minimal enclosure work should lead to more powerful clustering methods, and is an example of how strong “bottom-up” techniques from computational geometry can help the layout community.

We believe that some potentially promising layout techniques have been abandoned by the layout community because they lack strong bottom-up methods. The primary example of this is pre-packing. We challenge the view that pre-packing is impractical for multiple-container packing problems by generating a rich collection of groups for each container, using containment, and then applying an assignment strategy.

Our advances in containment will help automate layout for industrial processes that rely on two-dimensional packing of objects. Our results are particularly relevant to industries which deal with irregularly shaped items. This includes industries such as aerospace, ship building, apparel and shoe manufacturing, furniture production, and steel construction.

1.7 Overview

The remainder of the thesis is divided into two main parts. Part I describes our work on translational containment for nonconvex items in a nonconvex container. This part contains four chapters. The first chapter introduces our restrict/evaluate/subdivide paradigm for containment and provides the notational and technical background required by
our containment work. The second chapter describes our configuration space restrictions. The third chapter discusses evaluation and subdivision techniques. The fourth chapter presents two different algorithms which can be formed via specific choices of restriction, evaluation, and subdivision techniques. Portions of our containment work also appear in [DMI94, DM, DM95, MD95].

Part II of the thesis gives examples of how containment can be applied to solve various layout problems. This part contains four chapters. The first chapter discusses preliminaries. The second shows how to transform a container so that regions not reachable by any item are eliminated and the remaining reachable regions are decomposed in a natural way. Our container decomposition work also appears in [DM94]. Our theoretical work on finding the maximum-area axis-parallel rectangle in a polygon, which is used for the decomposition, is summarized in that chapter and also appears in [DMR93, DMR]. The third chapter solves four minimal container problems. The fourth solves four maximal item problems.

Conclusions, future work, and an appendix containing a file format for containment examples appear in Part III of the thesis.
Part I

Translational Containment
2.1 Introduction

In this part of the thesis we consider the two-dimensional translational containment problem for polygons. The problem is stated as follows: given a container and a set of items, find translations of the items which result in a non-overlapping arrangement of the items within the container. The items and container may be nonconvex.

Existing two-dimensional translational containment algorithms developed by other researchers are practical only for $k \leq 3$. We present new theory leading to the first practical running times for two-dimensional translational containment problems for up to ten non-convex items in a nonconvex container. For $3 \leq k \leq 10$, one of our algorithms typically runs in several minutes or less on a 50 MHz SPARCstation, and it is therefore practical for solving a number of layout-related problems. Most of our examples are from apparel manufacturing. Usually, each item has from 4 to 100 vertices, and the container has from 100 to 300 vertices.

Although containment is NP-hard in general, we believe that it is valuable to increase the practical value of $k$ as much as possible. Thus, we continue the “bottom-up” progression (1CC, 1NN, 2CN, 2NN, etc.) of containment work which has taken place in robotics and computational geometry. Since containment is NP-hard and all known containment algorithms have running time which is exponential in $k$, an increase in the practical value of $k$ from three to ten is significant.

Our ability to say “no” well comes primarily from the use of our new geometric techniques. Our ability to say “yes” for so many items is partly due to our geometric techniques and partly due to our application of and improvements to existing linear programming-based techniques. Thus, we demonstrate that a computational geometry approach can contribute significantly to practical algorithms for NP-hard geometric problems.

Saying “no” quickly is important. Current translational containment algorithms are iteration-based. Iteration-based algorithms for containment attain their worst-case running time when the containment problem is infeasible. This situation occurs frequently in practice. We replace iteration with subdivision. Our subdivision-based algorithms are specifically designed to detect infeasibility quickly.
We present a new paradigm for containment algorithms. The paradigm is based on operations on two-dimensional configuration spaces. As discussed in Section 1.2.1, each of these spaces represents the set of relative positions for a pair of polygons. The algorithm applies three types of operations, in order, to the configuration spaces: 1) restriction, which prunes the spaces and helps detect infeasibility, 2) evaluation, which tries to find a solution, and 3) subdivision. Subdivision partitions one of the configuration spaces into two parts. The algorithm then recurses on each part while keeping the other configuration spaces constant.

We introduce new theory and techniques for each of these types of operations for the case of two-dimensional containment problems in which the items and container may be irregular (i.e., nonconvex) and have multiple components, and the items may be translated, but not rotated. Our techniques can be combined to form a variety of two-dimensional translational containment algorithms which follow the restrict/evaluate/subdivide paradigm. The paradigm is designed so that, unlike existing iteration-based algorithms, containment algorithms based on the paradigm are adept at saying "no", even for slightly infeasible problems.

At present there appear to be two approaches to containment within the restrict/evaluate/subdivide framework: direct and indirect. Direct methods act on the configuration spaces directly, and these methods tend to be based on computational geometry. Indirect methods act on convex approximations to the configuration spaces: either inner or outer approximations. Indirect methods use linear programming or other types of mathematical programming, which is why they can only act on convex approximations. Direct methods have a more accurate view of the "world", but they can only operate on a few configuration spaces at a time (three, in our case). Indirect methods can act on all the configuration spaces simultaneously in a single operation, but they use only approximations to the configuration spaces.

This thesis presents two containment algorithms which follow our paradigm. The first operates directly on the configuration spaces and is therefore a direct algorithm. We call the second a hybrid algorithm because it uses an indirect evaluation method, and both direct and indirect methods for restriction. Our direct algorithm is also an approximate algorithm because it produces a configuration whose overlap is below a given tolerance. Ironically, our hybrid algorithm, whose indirect evaluation method operates on approximations of the configuration spaces, is an exact algorithm because it produces a configuration with no overlap. We briefly discuss each algorithm individually below.

**Direct Algorithm**

Our new direct algorithm for containment is an approximate algorithm which produces a solution whose overlap is below a given tolerance. As mentioned in Section 1.2.1, for some cutting applications, CAD vendors round polygon vertex coordinates to integers. In such cases, an overlap of less than one unit, corresponding to .01 inches, is acceptable. For the applications we consider, the ratio of the largest value to the smallest nonzero value is usually under 10,000; this makes an approximate algorithm a practical choice. Our approximate algorithm uses our new direct techniques for restriction, evaluation, and subdivision. The algorithm appears in [DM95, DM] as well as in Section 5.2.

Our direct restrictions are based on polygon set operations, and so we call them geometric restrictions. They are powerful tools for detecting infeasibility. We define two general types of restrictions: valid restrictions, which preserve all solutions to a containment problem, and
semi-valid restrictions, which preserve a single solution, if one exists. We present a variety of restrictions and several general properties of valid restrictions. We examine, in detail, the effectiveness of our geometric restrictions, which are valid restrictions. Our characterization of the types of containment problems for which these restrictions are effective is based on theoretical as well as experimental results.

The evaluation method uses restriction together with a geometric method for maximizing the intersection of translations of the configuration spaces. The evaluation method often finds a containment solution without any subdivision. The subdivision method is a size-based one. Its goal is to partition a configuration space into two parts such that each has less area than the configuration space. Our approximate algorithm has an extra layer of structure on top of the restrict/evaluate/subdivide paradigm. This layer has a set of containment subproblems to which coarse-grained parallelism could be applied in the future. We create the subproblems by characterizing a valid configuration for translational containment.

The current implementation of this algorithm is fast for up to four items. For \( k > 4 \), the current implementation of our geometric restrictions, based on polygon set operations, slows the algorithm down considerably. Increasing the speed of our polygon set operations is a subject of future work (see Section 10.2.1). In our experiments, our approximate algorithm performs fewer subdivisions than the faster, hybrid algorithm, and so we believe the approximate algorithm has the potential to be competitive with the hybrid algorithm.

Hybrid Algorithm

In addition to our direct algorithm, we also present significant improvements to the restriction, evaluation, and subdivision techniques of an existing indirect algorithm of Milenkovic [Mil]. Milenkovic’s indirect algorithm, in its original form, was slower than our direct algorithm for infeasible problems, and its practical upper bound on \( k \) was \( k \leq 3 \). The improved indirect algorithm is currently our fastest containment algorithm for \( 4 \leq k < 10 \).

Our first improvement to the indirect algorithm is a proof that restrictions, when used in the indirect algorithm, do not increase the algorithm’s worst-case running time if the algorithm bases certain decisions on the unrestricted configuration spaces. This is important because restrictions can add vertices to the configuration spaces, and so they might have increased the worst-case running time. The proof allows us to add our new direct (geometric) restrictions to the algorithm. It also allows us to apply a new restriction method, developed by Milenkovic, which is based on linear programming.

The second improvement to the indirect algorithm is related to the evaluation method. We apply the work of Li and Milenkovic on overlap resolution [Li94, ML, LM93c, LM93a] to improve the evaluation step. Their overlap resolution algorithm is an indirect algorithm. It uses inner convex approximations, an approach which complements the outer approximations of the indirect containment algorithm. Our third improvement is our new practical subdivision technique which attempts to maximize the distance from the current evaluation to the convex approximations of the subdivided configuration spaces.

We call the resulting algorithm a hybrid algorithm because it uses indirect methods for evaluation, developed by Li and Milenkovic, and both direct and indirect methods for restriction. The hybrid algorithm is described in [MD95] as well as in Section 5.3.

The hybrid algorithm, in addition to being a formidable containment algorithm, has the property that a partial containment solution from it is valuable. This is because each
evaluation step produces either a non-overlapping solution or an overlapping configuration for which a linear program has reached a local minimum. As a result, the hybrid algorithm, when terminated early, is the most powerful heuristic known for generating a local overlap minimum.

2.1.1 Overview of Chapter

In this chapter, we first give notation and technical background in Section 2.2. Section 2.3 establishes the NP-completeness of translational containment. Section 2.4 describes our restrict/evaluate/subdivide paradigm. Section 2.5 introduces the concept of size analysis. Section 2.6 supplements the discussion of related work in containment which appears in Section 1.2.1. Section 2.7 presents a characterization of translational containment problems. Section 2.8 outlines the contents of the remaining chapters in this part of the thesis.

2.2 Notation

The goal of translational containment is to translate \( k \) polygonal regions \( P_1, P_2, \ldots, P_k \) into a polygonal container without overlap. If we denote by \( P_0 \) the complement of the container region \( C \), then containment is equivalent to the placement of \( k + 1 \) polygons \( P_0, P_1, P_2, \ldots, P_k \) in non-overlapping positions.

**Definition 2.1 ([Min03, GRS83, Ser82, Ser88])** The Minkowski sum of two point-sets (of \( \mathbb{R}^2 \) in our case) is:

\[
A \oplus B = \{ a + b | a \in A, b \in B \} = \bigcup_{b \in B} (A + b).
\]

The Minkowski difference is:

\[
A \ominus B = \bigcap_{b \in B} (A - b).
\]

For a point-set \( A \), let \( \overline{A} \) denote the set complement of \( A \) and define \( -A = \{ -a | a \in A \} \). For a vector \( t \), define \( A + t = \{ a + t | a \in A \} \) and \( A - t = \{ a - t | a \in A \} \). Note that \( A + t = A \oplus \{ t \} \).

**Theorem 2.2.1 ([Ser82])** \( A \ominus B = \overline{A \oplus B} \).

Theorem 2.2.1 states that the Minkowski sum and difference are duals with respect to complementation.

**Theorem 2.2.2 ([GRS83, Ser82])** \( (B + t) \cap A \) if and only if \( t \in A \oplus -B \neq \emptyset \).

Figure 2.1 illustrates Theorem 2.2.2. Readers familiar with mathematical morphology will recognize this as the “hit” transformation [Ser82]. There is an analogous “miss” transformation involving \( \ominus \), which is illustrated in Figure 2.2 and stated in Theorem 2.2.3.

**Theorem 2.2.3 ([Ser82])** \( (B + t) \subseteq A \) if and only if \( t \in A \ominus -B \neq \emptyset \).
Figure 2.1: \((B + t) \cap A\) if and only if \(t \in A \oplus -B\)

Figure 2.2: \((B + t) \subseteq A\) if and only if \(t \in A \ominus -B\)
Definition 2.2 ([Ser82]) The opening of $A$ with respect to $B$ is $(A \oplus -B) \oplus B$.

If $B$ fits into $A$, then the opening of $A$ with respect to $B$ consists of all points of $A$ which are reachable by some part of $B$ via translation.

The symbol $U_{ij}$, $0 \leq i, j \leq k$, $i \neq j$, denotes the set of displacements from $P_i$ to $P_j$ such that they do not overlap each other. For translations $t_i$ and $t_j$, $P_i + t_i$ and $P_j + t_j$ overlap if and only if $t_j - t_i \in U_{ij}$. It follows that $U_{ij} = -U_{ji}$. Since $P_i$ cannot be displaced with respect to itself, we define $U_{ii} = \{(0,0)\}$. Expressed in terms of set operations and Minkowski sums,

$$U_{ij} = P_i \oplus -P_j, \quad 0 \leq i, j \leq k, \ i \neq j. \quad (2.1)$$

Equation 2.1 follows directly from Theorem 2.2.2. The set $U_{ij}$ is the two-dimensional free configuration space for placing $P_j$ with respect to $P_i$. Note that $U_{ij}$ is unbounded unless $i = 0$ or $j = 0$, assuming that the container and $P_1, \ldots, P_k$ are bounded.

Let $\mathcal{P}$ and $\mathcal{U}$ denote the set of all $P_i$ and $U_{ij}$, respectively. (Strictly speaking, these are lists; even if two elements, such as $U_{27}$ and $U_{36}$, are congruent, they are maintained as separate elements.)

Definition 2.3 A configuration of $\mathcal{P}$ is a $(k + 1)$-tuple $(t_0, t_1, \ldots, t_k)$ where $t_i$ is a translation $(x_i, y_i)$ of $P_i$ and $t_0$ is arbitrarily set to $(0,0)$. A valid configuration of $\mathcal{P}$ is a configuration of $\mathcal{P}$ which satisfies

$$t_j - t_i \in U_{ij}, \quad 0 \leq i < j \leq k. \quad (2.2)$$

A valid configuration is an exact solution to a translational containment problem. It corresponds to a point in $2k$-dimensional space. The set of points corresponding to all valid configurations for a given $\mathcal{P}$ is the $2k$-dimensional free configuration space for $\mathcal{P}$.

Definition 2.4 An $\epsilon$-approximate configuration of $\mathcal{P}$ is a configuration such that the distance of $t_j - t_i$ from $U_{ij}$ is $\leq 2\epsilon$ for $0 \leq i < j \leq k$.

In an $\epsilon$-approximate configuration, no point of any polygon in $\mathcal{P}$ is more than $2\epsilon$ inside of the boundary of any other polygon when the polygons are laid out according to the configuration.

Definition 2.5 Given $U_{ij} \in \mathcal{U}$ for an instance of translational containment, $U_{ij}^*$ is the set of all values of $t_j - t_i$ such that $t_i$ and $t_j$ are translations of $P_i$ and $P_j$, respectively, in a valid configuration of $\mathcal{P}$.

In other words, $U_{ij}^*$ is the projection of the set of all $2k$-dimensional valid configurations into $U_{ij}$. Clearly $U_{ij}^* \subseteq U_{ij}$. Also, $U_{ij} = -U_{ji}$ implies $U_{ij}^* = -U_{ji}^*$. Let $\mathcal{U}^*$ denote the set of all $U_{ij}^*$.

In Section 1.2.1, we defined the $k$NN problem, the $(k, r)$NN problem, and their variations (see Definition 1.1 and Definition 1.2). We refer to the entire collection of two-dimensional translational containment problems (including, $k$NN, $kCN$, etc.) as translational containment problems. Note that we do not require the set of all valid configurations; a single valid configuration suffices. Each problem has an approximate version, which is to find an $\epsilon$-approximate configuration. Each containment problem is an off-line problem because the entire set $\mathcal{P}$ is available at the start of the containment problem.
Let the number of vertices of $P_i$ be $m_i$. We define $m = \max_{1 \leq i \leq k} m_i$. We use $m_0$ frequently, and it tends to be larger than the other $m_i$. Hence, we distinguish $m_0$ by denoting it by $n$. In general, we express the running time of a containment algorithm in terms of $m$, $n$, $k$, and $r$.

This is not the most general containment problem, since it is restricted to two dimensions, does not allow rotation, and the representation of items is limited to polygons. However, this problem is still of great practical importance. Industries dealing with two-dimensional polygonal items include aerospace, ship building, apparel, shoe and furniture manufacturing, and steel construction. In applications such as apparel, shoe, and furniture manufacturing, only a discrete set of rotations is typically allowed; this can be dealt with by iterating over the angles. In contrast, metal pieces are often represented using curves as boundaries, and continuous rotation is frequently allowed.

### 2.3 Hardness

The main result of this section is that $k$NN is NP-complete. Before proving this, we establish two lemmas. The first shows that $k$RR, a containment problem for rectangular items in a rectangular container, is NP-hard.

**Lemma 2.3.1** The $k$RR problem is NP-hard.

**Proof:** Li [Li94] examines the two-dimensional coordinated motion planning problem of *strip compaction* in which the motion of the polygons is restricted to a strip of fixed width and arbitrary length, and the goal is to minimize the length of the strip. He shows this problem is NP-hard when all the polygons are rectangles and only translations are allowed. He reduces the NP-complete PARTITION problem to strip compaction. We use a similar reduction here to establish that $k$RR is NP-hard. In an instance of PARTITION [GJ79], we are given a finite set $A$ and a size $s(a) \in Z^+$ for each $a \in A$. We ask: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$? We reduce the NP-complete PARTITION problem to $k$RR. For each $a \in A$ we construct a rectangle of unit height and width $s(a)$. We also construct a container of height 2 and width $(\sum_{a \in A} s(a))/2$. PARTITION has a solution if and only if there is a valid configuration of the rectangular items inside the rectangular container. This establishes that $k$RR is NP-hard. \hfill $\Box$

**Lemma 2.3.2** Two-dimensional translational containment is in NP.

**Proof:** Milenkovic, et al. [MDI91] show that the marker-making problem, in the absence of rotations, is in NP. The main idea of their argument applies to translational containment. They show that, if a solution exists, then there exists a (possibly different) solution which is uniquely determined by its set of vertex-edge contacts. We observe that there are only $2k$ degrees of freedom. Therefore, we can (nondeterministically) select the correct $2k$ vertex-edge contacts, solve the set of linear equations, and verify the solution. This is a polynomial-time process because the linear system has a polynomial number of equations, and a polynomial number of linear equations can be solved in time which is polynomial in the number of equations. \hfill $\Box$

**Theorem 2.3.3** Two-dimensional translational containment is NP-complete.
Proof: Lemma 2.3.1 shows that \( k \text{RR} \) is NP-hard. Since all the translational containment problems are harder than \( k \text{RR} \), this proves that translational containment is NP-hard. Lemma 2.3.2 shows that translational containment is in NP, which completes the proof. □

2.4 The Restrict/Evaluate/Subdivide Paradigm

The set \( \mathcal{U} \) encodes the entire containment problem, and one can think of it as the input to and current state of a containment algorithm. We refer to \( \mathcal{U} \) as the hypothesized solution space or hypothesis. A solution within the current hypothesis consists of a valid configuration for the current set \( \mathcal{U} \).

Our paradigm uses three operations on the hypothesis:

Restriction: prune the hypothesis by applying restrictions to the current set \( \mathcal{U} \). A restriction is an operation on \( \mathcal{U} \) which replaces some or all \( U_{ij} \)'s by subsets of themselves. Naturally, if any \( U_{ij} \in \mathcal{U} \) is replaced by the empty set, then there is no solution.

Evaluation: try to find a solution within the given hypothesis. The result is that either:
1) yes, a valid configuration exists within the hypothesis, 2) no, a valid configuration does not exist within the hypothesis, or 3) maybe a valid configuration exists, but it has not yet been found.

Subdivision: divide the current hypothesis into two sub-hypotheses such that if a solution exists within the current hypothesis, then a solution must exist within one of the two sub-hypotheses.

The pseudo-code below describes the flow of a general containment algorithm which abides by our paradigm. We present it in the form of a decision procedure which returns YES or NO. Of course, if a valid configuration is found, the configuration can be stored in a global data structure. The routines RESTRICT(), EVALUATE(), and SUBDIVIDE() perform the functions described in the list of operations given above. RESTRICT() returns a restriction of \( \mathcal{U} \). EVALUATE() returns a status flag indicating whether it decides YES, NO, or MAYBE. SUBDIVIDE() partitions one of the configuration spaces \( U_{ij} \in \mathcal{U} \) into two parts \( U_{ij}' \) and \( U_{ij}'' \). The algorithm then recurses on each part while keeping the other configuration spaces constant. That is, it creates \( \mathcal{U}' = (\mathcal{U} \setminus \{U_{ij}\}) \cup \{U_{ij}'\} \) and \( \mathcal{U}'' = (\mathcal{U} \setminus \{U_{ij}\}) \cup \{U_{ij}''\} \) and then recurses on \( \mathcal{U}' \) and \( \mathcal{U}'' \).

\[
\text{CONTAIN}(\mathcal{U}, k)
\begin{align*}
\mathcal{U} & \leftarrow \text{RESTRICT}(\mathcal{U}, k) \\
\text{if (some } U_{ij} \in \mathcal{U} = \emptyset \text{) return NO} \\
\text{status} & \leftarrow \text{EVALUATE}(\mathcal{U}, k) \\
\text{if (status = YES) return YES} \\
\text{if (status = NO) return NO} \\
\mathcal{U}', \mathcal{U}'' & \leftarrow \text{SUBDIVIDE}(\mathcal{U}, k) \\
\text{return CONTAIN}(\mathcal{U}', k) \text{ or CONTAIN}(\mathcal{U}'', k)
\end{align*}
\]

A containment algorithm can be designed by selecting a restriction method, an evaluation method, and a subdivision method. The correctness of the algorithm depends on the
correctness of the chosen methods. Termination of the algorithm depends on the compatibility of the methods. For example, suppose we have an algorithm which has a certain worst-case running time based on a subdivision method for which the number of reflex vertices of $U_{ij}^1$ and $U_{ij}^0$ is less than the number of reflex vertices of $U_{ij}$. Now, suppose we add restrictions to this algorithm. The restriction process may increase the number of reflex vertices in the hypothesis. This can increase the worst-case running time of the algorithm.

A mismatch between evaluation and subdivision methods can also be problematic. For instance, consider an algorithm which uses the subdivision technique mentioned above for reducing the number of reflex vertices, together with a size-based evaluator. That is, the evaluator says “no” if it cannot find a valid configuration in $U$ and the size\(^1\) of $U$ is below a tolerance. Subdivision will eventually produce hypotheses containing convex polygons, but those polygons will not necessarily have size less than the tolerance. If a valid configuration exists within $U$ in this case, the algorithm will not terminate unless the evaluator happens to find it.

Course-grained parallelism can be applied to this paradigm, and is a subject for future work. In order to parallelize the general containment algorithm, one would maintain a queue of hypotheses instead of performing recursion. Each processor would remove a hypothesis from the queue and restrict it and evaluate it. If no solution were found, the hypothesis would be subdivided and the two resulting sub-hypotheses would be added to the queue. Each hypothesis represents a “self-contained” containment problem, and so processors could work on hypotheses independently. Of course, the enqueueing and dequeueing of hypotheses should be implemented as atomic operations.

One limitation of this approach is that the subproblems would not necessarily all be available at the start of the algorithm. Some processors might remain idle until a sufficient number of hypotheses were created.

### 2.5 Approach to Running Time Analysis

In order to analyze the worst-case time complexity of a containment algorithm which follows our paradigm, one must know the time complexity of RESTRICT($\cdot$), EVALUATE($\cdot$), and SUBDIVIDE($\cdot$). In addition, one needs an upper bound on the number of times CONTAIN($\cdot$) is invoked; this is determined by the conditions for which EVALUATE($\cdot$) decides “no”. Finally, an upper bound on the total number of vertices of $\mathcal{U}$ is required.

Calculating the total number of vertices of $\mathcal{U}$ is complicated by the possibility that RESTRICT($\cdot$) and SUBDIVIDE($\cdot$) may introduce new vertices into $\mathcal{U}$, in spite of the fact that each replaces some $U_{ij}$s by subsets of themselves. One way to treat this is to estimate the worst-case upper bound on the number of new vertices. However, this upper bound may bear little resemblance to the actual worst-case size which is attainable in practice.

For example, suppose that RESTRICT($\cdot$) and SUBDIVIDE($\cdot$) use only polygon set operations such as intersection and union. In theory, for polygons $A$ and $B$, $|A \cup B|, |A \cap B| \in \mathcal{O}(|A| \cdot |B|)$, but, in practice, $|A \cup B|, |A \cap B| \approx |A| + |B|$. On the one hand, new vertices are created by edge-edge intersections, but, on the other hand, many vertices are discarded when the union or intersection boundary is computed.

We introduce the concept of size analysis in order to capture the practical running time of a containment algorithm which follows our paradigm. Let $s$ be the maximum number \(^1\)One measure of the “size” of $\mathcal{U}$ is the total area of all $U_{ij}$ in $\mathcal{U}$.
of vertices produced by a possibly infinite sequence of polygon operations which replaces \( \mathcal{U} \) by a subset of \( \mathcal{U} \). (In Chapter 3 we are more specific about the operations.) In some cases, one can prove that \( s \) is finite. Although \( s \) can be exponential in the original size of \( \mathcal{U} \), in practice, it is usually not more than quadratic. Since our analysis is based on \( s \), we refer to size analysis as \( s \)-analysis in the remainder of the thesis.

2.6 Related Work

In Section 1.2.1 we reviewed the literature on containment. Here we fill in some of the technical details omitted by that discussion.

A “naive” \( k \)NN algorithm requires \( O(k^2(mn)^2 \log mn) \) time. It searches for a valid configuration by iterating over all choices of \( 2k \) contacting pairs among the polygons and container. There are \( O((kmn + k^2m^2)^2k) = O((mn)^2k) \) possible choices of \( 2k \) vertex-edge contacts. For each choice we can solve a set of \( 2k \) linear equations in \( O(k^3) \) time (the cost of Gaussian elimination) to obtain \( t_i, 1 \leq i \leq k \). This configuration can be checked for overlap in \( O(\log n) \) time, assuming the following preprocessing. Precompute \( \mathcal{U} \) according to Equation 2.1. The number of vertices in each of these polygons has an upper bound of \( O(m^2n^2) \). Preprocess the polygons in \( \mathcal{U} \) so that a point-in-polygon test can be performed in \( O(\log mn) \) time [Kir83]. To check a layout, perform a point-in-polygon test for each \( t_j - t_i \) in \( U_{ij} \). This costs \( O(k^2 \log mn) \) time. The naive algorithm has running time in \( \Omega((mn)^{2k}) \) because it must iterate over all possible sets of contacts whenever the answer is “no.”

The following running times were given in Section 1.2.1.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( k )CN</th>
<th>( k )NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( O(mn \log mn) ) [For85]</td>
<td>( O(m^2n^2 \log mn) ) [AB87, Avn89]</td>
</tr>
<tr>
<td>2</td>
<td>( O(m^2n^2 \log m) ) [Dev90]</td>
<td>( O(m^4n^4 \log mn) ) [AB87, Avn89]</td>
</tr>
<tr>
<td>3</td>
<td>( O(m^3n^3 \log m) ) [Dev90]</td>
<td>( O(m^{14}n^6 \log mn) ) [Avn89]</td>
</tr>
</tbody>
</table>

Avnaim and Boissonnat also give a solution to the 3NP problem, three nonconvex polygons in a parallelogram container, using time in \( O(m^{60} \log m) \) [AB87, Avn89]. Even though the asymptotic running time of the 3NP algorithm contains a polynomial whose degree is high compared to the others, the 3NP algorithm is actually more practical than some of the other lower degree algorithms. Avnaim’s 3NN algorithm, in particular, iterates over all choices of edges from \( U_{01}, U_{02}, U_{03}, U_{12} \) and \( U_{13} \). (Avnaim uses a sweep-line algorithm to avoid iterating over the edges of \( U_{23} \).) The worst-case running time of the 3NN algorithm is therefore proportional to the product of the sizes of these sets. To determine that a configuration is impossible, the algorithm must iterate over all these choices. Its \( O(m^{14}n^6 \log mn) \) running time is worse than the \( \Omega((mn)^6 \log n) \) running time for the naive algorithm for \( k = 3 \). For reasons outlined below, it is faster than the naive algorithm in practice.

The \( m^{60} \) factor for 3NP arises from the worst case bound on the number of vertices of the outputs of several consecutive Minkowski sums and polygon unions/intersections. In practice, the worst case is usually a gross overestimate, and unlike the \( \Omega(m^{14}n^6) \) lower bound for the 3NN algorithm, the factor of \( m^{60} \) may not be attainable. At each step, the running time of the 3NP algorithm is sensitive to the output of the previous step, and so the algorithm takes advantage of the fact that the number of vertices almost certainly grows

\(^2\)See Section 2.7 for a description of the 3NN characterization which allows this iteration.
much more slowly than the estimate. Recall our discussion of $s$-analysis in Section 2.5. An observation similar to the one made about polygon intersection and union holds for the Minkowski sum. Using $s$-analysis, the 3NN algorithm has running time in $O(s^5 \log s)$, whereas the 3NP algorithm has running time in $O(s \log s)$. The naive algorithm has running time in $O(s^{4k} \log s)$; for $k = 3$ this is $O(s^{12} \log s)$. With this analysis, 3NP is faster than 3NN, and both are faster than the naive algorithm.

2.7 Containment Characterization

Here we take advantage of the fact that the goal of our containment problem is to find a single valid configuration as opposed to finding the entire set of valid configurations. We characterize a containment solution by showing that if a valid configuration $\tau$ exists for a given containment problem, then a valid configuration $\tau'$ must exist, where $\tau'$ can be reached from $\tau$ by a non-overlapping motion of the pieces, and $\tau'$ possesses characteristics which facilitate the search for a solution. In particular, $\tau'$ is a vertex of a component of the free configuration space, and this component contains $\tau$. In this case we can solve the containment problem by finding $\tau'$. This type of approach has been used for containment by Chazelle to solve 1NN with rotation [Cha83], by Avnaim to solve 3NN with translation only [Avn89], and by Milenkovic, et al. to solve 3CN [DML94].

Our characterization of a solution to $k$NN stems from a small number of simple observations about contacts and degrees of freedom. First, we observe that because $k$NN allows translation in $x$ and $y$, each polygon has two degrees of freedom. Second, the collection of $k$ polygons has $2k$ degrees of freedom.

There are two types of contacts. The first type involves a vertex and an edge. A vertex-edge contact is a single contact because it implies a single linear constraint. The second type involves two vertices. A vertex-vertex contact is a type of double contact because it implies two linear constraints. A linear constraint reduces the number of degrees of freedom by one if and only if it is linearly independent of the existing system of constraints.

In terms of $U$, a single contact of $P_i$ with $P_j$ means $t_j - t_i$ is on the boundary of $U_{ij}$ and a double contact means $t_j - t_i$ is at a vertex of $U_{ij}$.

In our characterization below, let $I$ be an instance of $k$NN.

**Theorem 2.7.1** If $I$ has a valid configuration, then $I$ has a valid configuration with the following characteristics:

- $\geq 2k$ contacts;
- $\geq 2$ contacts per polygon $P_i$, $0 \leq i \leq k$;

**Proof:** Assume $I$ has a valid configuration. We first show that there is a valid configuration with $\geq 2k$ contacts by arguing as in [MDL91]. Less than $2k$ linear constraints cannot determine the value of $2k$ variables, and therefore there must be at least one degree of freedom. There must therefore be some simultaneous motion of the polygons $P_i$, $1 \leq i \leq k$, for which at least some subset of the polygons move along lines while all polygons retain their contacts. If some subset of the polygons are moving along lines, then eventually some new contact must form. One of three events must occur: 1) two polygons collide, neither of which is $P_0$, or 2) two polygons collide, one of which is $P_0$, or 3) a new contact occurs.
between two already contacting polygons. The occurrence of one of these events is guaranteed by the fact that the complement of \( P_0 \) is bounded (recall that \( \overline{P_0} \) is the container). If two polygons \( P_i \) and \( P_j \) collide, then a new vertex-edge contact is formed. If \( P_i \) and \( P_j \) already have a vertex-edge contact, then that vertex may slide to the end of the edge and form a vertex-vertex contact. We stop the motion when a new contact occurs. Repeating this process forms a set of \( 2k \) contacts.

To establish the second part of the claim, we observe that if a valid configuration for \( \mathcal{I} \) exists and a polygon has fewer than two contacts, then we can translate that polygon until it loses both degrees of freedom. “Translating” \( P_0 \) is equivalent to fixing \( P_0 \) and translating the remaining \( P_i \) by the same amount in the opposite direction. This implies a valid configuration in which each polygon has at least two contacts.

A containment problem can be converted to a set of independent containment subproblems using Theorem 2.7.1. A collection of independent subproblems can be solved in parallel if sufficient processors are available. In the special case where some of the \( P_i \) are identical, equality of some of the \( U_{ij} \) reduces the number of subproblems.

The cost-effectiveness of iterating over a particular set of subproblems should be carefully evaluated when designing a containment algorithm. Let “contacting pair” refer to a pair of polygons which is in contact. The characterization tells us which contacting pairs can occur. The number of possible contacting pairs is purely a function of \( k \), not \( m \) or \( n \). Testing the characterization requires solving a subproblem for each possible contacting pair. If the pair is in single contact, we solve a boundary-restricted subproblem by replacing \( U_{ij} \) by its boundary. If the pair is in double contact, we solve a vertex-restricted subproblem by replacing \( U_{ij} \) by its vertices. Since \( k \) is a “constant”, the characterization does not increase the asymptotic running time in terms of \( m \) or \( n \). Nevertheless, the number of ways to choose contacting pairs is exponential in \( k \), roughly \( k^{2k} \).

For \( k = 3 \) the implication of the first part of the characterization (\( \geq 2k \) contacts) is particularly nice. If there exists a solution to 3NN, then there exists a solution in which a set of \( 2k = 6 \) determining contacts contains either a double contact or all single contacts. Testing for a double contact solution requires solving a vertex-restricted subproblem for each \( U_{ij} \). For a solution where six pairs of polygons are in contact, each \( (P_i, P_j) \) pair must be in contact. This implies a subproblem in which every polygon in \( \mathcal{U} \) is boundary-restricted. Avnaim [Avn89] takes advantage of this to eliminate a factor of \( m^4 \) from the running time of his 3NN algorithm.

Unfortunately, for \( k > 3 \) we must choose \( 2k \) out of \( \binom{k+1}{2} \) possible contacting pairs. The number of ways to choose \( 2k \) pairs is exponential in \( k \). Both a generalization of Avnaim’s 3NN algorithm and the naive algorithm must iterate over all ways of choosing \( 2k \) single contacts. Using the second part of the characterization (\( \geq 2 \) contacts per polygon) also generates an exponential number of subproblems if we apply it to every polygon. Applying it to only a single polygon \( P_i \) requires that we check two cases:

1. a double contact with at least one \( P_j, 0 \leq j \leq k, i \neq j \), or
2. a single contact with at least one \( P_j, 0 \leq j \leq k, i \neq j \) and a single contact with at least one \( P_{j'}, 0 \leq j' \leq k, i \neq j \neq j' \).

Testing the first case requires \( k \) vertex-restricted subproblems (one for each \( U_{ij}, 0 \leq j \leq k, i \neq j \)). Each subproblem is vertex-restricted for \( U_{ij} \). The second case requires
(k)(k − 1)/2 subproblems. Each subproblem is boundary-restricted for two $U_{ij}$s. This approach therefore requires solving $O(k^2)$ subproblems.

Applying part of this characterization leads to a semi-valid configuration space restriction in Chapter 3. Partial characterization is also used in Chapter 5 for our approximate containment algorithm.

2.8 Overview of Part One

The remainder of the containment part of the thesis contains three chapters. Chapter 3 describes our configuration space restrictions. Chapter 4 discusses evaluation and subdivision techniques. Chapter 5 gives two different algorithms which can be formed via specific choices of restriction, evaluation, and subdivision techniques. The first of the two algorithms is an approximate and direct algorithm; the second is our exact, hybrid algorithm. Portions of our containment work also appear in [DML94, DM, DM95, MD95].

\footnote{A semi-valid restriction is guaranteed to preserve at least one valid configuration, if one exists.}
Chapter 3

Configuration Space Restrictions

3.1 Introduction

Here we show how to restrict certain two-dimensional configuration spaces associated with translational containment. Recall from Section 2.4 that a restriction of \( U \) replaces some or all \( U_{ij} \)s by subsets of themselves. Naturally, if any \( U_{ij} \in U \) is replaced by the empty set, then there is no solution for the hypothesis \( U \). We define two general types of restrictions: valid restrictions, which preserve all solutions to a containment problem, and semi-valid restrictions, which preserve a single solution, if one exists. Restrictions can prune away invalid regions from the configuration spaces before a containment algorithm is applied. They can help an algorithm quickly detect infeasibility. Restrictions can also be integrated into the body of an existing containment algorithm.

This chapter makes several general observations about restrictions of configuration spaces for translational containment. We present a variety of valid and semi-valid restrictions. We examine, in detail, the effectiveness of our geometric restrictions, which are valid restrictions. Our characterization of the types of containment problems for which these restrictions are effective is based on theoretical as well as experimental results. Most of our restrictions are based on closed\(^1\), rather than regularized, set operations, and so we discuss this important implementation issue.

3.1.1 Overview

Section 3.1.2 provides notation for this chapter. Section 3.2 makes several general observations about valid restrictions. The next three sections present valid restrictions. In Section 3.3 we present our geometric restriction, give an example, and discuss the effectiveness of this restriction. In Section 3.4 we describe a linear programming restriction, give an example, and discuss the effectiveness of this restriction. Section 3.5 gives a restriction which is valid when one item is a subset of another. Section 3.6 gives a parallelizable, valid restriction based on unions. Section 3.7 gives a parallelizable restriction based on the characterization in Section 2.7. Section 3.8 gives a restriction for identical items. Section 3.9

---

\(^1\)Regularized operations replace each polygon by the closure of its interior. Our closed operations simply replace each polygon by its closure. See Section 3.9.
discusses closed operators, on which many of these restrictions are based. Section 3.10 summarizes this chapter and presents conclusions.

Note: Figures depicting $U_{ij}$ polygons appear throughout this chapter to illustrate the difference between $U_{ij}$s before and after a restriction is applied to $U$. A $U_{ij}$, for $1 \leq i < j \leq k$, is unbounded in its original form, as defined by Equation 2.1, so we perform the following bounding operation before displaying a $U_{ij}$:

$$U_{ij} \leftarrow U_{ij} \cap (U_{i0} \oplus U_{0j})$$  \hspace{1cm} (3.1)

This operation is a valid restriction, as established in Section 3.3.

### 3.1.2 Notation

**Definition 3.1** A restriction is an operation on $U$ that replaces some or all $U_{ij} \in U$ by subsets of themselves. A valid restriction preserves the invariant $U_{ij}^* \subseteq U_{ij}$.

Recall from Section 2.2 that $U_{ij}^*$ is the set of all values of $t_j - t_i$ which belong to a valid configuration. If we denote by $U_{ij}^*$ the replacement of $U_{ij}$ in a valid restriction of $U$, then $U_{ij}^* \subseteq U_{ij}^* \subseteq U_{ij}$. We also denote the set of all $U_{ij}^*$ for a particular valid restriction by $U^*$.

**Definition 3.2** A semi-valid restriction of $U$ is a restriction of $U$ having the property that if there exists a valid configuration in $U$, then there exists a valid configuration in the restriction of $U$.

When it is important to refer to an original $U_{ij}$, as defined by Equation 2.1, we use the notation $U_{ij}^{\text{orig}}$, and $U^{\text{orig}}$ for the set of all $U_{ij}^{\text{orig}}$.

### 3.2 Properties of Valid Restrictions

Here we discuss several general properties of any valid restriction for translational containment.

#### 3.2.1 Boundary Intersection

Consider the following two restriction examples$^2$. The first example, shown in Figure 3.1, is of four nearly rectangular items to be placed in a nearly convex container. Figure 3.2 shows two one-component $U_{0j}$ polygons for this example, with $U_{0j}^*$ polygons shaded in black. Figure 3.3 shows two two-component $U_{ij}$ polygons with $U_{ij}^*$ polygons.

The second example, shown in Figure 3.4, is of four oblong items to be placed in a nonconvex container. Figure 3.5 shows a one-component $U_{0j}$ polygon for this example, with $U_{0j}^*$ polygons shaded in black. Figure 3.6 shows a two-component $U_{ij}$ polygon, with $U_{ij}^*$ polygons.

These examples have the property that each component of $U_{ij}^*$ intersects the boundary of $U_{ij}^{\text{orig}}$. That is, they share part of the boundary in common. This phenomenon occurs frequently in practice. The reason this behavior occurs often in practice is due to the characterization of translational containment problems in [DM, DM95], summarized by Theorem 2.7.1. The following theorem formalizes the relationship between Theorem 2.7.1 and the boundary intersection behavior.

---

$^2$The algorithms which generate these examples appear later in this chapter.
Figure 3.1: Boundary intersection example 1: the containment problem

Figure 3.2: Boundary intersection example 1: two one-component \( U_{0j} \)s, with \( U_{0j}^* \)s
Figure 3.3: Boundary intersection example 1: two two-component $U_{ij}$s, with $U_{ij^*}$s

Figure 3.4: Boundary intersection example 2: the containment problem
Figure 3.5: Boundary intersection example 2: a one-component $U_{0j}$, with $U_{0j}^x$s

Figure 3.6: Boundary intersection example 2: a two-component $U_{ij}$, with $U_{ij}^x$s
Theorem 3.2.1 If an instance of translational containment has a valid configuration, then, for each \( j, 0 \leq j \leq k \), there exists \( i \neq j, 0 \leq i \leq k \), such that at least one component of \( U_{ij}^* \) shares part of its boundary with \( U_{ij}^{\text{orig}} \).

Proof: Part (2) of Theorem 2.7.1 implies the weaker condition that, if a solution exists, then a solution exists such that, for each \( j, 0 \leq j \leq k \), there is some \( i \neq j, 0 \leq i \leq k \), such that \( P_i \) and \( P_j \) are in (at least single) contact. A contact corresponds to a point on the boundary of \( U_{ij}^{\text{orig}} \); this point remains after any valid restriction because it is associated with a valid configuration. Thus, a component of a replacement of \( U_{ij} \) in a valid restriction of \( \mathcal{U} \) must intersect the boundary of \( U_{ij}^{\text{orig}} \) for some \( i, 0 \leq i \leq k \).

This explains components of \( U_{0j} \) that cling to its boundary, as in Figure 3.2 and Figure 3.5. It also explains components of \( U_{ij}, 1 \leq i \leq k \), that cling to the “inner” boundary, which is the boundary of \( P_i \oplus -P_j \) (see the right-hand two-component \( U_{ij} \) of Figure 3.3).

Theorem 3.2.1 is an important consideration in designing a containment algorithm. The boundary intersection behavior occurs frequently in practice, and so it is unreasonable to expect, for example, that the convex hull of a replacement polygon is strictly inside the original polygon.

3.2.2 Loose Fits

It is clear that little restriction can be expected if the containment problem has a “loose-fitting” solution. In the extreme case of no container, no restriction is possible at all.

3.2.3 Annular Configuration Spaces

For a given set of items, as the container becomes less restrictive, each replacement of a \( U_{ij} \) polygon in a restriction of \( \mathcal{U} \), for \( 1 \leq i \leq k \), assumes an annular shape. Here we use the terms annular and annulus in the topological sense (see Figure 3.7).

This behavior occurs because, when the fit is loose enough, each point on the boundary of \( P_i \oplus -P_j \) is a valid relative displacement. That is, each possible contact between \( P_i \) and \( P_j \) is legal. An example is shown in Figure 3.7. In this example the outer boundary of the annulus is formed by our geometric restriction (see Section 3.3).

The annular shape presents a challenge for certain types of containment algorithms. For example, if the algorithm uses external approximations to the \( U_{ij} \)s, then it will consider a configuration to be valid if every \( t_j - t_i \) is within the approximation to \( U_{ij} \). However, if some \( t_j - t_i \) is within the inner hole of the annulus, then \( P_i \) and \( P_j \) overlap and the configuration is not valid.

3.2.4 Size and \( s \)-Analysis

Restriction reduces the area of \( \mathcal{U} \), but it is not guaranteed to reduce the number of vertices of \( \mathcal{U} \). This can cause problems in analyzing the running time of a restriction or a containment algorithm. For this reason, for a given set of restriction operations, we choose to express the running time of such an algorithm in terms of the maximum number of vertices \( s \) generated by applying any sequence of the restriction operations to \( \mathcal{U} \) [DM95]. Expressing running time in terms of \( s \) is what we call size analysis, or \( s \)-analysis of the algorithm, as
introduced in Section 2.5. The use of s-analysis gives a measure of the practical running time of a containment algorithm.

Suppose \( U \) has a finite number of vertices. Some sets of restriction operations on \( U \) have finite \( s \), even when applied an infinite number of times. One example is intersections using vertical (or horizontal) half-planes. Intersections with arbitrary half-planes, however, cause \( s \) to be infinite. The question of whether or not the operations we use in our valid restrictions cause \( s \) to be infinite is an open problem, and a subject of future work (see Section 10.2.1). We show below that if all possible valid restrictions are performed on \( U \), the resulting number of vertices is finite.

**Theorem 3.2.2** \( U_{ij}^* \) has a finite number of vertices and a finite number of connected components.

**Proof:** We first show that \( U_{ij}^* \) has a finite number of vertices. It then follows that the number of connected components is also finite. Recall from Section 2.2 that \( m = \max_{1 \leq i \leq k} m_i \).

Here we let \( M = \max_{0 \leq i \leq k} m_i \). We establish a rough upper bound on the number of vertices in \( U_{ij}^* \) by first finding an upper bound on the number of vertices of the \( 2k \)-dimensional free configuration space \( S \). Each vertex of \( S \) is the intersection of \( 2k \) hyper-planes. Let \( Q \) be an upper bound on the number of hyper-planes; then the number of vertices of \( S \) is no more than \( Q^{2k} \), because it is well-known that the size of an arrangement of \( n \) hyper-planes in \( d \) dimensions is \( \Theta(n^d) \) [EdE87]. Each hyper-plane corresponds to a contact between two polygons \( P_i \) and \( P_j \). For each polygon pair there are at most \( M^2 \) contacts, and there are

---

[3] Knowing all possible valid restrictions is NP-hard.
$(k + 1)k/2$ pairs, so $Q = M^2(k + 1)k/2$. Thus, an upper bound on the size of the free configuration space is given by: $O((M^2(k + 1)k/2)^{2k}) = M^{4k}((k + 1)k)^{2k}$. Now, $U_{ij}^*$ is a projection of $S$. The projection step can introduce new vertices, and so the number of vertices in $U_{ij}^*$ is bounded by the size of an arrangement of the hyper-planes. Thus, the number of vertices in $U_{ij}^*$ is bounded by $O((M^2((k + 1)k)^{2k})/2)$. Therefore, $U_{ij}^*$ has a finite number of vertices and connected components.

We note that tighter bounds have been obtained for the size of the free configuration space in special cases. For example, Avnaim and Boissonnat [AB87] show that for two non-convex polygons and a nonconvex container, the free configuration space is of size $O(M^{20})$. Recently, for example, Aronov and Sharir [AS94] showed that, for a convex polyhedron $B$ translating in 3-space among $k$ convex polyhedral obstacles $A_1, \ldots, A_k$ with disjoint interiors, the free configuration space for $B$ has size $O(Mk\log^2 k)$, where $M$ is the total size of the Minkowski sums $P_i = A_i \oplus (-B)$, for $i = 1, \ldots, k$.

The number of connected components of the restricted $U$ is closely related to the number of vertices. We simply remark here that restriction can fragment $U$ into many small connected components, and that this should be taken into account when designing a containment algorithm.

### 3.2.5 Identical Items

If $P_i$ and $P_j$ are identical (with no rotations allowed), then $U_{0i} = U_{0j}$, so $U_{ij}^*$ and $U_{0j}^*$ can replace each other. In this case, $P_i \oplus -P_j$ is symmetric about the origin\(^4\). If any point $p$ in $U_{ij}$, $1 \leq i \leq k$, is removed by a restriction, then $-p$ can also be removed. Thus, $U_{ij}^*$ is symmetric with respect to the origin.

This symmetry reflects the fact that, given a valid configuration, identical items produce valid equivalent configurations\(^5\). If an algorithm expects to deal often with identical items, it is wise to take advantage of the equivalent configurations (see Section 3.8).

### 3.3 Steady-State Geometric Restriction

In [DM, DM95] we present restrictions which operate directly on $U$. The restrictions use the Minkowski sum and a closed intersection operator (see Section 3.9). Each element of $U$ is replaced using operations that include other elements of $U$ as input. Theorem 3.3.1 below encompasses all three restrictions from [DM, DM95]. Figure 3.8 illustrates the theorem.

**Theorem 3.3.1** $U_{ij} \leftarrow U_{ij} \cap (U_{ih} \oplus U_{hj})$, \hspace{1cm} $0 \leq h, i, j \leq k, h \neq i \neq j$, is a valid restriction.

**Proof:** The theorem states that a valid displacement from $P_i$ to $P_j$ must be a valid displacement from $P_i$ to $P_h$ plus a valid displacement from $P_h$ to $P_j$. \hfill $\square$

\(^4\)Guibas, et al. note [GRS83] that, if $A$ is convex, then $A \oplus -A$ is symmetric with respect to the origin. The extension to arbitrary $A$ is trivial.

\(^5\)Two configurations are equivalent if their $t$s correspond to a permutation of identical items.
3.3.1 Avnaim and Boissonnat’s Solutions to 2NN and 3NP

We show that the operation of Theorem 3.3.1 can be viewed as a generalization of Avnaim and Boissonnat’s [AB87] solutions to 2NN and 3NP.

Avnaim and Boissonnat show for 2NN that the set of all the valid relative positions of $P_j$ with respect to $P_i$ is given by (in our notation): $U_{ij}^* = U_{ij} \cap (U_{00} \oplus U_{0j})$. This is an exact solution. Each element of the set $U_{ij}^*$ corresponds to a valid configuration. We observe that, for $k$NN, each valid displacement for two polygons must also be a solution for 2NN.

Avnaim and Boissonnat observe that, for 3NP, a valid translation $t_1$ exists for $P_1$ only if:

$$t_1 \in U_{01} \cap (U_{02} - (t_2 - t_1)) \cap (U_{03} - (t_3 - t_1)) \neq \emptyset$$

where $(t_2 - t_1) \in U_{12}$ and $(t_3 - t_1) \in U_{13}$. Thus, $U_{01} - U_{01} \cap (U_{02} \oplus U_{21}) \cap (U_{03} \oplus U_{31})$ is a valid restriction for 3NP.

For their solution to 3NP, Avnaim and Boissonnat show that any vector in $U_{12} \cap (U_{13} \oplus U_{32})$ is guaranteed to have an associated displacement for $P_2$ with respect to $P_1$ corresponding to a solution to 3NP. The proof relies on their application of Helly’s Theorem. Suppose we have three parallelograms which are the same affine transformation of (different but axis-parallel) rectangles. If the three parallelograms intersect pairwise, then all three have a common point. We observe that, in the general case, $U_{ij}^* \subseteq U_{ih} \oplus U_{hj}$ for $h \neq i \neq j$.

In our notation, Avnaim and Boissonnat’s solution to 3NP applies $U_{ij} \leftarrow U_{ij} \cap (U_{00} \oplus U_{0j})$, $1 \leq i < j \leq 3$, followed by $U_{ij} \leftarrow U_{ih} \oplus U_{hj}$ for $1 \leq h \neq i \neq j \leq 3$. Each element of $U_{ij}$ then corresponds to a solution to 3NP.

3.3.2 Steady-State Restriction

One would hope that repeated applications of the restriction in Theorem 3.3.1 would yield the set of $U_{ij}^*$ in the limit. Actually, this is always true only for $k = 2$. For $k = 3$, one can construct counterexamples, but only with difficulty. For $k > 3$, it is easy to construct counterexamples. We iteratively apply the valid restriction of Theorem 3.3.1 until no more significant shrinking occurs. We denote the resulting restriction by SSGR (steady-state geometric restriction). In the pseudo-code below, Empty is a set containing $k(k+1)/2$ copies of the empty set.
SSGR(\(\mathcal{U}, k\))

\[
\text{do}
\]

\[
\text{for } (i = 0 \text{ to } k)
\]

\[
\text{for } (j = i + 1 \text{ to } k)
\]

\[
\text{for } (h = 0 \text{ to } k)
\]

\[
\text{if } (h \neq i \neq j)
\]

\[
U_{ij} \leftarrow U_{ij} \cap (U_{ih} \oplus U_{hj})
\]

\[
\text{if } \text{(any } U_{ij} \in \mathcal{U} = \emptyset) \text{ return } \text{Empty } \mathcal{U}
\]

\[
\text{else } \quad U_{ji} \leftarrow -U_{ij}
\]

\[
S = \text{maximum fractional area shrinkage in } \mathcal{U}
\]

\[
d = \text{maximum diameter of a polygon in } \mathcal{U}
\]

\[
\text{while } (S > \alpha \text{ and } d > \epsilon)
\]

\[
\text{return } \mathcal{U}
\]

The parameter \(\epsilon\) is specified by the user. It is used to help judge when to terminate the restriction process. The parameter \(\alpha\) is not specified by the user. It is the relative area shrinkage, \(0 \leq \alpha \leq 1\). For \(\alpha = 0\), the program achieves a “true” steady state (if the maximum polygon diameter is greater than \(\epsilon\) and the maximum area is greater than \(\epsilon^2\)) in which additional restrictions cannot shrink the polygons further. However, this can require an infinite number of iterations. We use \(\alpha = 0.1\) and \(\epsilon = .5\) in our implementation; these are satisfactory for our applications. Using nonzero values of \(\alpha\) and \(\epsilon\) gives us a bound on the number of restriction iterations (see Section 3.3.3).

The steady-state restriction process is quite powerful. We know that \(U_{ij}^* \subseteq U_{ih} \oplus U_{hj}\) for \(h \neq i \neq j\). However, it is also true that: \(U_{ij}^* \subseteq U_{ih} \oplus U_{hq} \oplus U_{qj}\) for \(q, h \neq i \neq j\). In order to generalize this result, we define the notion of a list restriction. We then show in Theorem 3.3.2 that SSGR satisfies all list restrictions.

**Definition 3.3** Given a list of indices \(\pi = i, \pi_1, \pi_2, \pi_3, \ldots, \pi_{q-1}, \pi_q, j\), the list restriction of \(U_{ij}\) corresponding to \(\pi\) is the restriction:

\[
U_{ij}^\pi = U_{ij} \cap (U_{i\pi_1} \oplus U_{\pi_1, \pi_2} \oplus U_{\pi_2, \pi_3} \oplus \cdots \oplus U_{\pi_{q-1}, \pi_q} \oplus U_{\pi_q, j})
\]

In the following theorem, our use of the term “steady state” refers to the “true” steady state described above, in which \(\alpha = 0\).

**Theorem 3.3.2** If multiple applications of SSGR reach a steady state, then all possible list restrictions are satisfied.

**Proof:** The proof is by induction on the length of a list, which is the number of indices in the list. The base case, for a path of length two, is true because SSGR satisfies all list restrictions of length two. The induction hypothesis is that the list restriction for every list of length \(q + 1\) is satisfied. Consider the list \(\pi = i, \pi_1, \pi_2, \ldots, \pi_a, \pi_b, \pi_c, \ldots, \pi_q, j\) of length \(q + 2\). We show that the list restriction for \(\pi\) is satisfied. First, by the inductive hypothesis, the list restriction for the path \(\pi' = i, \pi_1, \pi_2, \ldots, \pi_a, \pi_c, \ldots, \pi_q, j\) of length \(q + 1\) is satisfied, so

\[
U_{ij}^\pi \subseteq U_{ij}^\pi' \cap (U_{i\pi_1} \oplus \cdots \oplus U_{\pi_{q-1}, \pi_q} \oplus \cdots \oplus U_{\pi_q, j})
\]
Now, because a steady state has been achieved, applying SSGR to $U_{\pi_a \pi_c}$ does not change $U_{\pi_a \pi_c}$. This means

$$U_{\pi_a \pi_c} \subseteq U_{\pi_a \pi_c} \cap (U_{\pi_a \pi_b} \oplus U_{\pi_b \pi_c})$$

and hence $U_{\pi_a \pi_c} \subseteq U_{\pi_a \pi_b} \oplus U_{\pi_b \pi_c}$. Substitution yields

$$U_{ij} \subseteq U_{ij} \cap (U_{i\pi} \oplus \cdots \oplus U_{\pi_b \pi_b} \oplus U_{\pi_b \pi_c} \oplus \cdots \oplus U_{\pi_{qj}})$$

which is the list restriction for $\pi$. Thus, the list restriction for $\pi$ is satisfied. \qed

### 3.3.3 Running Time

Here we calculate the asymptotic running time of SSGR. We assume:

- $s$ is the largest number of vertices of any polygon generated by a sequence of intersection and Minkowski sum operations on polygons in $\mathcal{U}$,
- $\epsilon$ is a real number representing a lower bound on the maximum diameter of a polygon in $\mathcal{U}$,
- $\alpha$ is the fractional shrinkage in area required by each iteration of SSGR.

**Theorem 3.3.3** The running time of SSGR is in $O(\log \left(\frac{1}{\epsilon}\right) k^3 s \log s)$.

**Proof:** Let $r$ be the number of iterations of the while loop in SSGR. We have:

$$r = O(\log_{1-\alpha} \left(\frac{1}{\epsilon}\right)) = O(\log \left(\frac{1}{\epsilon}\right))$$

We perform the operation $U_{ij} := U_{ij} \cap (U_{ih} \oplus U_{kj})$ inside of three nested loops. Each operation takes $O(s \log s)$ time. Each loop executes $k$ times. Therefore, the running time is $k^3 r(s \log s)$.

The total running time is therefore:

$$O(\log \left(\frac{1}{\epsilon}\right) k^3 s \log s)$$

\qed

### 3.3.4 Example

Figure 3.9 shows an example from the apparel industry. Three “trim” pieces are to be placed. The nearly triangular shape displayed in Figure 3.10 is a scaled version of the $U_{0j}$ polygon for the rectangular trim piece with respect to the container; on it the $U_{0j}^* \pi_a \pi_c$ polygon for the SSGR of $\mathcal{U}$ is shown as a black filled region. More than 95% of the area is eliminated by the restriction process: all that remains is the tip. Figure 3.11 shows the two-component $U_{ij}^*$ polygon for the rectangular trim piece with respect to one of the other two trim pieces. The $U_{ij}^*$ polygon for the SSGR of $\mathcal{U}$ is shown as a black filled region. Again, more than 95% of the area is removed by our restrictions. SSGR takes three seconds to execute this example on a 50 MHz SPARCstation.
Figure 3.9: SSGR containment problem: place 3 convex items

Figure 3.10: A one-component $U_{0j}$, with $U_{0j}^*$, for the SSGR of $\mathcal{U}$
3.3.5 Effectiveness

In order to gauge the effectiveness of SSGR for $3 \leq k \leq 8$, we applied it to more than 100 containment problems. Some of these were infeasible examples, and some were feasible. Most of these examples were drawn from apparel manufacturing. Typical ranges for $m$ and $n$ were $4 \leq m \leq 100$ and $100 \leq n \leq 300$. For our apparel examples, we produced each feasible containment problem by removing a group of neighboring items from an existing layout. To create an infeasible problem, we moved some of the remaining items a small distance into the container. In all cases, the area of the container was at least as large as the sum of the item areas. In addition, each pair of items fit into the container, so that all $U_{ij}$s were non-null before applying SSGR.

The actual running time of SSGR, in practice, depends on the values of $k$, $m$, $n$, $\epsilon$, $\alpha$, and the speed of the underlying set operations. We currently use $\epsilon = .5$ and $\alpha = .1$. Although, in theory, SSGR can continue for many rounds before reaching a steady state, we find that if we terminate SSGR when no polygon’s area decreases by at least 10%, one or two rounds is usually sufficient. For these values of $\epsilon$ and $\alpha$, $3 \leq k \leq 10$, our current implementation of set operations, and the typical ranges for $m$ and $n$ given above, our SSGR runs approximately ten times more slowly than the steady-state linear programming restriction of Section 3.4. On a 50 MHz SPARCstation, SSGR running times tend to range from less than one second for small values of $k$, $m$ and $n$ to several minutes for larger values of $k$ near 10, $m$ near 100, and $n$ near 300. For more details, see the running times for our approximate algorithm in Section 5.2.3; this algorithm uses SSGR.

Our focus here is on how well SSGR restricts $U$. We measure effectiveness as the
percentage of area removed by SSGR, averaged over all $U_{ij}$s. Since the $U_{ij}^{\text{orig}}$s are unbounded and infinite in area (for $i, j \neq 0$), we apply Equation 3.1 before measuring performance. As expected, little or no restriction occurs for loose fits.

**Infeasible Examples**

Table 3.1 shows results for infeasible examples. We use the $kCN$ and $kNN$ notation introduced in Section 1.2.1. For each case where SSGR does not detect infeasibility, we give the average relative percentage area restriction over all the $U_{ij}$s. In all the failure cases, this figure is less than 30%.

In slightly more than 50% of the 15 cases, SSGR detects infeasibility. This suggests that SSGR can be valuable as a preprocessing step for a containment algorithm.

<table>
<thead>
<tr>
<th>Type</th>
<th>Detected?</th>
<th>%</th>
<th>Type</th>
<th>Detected?</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>3CN</td>
<td>yes</td>
<td></td>
<td>4CN</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>4CN</td>
<td>no</td>
<td>20%</td>
<td>4CN</td>
<td>no</td>
<td>29%</td>
</tr>
<tr>
<td>4NN</td>
<td>yes</td>
<td></td>
<td>4NN</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>4NN</td>
<td>yes</td>
<td></td>
<td>4NN</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>5NN</td>
<td>no</td>
<td>0%</td>
<td>5NN</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>6CN</td>
<td>no</td>
<td>14%</td>
<td>6NN</td>
<td>no</td>
<td>3%</td>
</tr>
<tr>
<td>7CN</td>
<td>no</td>
<td>0%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: SSGR applied to infeasible examples

**Feasible Examples**

For the 85 feasible examples, $3 \leq k \leq 8$. For 72% of the feasible examples, the average relative percentage area restriction over all the $U_{ij}$s is below 30%. We found that 16% is above 60%, and 10% is above 90%. This data shows a pattern similar to the infeasible examples: for our test cases, it is rare to see a modest amount of SSGR restriction of around 50%.

**Constructing Restrictive Examples**

Our goal here is to provide insight into the types of situations in which SSGR performs well. Naively, one might think that a tight-fitting example whose only solutions are highly interlocking would have high restriction. However, the requirements are more subtle. The only valid configurations for the 6NN example in Figure 3.12 are tight-fitting and highly interlocking, yet the average SSGR percentage for this example is only 1.8%.

In order to create a “restrictive” example, we want $U_{ij} \subseteq (U_{ih} \oplus U_{kj})$ to be false as much as possible. That is, we desire restrictive triples, as defined below in Definition 3.4. We show below some ways of creating restrictive triples.

**Definition 3.4** If $U_{ij} \not\subseteq (U_{ih} \oplus U_{kj})$, then $(P_i, P_j, P_k)$ is a restrictive triple.
Figure 3.12: A 6NN example with 1.8% SSGR restriction

Figure 3.13: “L”-shaped container and 3 items
A simple “L”-shaped container can be used to induce restriction because the base of the “L” can be used to trap an item. Consider the example in Figure 3.13. Restrictions will reduce the area of $U_{01}$ and $U_{02}$, because $P_3$ can only fit in the base of the “L”; this prohibits $P_1$ and $P_2$ from being in the base of the “L”. The limited possibilities for $P_3$ will also cause restrictions to reduce the area of $U_{23}$. SSGR for this example reduces the area of $U_{01}$ and $U_{02}$ each by more than 50%, and the area of $U_{12}$ by 80%.

Figure 3.14 shows a 5NN example which has average SSGR restriction of 52%. The shape of the container greatly restricts the positions for the narrow pointed item (in the middle of the column of unplaced items). This means there are many restrictive triples. Swapping this item with the rectangular item at the bottom right corner of the layout produces an example with average SSGR restriction of only 11%.

![Figure 3.14: A 5NN example with 52% average SSGR restriction](image)

In both Figure 3.13 and Figure 3.14, the shape of the container limits the possible positions of one item $P_i$; this, in turn, removes placement possibilities for the remaining items. Thus, $U_{0j} \notin (U_{0i} \oplus U_{ij})$, for some $i \neq j$.

We define a “restrictive” container as follows. Consider $U_{\text{orig}}$. Apply Equation 3.1 so that each $U_{ij}$ is bounded. If, for the resulting $U$, there is some $U_{0j} \notin (U_{0i} \oplus U_{ij})$, then the container is restrictive. Note that, except for applying Equation 3.1, we do not use replacement polygons in this definition. Under these assumptions, if there is a restrictive triple which includes $P_0$, then the container is restrictive. Clearly, if a containment problem has no restrictive triple, then the container is not restrictive.

Figure 3.15 shows a different type of situation. Here the container is complicated, and the containment problem is infeasible. SSGR detects the infeasibility. However, the container is not restrictive because the placement positions for a single item within the container are not limited by adding a second item to the container. In this example, it is restrictions of the form $U_{ij} - U_{ij} \cap (U_{ih} \oplus U_{hj})$, $i \neq j \neq h \neq 0$ which are instrumental in detecting the infeasibility.

If we choose an unrestricted container, we must select the items carefully in order to induce strong restriction. For example, given a rectangular container of height $h$ and width $w$, we can form three items of height $h/3$ and width $w$ (see Figure 3.16, in which the items are shown placed in the container). In this case, each $U_{0j}^{\text{orig}}$ is a vertical line segment of
length $2h/3$. Each $U_{ij}$, $1 \leq i \neq j \leq 3$, after applying Equation 3.1, is a vertical line segment of length $4h/3$ with a segment of length $2h/3$ removed from its center. Restrictions shrink each $U_{ij}$, $1 \leq i \neq j \leq 3$, down to four points, which are the endpoints of the two line segments. These are propagated to $U_{0j}$, $1 \leq j \leq 3$, leaving three points on its vertical line, spaced at intervals of $h/3$.

![Diagram](image)

Figure 3.16: A 3CC example with an unrestricted container

This example does not work for $k > 3$ because, in that case, there is no restrictive triple which excludes $P_0$.

**Theorem 3.3.4** Given a rectangular container of height $h$ and width $w$, and an integer $k > 3$, for the translational containment problem with $k$ rectangular items, each of height $h/k$ and width $w$, the $U_{ij}$, $1 \leq i \neq j \leq k$, are unaffected by restrictions of the form: $U_{ij} = U_{ij} \cap (U_{ih} \oplus U_{hj})$ for which neither $i$, $j$, nor $h$ is 0.

**Proof:** The $U_{ij}$, $1 \leq i \neq j \leq k$, are identical because the items are identical. Hence, each replacement of such a $U_{ij}$ is the same. Recall from Section 2.2 that $U_{ij} = -U_{ji}$; each restriction therefore reduces to $U_{ij} = U_{ij} \cap (U_{ij} \oplus U_{ij})$. This has no effect if $U_{ij} \subseteq U_{ij} \oplus U_{ij}$. Each $U_{ij}$, after applying Equation 3.1, is a vertical line segment of length $2h(k - 1)/k$ with a segment of length $2h/k$ removed from its center. This forms two vertical line segments; let $a$, $b$, $c$, and $d$ be their endpoints, from top to bottom. $U_{ij} + a$ covers all of segment $ab$ except for a segment of length $h/k$, extending downwards from $a$. $U_{ij} + b$ covers all of segment $ab$ except for a segment of length $h/k$, extending upwards from $b$. Because each of the two segments comprising $U_{ij}$ is of length $\geq 2h/k$, the union of these sums completely
covers segment $ab$. Similarly, the union of the sums $U_{ij} + c$ and $U_{ij} + d$ cover segment $cd$. Thus, $U_{ij}$ is a subset of $U_{ij} \oplus U_{ij}$, and therefore the $U_{ij}$s, $1 \leq i \neq j \leq k$, are unaffected by restrictions of the form: $U_{ij} \leftarrow U_{ij} \cap (U_{ih} \oplus U_{kj})$ for which neither $i$, $j$, nor $h$ is 0. □

A containment problem which satisfies the conditions of Theorem 3.3.4 clearly also has no restrictive triples which include $P_0$. Thus, SSGR applied to such a problem produces no restriction.

Figure 3.17: A 6NC example with 53% average SSGR restriction

The previous example might suggest that a rectangular container cannot be restrictive for $k > 3$. This is not true. It is easy to create a restrictive triple involving a rectangular container for $k > 3$ using nonconvex items. Figure 3.17 shows one example for which the items are placed in the container. SSGR provides average restriction of 53% in this case, with the area of some $U_{ij}$s reduced by more than 74%. One of the restrictive triples in this example is $(P_0, P_1, P_6)$.

An example for $k > 3$ with a rectangular restrictive container and convex items can be constructed using only slightly more effort. It can be done using rectangles and the idea of guillotine cuts. An example is given in Figure 3.18; the items are shown placed in the container. This example has 48% average SSGR restriction. Item 1 participates in several restrictive triples. The amount of restriction is controlled by the width of item 1. In order to prevent items 2-5 from fitting side by side, the width of item 1 should be less than half the width of the container.

**Theorem 3.3.5** Given a rectangular container of height $h$ and width $w$, and an integer $k \geq 3$, for the translational containment problem with $k$ rectangular items, one of height $h$ and width $z < w/2$, and $k-1$ of height $h/(k-1)$ and width $w-z$, the $U_{ij}$, $1 \leq i \neq j \leq k$, are affected by restrictions of the form: $U_{ij} \leftarrow U_{ij} \cap (U_{ih} \oplus U_{kj})$ for which neither $i$, $j$, nor $h$ is 0.

**Proof:** Assume, w.l.o.g., that the item of width $z$ is item number 1. We show that $(U_{23} \cap (U_{21} \oplus U_{13})) \subset U_{23}$, which implies that the restriction $U_{23} \leftarrow U_{23} \cap (U_{21} \oplus U_{13})$ removes area from $U_{23}$. Refer to Figure 3.19, in which the origin is marked “o”. The two

---

6A guillotine cut is a common term in the cutting and layout world. It refers to a method of applying a sequence of horizontal and vertical cuts to an axis-parallel rectangle. Each cut goes all the way through the remaining rectangle. (See, for example, [DD92]).
shaded rectangles represent $U_{23}$. Each rectangle has width $2z$ and height $(k - 3)h/(k - 1)$. The rectangles are separated by a vertical distance of $h/(k - 1)$. The two vertical dashed lines represent $U_{21} = U_{13}$. Each line has length $h$, and is at a distance $w/2$ from the origin. The Minkowski sum $U_{21} \oplus U_{13}$ consists of three vertical lines of length $2h$; adjacent lines are separated by a distance of $w$. Two of them are at a distance of $w$ from the origin, and hence cannot intersect $U_{23}$. One of the vertical lines runs through the origin. Because it extends a length of $h$ above and below the origin, and each rectangle of $U_{23}$ ends at a distance of $(k - 2)h/(k - 1)$ above or below the origin, the intersection of $U_{23}$ with this vertical line consists of two vertical line segments (shown as thick lines in Figure 3.19). This intersection is clearly a proper subset of $U_{23}$.

Figure 3.19: $U_{23} \cap (U_{21} \oplus U_{13})$

Figure 3.20 shows another guillotine-like layout with the polygons placed in the container. In such an example, the area of some $U_{ij}$ polygons is reduced by as much as 90%. For example, the area of $U_{02}$ is greatly reduced due to items 1 and 4.

Our final example of how to construct a restrictive containment problem is shown in Figure 3.21. Figure 3.21 depicts a highly restrictive 6NN example. In this case, the large nonconvex item together with the two large ovals form a restrictive triple. Figure 3.22 shows the effect of SSGR on two one-component $U_{0j}$'s in this case; each has more than 90% of its area removed by the restrictions.
Figure 3.20: A 9CC example with strong SSGR restriction

Figure 3.21: A highly restrictive 6NN example
Barriers to Restriction

Restrictions of the form \( U_{ij} = U_{ij} \cap (U_{ih} \oplus U_{kj}) \) are ineffective when \( U_{ij} \subseteq U_{ih} \oplus U_{kj} \) or, equivalently, when \( U_{ij} \subseteq U_{ih} \oplus -U_{kj} \). The following theorem provides a condition for which this is true.

Theorem 3.3.6 Let \( A \) and \( B \) be polygons and let the origin be denoted by \( O \). If either \( O \subseteq B \) or \( A \) is connected and no translated copy of \( A \) lies entirely within the component of \( B \) that contains \( O \), then \( A \subseteq A \oplus -B \).

Proof: Consider a point \( p \in A \). To prove that \( A \subseteq A \oplus -B \), it suffices to show that \( p = a - b \), where \( a \in A \) and \( b \in B \). Translate \( A \) by \(-p\), and denote this \( A_p \). If \( B \) contains \( O \), then \( p = p - O \), so we're done. Otherwise, let \( B \) be the component of \( B \) which contains \( O \). No translated copy of \( A \) lies entirely within \( B \); this guarantees the existence of \( q = p \in A_p \) such that \( q - p \notin B \). If \( q - p \in B \), then \( q - p = b \in B \) implies \( p = q - b \), so we're done. Now assume \( q - p \notin B \). \( A_p \) is connected, and so there exists a connected path \( \pi \) from \( O \in A_p \) to \( q - p \in A_p \). The endpoints of \( \pi \) are also in different components of the exterior of \( B \), so \( \pi \) must intersect \( B \) at some point \( b \in B \). Then because \( \pi \in A_p \), there exists \( a - p \in A_p \) such that \( b = a - p \). This implies \( p = a - b \), which completes the proof. \( \square \)

The following corollary is immediate from the above theorem.

Corollary 3.3.7 Let \( A \) be a simple, bounded, connected polygon such that the origin is contained within the interior of the outer boundary of \( A \). Then \( A \subseteq A \oplus -A \).
Note: It is not true, in general, that $A \subseteq A \oplus -A$, even with the additional constraint that $A = -A$. Figure 3.23 presents a counterexample. In the figure, $A$ consists of two points, and $A = -A$. However, $A \oplus -A$ consists of three points which are not part of $A$. Therefore, $A \not\subseteq A \oplus -A$.

![Counterexample](image)

Figure 3.23: Counterexample for $A \subseteq A \oplus -A$

The next corollary follows immediately from transitivity.

**Corollary 3.3.8** Let $A$ and $B$ be polygons satisfying the conditions of Theorem 3.3.6, and let $C$ be a subset of $A$. Then $C \subseteq A \oplus -B$.

In practice, we observe that it is rare for SSGR to remove any area when all items are identical. Even when only some items are identical, we observe limited restriction. This makes sense in light of the above results. For example, if all the items are identical, then $U_{ij} = U_{ik} = U_{kj}$. If, in addition, $U_{ij}$ satisfies the conditions of Theorem 3.3.6 on $A$ and $B$, then $U_{ij} \subseteq U_{ik} \oplus -U_{kj}$, and therefore no restriction will occur. Although SSGR can be ineffective when items are identical, Milenkovic has recently designed a restriction to deal effectively with identical items (see the brief description of his symmetry breaking restriction in Section 3.8). In addition, we remind the reader that when items are identical, the number of subproblems created by using the translational containment characterization of Section 2.7 is reduced.

When the boundary of a $U_{ij}$, for $1 \leq i \leq k$, is a closed curve, it always contains the origin. In this case the $U_{ij}$ satisfies a necessary condition for Theorem 3.3.6 and its corollaries.

We tested the hypothesis that the extent to which the $U_{ij}$s, $1 \leq i \neq j \leq k$, are closed curves is a key factor in the effectiveness of SSGR. For each of the more than 80 feasible cases, we estimated the “closure” by shooting rays from the origin at equally spaced angles about the origin. Let $\delta$ be the fraction of rays which hit the $U_{ij}$. We recorded the average $\delta$ over all $U_{ij}$s, as well as the maximum and minimum $\delta$, for each example. We then calculated correlation coefficients for each of these with respect to the total restriction percentage. We hoped to find high negative correlation, indicating a linear relationship in which high $\delta$ and a small amount of restriction were related.

We did find that whenever the average $\delta$ was high, SSGR effectiveness was low. For $\delta \geq .9$, effectiveness was $\leq 18\%$, and for $\delta \geq .95$, effectiveness was $\leq 7\%$. The interlocking 6NN example of Figure 3.12 has average $\delta = .83$ and SSGR restriction of less than 2%. Some of its $U_{ij}$s are shown in Figure 3.24.

However, effectiveness is sometimes low even when average $\delta$ is not high. This tends to occur when several items are identical. The correlation coefficient for average $\delta = -.55$, meaning only 30% of the restriction results can be predicted using average $\delta$. The correlation coefficient for maximum $\delta = -.48$, and for minimum $\delta = -.49$. Thus, $\delta$ alone is not a sufficient predictor of SSGR effectiveness.
3.4 Steady-State Linear Programming Restriction

A different method of restriction, based on linear programming, was developed by Milenkovic and is described in [DM, MD95]. This steady state linear programming restriction technique (SSLPR) is an indirect one which uses the convex hulls of the configuration spaces. It works as follows. Temporarily replace every element \( U_{ij} \in \mathcal{U} \) by its convex hull \( H(U_{ij}) \). The set of constraints

\[
1 \leq i < j \leq k
\]

\[ t_j - t_i \in H(U_{ij}) \]

can be expressed as a linear program which is called the CLP (constraint linear program). A polynomial-time algorithm is given to project the \( 2k \)-dimensional feasible space \( F \) into any element \( U_{ij} \) of \( \mathcal{U} \). If the projection is denoted \( F_{ij} \), then the replacement

\[ U_{ij} \leftarrow U_{ij} \cap F_{ij} \]

is a valid restriction.

3.4.1 Example

We present an example of SSLPR from the apparel industry. Figure 3.25 depicts three “v” shaped items which are to be placed in a non convex gap in a clothing marker. Figure 3.26 shows two two-component \( U_{0j} \) polygons for this example. Their replacements

\[ \text{Figure 3.24: Selected } U_{ij} \text{s for 6NN example} \]
are shaded in black. Figure 3.27 shows two $U_{ij}$ polygons and their replacements. The one-component polygon on the left is unaffected by restriction. The area of the three-component polygon on the right, however, is visibly reduced. This example takes less than 1 second to execute on a 50 MHz SPARCstation.

Figure 3.25: SSLPR containment problem: place 3 nonconvex items

Figure 3.26: Two two-component $U_{0j}$s, with $U_{0j}^*$s for SSLPR of $U$
Figure 3.27: A one-component $U_{ij}$, with $U_{ij}$s for SSLPR of $U$.

### 3.4.2 Effectiveness

Here we discuss the effectiveness of SSLPR. We also compare the effectiveness of SSLPR with SSGR. Preliminary results on the effectiveness of the linear programming restriction appear in [Sha94].

SSLPR has the property that it cannot increase the number of reflex vertices in $U$. Of course, SSGR has no such guarantee.

SSLPR approximates each $U_{ij}$ by its convex hull. If all the $U_{ij}$s are convex, then SSLPR can give an exact solution to the containment problem. If a $U_{ij}$ is very different from its convex hull, then many invalid points will be considered valid by the linear program. Therefore, as Shankar observes [Sha94], SSLPR provides better restriction when the $U_{ij}$ polygons are similar to their convex hulls than when the $U_{ij}$s and their convex hulls are dissimilar.

Unfortunately, most of the $U_{ij}$s, $1 \leq i \neq j \leq k$ are quite different from their convex hulls immediately after applying Equation 3.1, because each becomes the difference between two polygons. For this reason, we observe very little SSLPR restriction for our collection of more than 100 test cases. Only 10 of the feasible examples have any SSLPR restriction. SSLPR is not able to detect infeasibility in any of the infeasible examples. For all the cases where SSLPR achieves some restriction, the SSLPR of $U$ is a superset of the SSGR of $U$.

SSLPR clearly cannot be expected to function well for examples having many annular $U_{ij}$s. This includes loose fits and identical items.

The examples in which SSLPR is effective have restrictive containers that force the $U_{0j}$ of some item to consist of a small, single, nearly convex component. For example, for the
“L”-shaped container of Figure 3.13 and its three items, SSLPR achieves an average of 62% restriction. SSLPR does not do well in situations for which the container is unrestricted. For example, Figure 3.20, where SSGR reduces the area of some $U_{ij}$s by up to 90%, has no SSLPR restriction.

Based on the success of SSLPR on examples like that of Figure 3.13, it is reasonable to expect SSLPR to be effective in propagating the effect of selecting a single (nearly convex) component of a $U_{ij}$ for consideration. An example is shown in Figure 3.28, where a single component of a $U_{ij}$ (on the far right) is selected, causing the area of the two two-component $U_{ij}$s on the left to be reduced. This “selection” is useful in a containment algorithm which relies on subdivision (see Section 5.3.2). For our current implementation, SSLPR is roughly ten times faster than SSGR; this makes it a practical type of restriction to use within a subdivision containment algorithm.

Although our tests suggest that SSGR is more powerful than SSLPR, there are, in theory, cases for which SSLPR can provide more restriction than SSGR. Also, in our hierarchical containment algorithm, we have occasionally seen SSLPR remove area not removed by SSGR (see Section 5.3.2.)

The following is an example for which SSLPR is more powerful than SSGR. Consider the container and three items in Figure 3.29. The container is the union of the boundaries of two equilateral triangles. Each triangle has sides of length $s$. The points $a$, $b$, and $c$ at alternating reflex vertices of the container form an equilateral triangle. Each of the three items has length $s$. Each also has two holes; each hole is $1/3$ of the way from an endpoint. The local origin of each item is indicated in the figure. The reader may verify that $U_{01} = \{b, c\}$, $U_{02} = \{c, a\}$, and $U_{03} = \{a, b\}$. Also, $U_{12} = \{0, a - b\}$, $U_{23} = \{0, b - c\}$,
and $U_{13} = \{0, a - c\}$. This example is infeasible because any simultaneous placement of the three items would force their endpoints to collide. SSGR is unable to detect this infeasibility, because the $U_{ij}$s satisfy $U_{ij} = U_{ih} \oplus U_{kj}$. However, there is also no feasible solution to the CLP using the convex hulls of the $U_{ij}$s, and so SSLPR can detect the infeasibility.

It is easy to construct an example for which SSGR removes a region which SSLPR cannot. Consider again the containment problem of Theorem 3.3.5. In that case, each $U_{1j}$ is a pair of parallel vertical line segments. The convex hull of these segments will cause the CLP to allow solutions in which item 1 overlaps another item. This, in turn, will cause points of the $U_{ij}$ for an identical pair of rectangles (which SSGR removes) to be valid for the CLP. Thus, points will be removed by SSGR that are not removed by SSLPR.

We conclude that SSGR and SSLPR are complementary restrictions.

![Figure 3.29: Example for which SSLPR is more powerful than SSGR](image)

### 3.5 Subset Restriction

If one item $P_i$ is a subset of another item $P_j$, then the following theorem can be used. The theorem assumes that when the local origins of the two items are aligned, then $P_i \subseteq P_j$.

**Theorem 3.5.1** If $P_i \subseteq P_j$, then, for $P_h$, $U_{jh}^* \subseteq P_i \oplus -P_h$ and $U_{hj}^* \subseteq P_h \oplus -P_i$.

**Proof:** $P_i \subseteq P_j$ implies that $P_i \oplus (-P_h) \subseteq P_j \oplus (-P_h)$. Since $P \subseteq Q$ implies that $\overline{Q} \subseteq \overline{P}$, we have $P_j \oplus (-P_h) \subseteq P_i \oplus (-P_h)$, so $U_{jh}^* \subseteq P_i \oplus -P_h$. For the second part, $U_{hj}^* = -U_{hj}^*$, and $P_i \oplus -P_h = -P_h \oplus -P_i$. Applied to $U_{hj}^* \subseteq P_i \oplus -P_h$ yield $-U_{hj}^* \subseteq -P_h \oplus -P_i$. This immediately gives $U_{hj}^* \subseteq P_h \oplus -P_i$.

**Corollary 3.5.2** If $P_i \subseteq P_j$, then $U_{jh} = U_{jh} \cap \overline{P_i \oplus -P_h}$ and $U_{hj} = U_{hj} \cap \overline{P_h \oplus -P_i}$ are valid restrictions.

We call the valid restrictions given by Corollary 3.5.2 subset restrictions. Subset restrictions can be useful when initially calculating $\mathcal{U}$. If a $U_{hi}$ is null, then $U_{hj}$ need not be calculated.

### 3.6 Union Restriction

This section introduces a valid restriction which assumes that a particular $U_{ij}$ is the union of a collection of polygons. First, we argue that if $A \subseteq U_{ij}$, then restrictions on $A$ are well-defined if we replace $U_{ij}$ in $\mathcal{U}$ by $A$. Recall from Section 2.2 that $U_{ij}^*$ is the set of
all values of $t_j - t_i$ which belong to a valid configuration, and recall from Section 3.1.2 that $U_{ij}^\ast$ is the replacement of $U_{ij}$ in a valid restriction of $\mathcal{U}$. Analogously, $A^\ast$ is the set of all values of $t_j - t_i$ in $A$ which belong to a valid configuration, and $A^\ast$ is the replacement of $A$ in a valid restriction.

**Lemma 3.6.1** If $A \subseteq U_{ij}$, then $A^\ast \subseteq U_{ij}^\ast$ and $A^\ast = A \cap U_{ij}^\ast$.

**Proof:** The first part follows immediately from $A \subseteq U_{ij}$. To obtain the second part, by definition, $A^\ast \subseteq A$; this together with the first part of the lemma imply $A^\ast \subseteq A \cap U_{ij}^\ast$. To obtain equality, observe that if a point $t_j - t_i$ is in $A \cap U_{ij}^\ast$ (and hence in $U_{ij}$), it is part of a valid configuration; therefore $t_j - t_i$ must also be in $A^\ast$. $\square$

**Theorem 3.6.2** If $U_{ij} = \bigcup_h A_h$, then $U_{ij}^\ast \subseteq \bigcup_h (A_h^\ast)$.

**Proof:** By definition of a valid restriction, $A^\ast_h \subseteq A^\ast$. This implies $\bigcup_h A^\ast_h \subseteq \bigcup_h A^\ast$. Now, since $A_h \subseteq U_{ij}$, Lemma 3.6.1 implies $A^\ast = A_h \cap U_{ij}^\ast$. Substituting this into $\bigcup_h A^\ast_h \subseteq \bigcup_h A^\ast$ yields $\bigcup_h (A_h \cap U_{ij}^\ast) \subseteq \bigcup_h A^\ast_h$. This implies $U_{ij}^\ast \cap (\bigcup_h A_h) \subseteq \bigcup_h A^\ast_h$. Since $U_{ij} = \bigcup_h A_h$, this simplifies to $U_{ij}^\ast \subseteq \bigcup_h (A_h^\ast)$. $\square$

Theorem 3.6.2 immediately yields a valid restriction, which we call a union restriction. Course-grained parallelism can be used to implement this restriction. In the pseudo-code below, let $A$ be the set of all the $A_h$ polygons whose union is $U_{ij}$, and suppose there are $l$ such polygons. $\mathcal{U}^h$ represents a copy of $\mathcal{U}$ for the $h$th processor, and $U_{ij}^h$ is processor $h$'s configuration space for $P_j$ with respect to $P_i$. Empty $\mathcal{U}$ is a set containing $k(k+1)/2$ copies of the empty set.

```
UNION-RESTRICT(\mathcal{U}, k, i, j, A, l)
    for (h = 1 to l) do in parallel
        \mathcal{U}^h = \mathcal{U}
        U_{ij}^h = A_h
        \mathcal{U}^h = ANY-VALID-RESTRICTION(\mathcal{U}^h, k)
    \mathcal{U} = Empty \mathcal{U}
    for (s = 1 to k) do
        for (t = 1 to k) do
            for (h = 1 to l) do
                U_{st}^h = U_{st} \cup U_{st}^h
    return \mathcal{U}
```

### 3.7 Characterization-Based Restriction

We present here a restriction which does not satisfy the criteria for a valid restriction. That is, it does not preserve the invariant $U_{ij}^\ast \subseteq U_{ij}$. However, the restriction is useful when only a single valid configuration is required, because if a valid configuration exists, a valid configuration also exists after the restriction. Thus, the restriction is a semi-valid restriction (see Definition 3.2).

The restriction is based on the containment characterization given in Section 2.7. We use a partial characterization based on the second part of Theorem 2.7.1, which guarantees
that if a valid configuration exists, then one also exists for which there are at least two contacts per polygon. We apply this to $P_0$, and use the weaker condition that the polygon has at least one contact. This means that if a valid configuration exists, there must exist a valid configuration for which some $t_h - t_0$ is on the boundary of its $U_{0h}$. Therefore, we preserve the existence of such a configuration if we replace each $U_{0h}$ by its boundary, apply any valid restriction, and then set $U$ equal to the union of the restricted sets. Course-grained parallelism can be used here.

As in the previous section, in the pseudo-code below, $U^h$ represents a copy of $U$ for the $h$th processor, and $U^h_{ij}$ is processor $h$'s configuration space for $P_j$ with respect to $P_i$. $\text{Empty}_U$ is a set containing $k(k+1)/2$ copies of the empty set.

```plaintext
BOUNDARY-RESTRICT($U$, $k$)
  for $(h = 1$ to $k)$ do in parallel
    $U^h \leftarrow U$
    $U^h_{0h} \leftarrow \text{BOUNDARY-OF}(U_{0h})$
    $\check{U}^h \leftarrow \text{ANY-VALID-RESTRICTION}(U^h, k)$
    $U \leftarrow \text{Empty}_U$
  for $(i = 1$ to $k)$ do
    for $(j = 1$ to $k)$ do
      for $(h = 1$ to $k)$ do
        $U_{ij} \leftarrow U_{ij} \cup U^h_{ij}$
      return $U$
```

### 3.8 Symmetry Breaking and Subset Substitution Restrictions

Section 3.2.5 observes that identical items produce equivalent configurations. These pose a special challenge for SSGR, as discussed in Section 3.3.5. In recent work [Mil95b], Milenkovic has developed a new form of restriction to overcome the difficulty that SSGR and SSLPR have with identical items. His idea is to arbitrarily choose a vector $v$, and then impose the constraint that $t_i \cdot v \leq t_j \cdot v$ if items $P_i$ and $P_j$ are identical. This has the effect of a 50% reduction in the area of $U_{ij}$. A “resymmetrizing” step is ultimately necessary to make the symmetry breaking restriction a valid restriction; without this it is only a semi-valid restriction.

The symmetry breaking restriction can also handle non-identical items through a valid restriction called a *subset substitution* restriction. The main idea of this restriction is as follows. If a polygon $A$ is a subset of each of a collection of polygons, then we can substitute a copy of $A$ for each of the polygons in the collection, calculate $U$ using these copies of $A$, perform restrictions on $U$ for the copies of $A$, and then use these restrictions as valid restrictions for the $U$ involving the original items. Note that this is not the same as the subset restriction of Section 3.5; in that case both $P_i$ and $P_j$ are to be placed simultaneously in the same layout, and $P_i$ is not substituted for $P_j$. In the most general form of the subset substitution restriction, one can replace each polygon $P_i$ by a subset $P'_i$. The $P'_i$'s can all be the same (as above), or they can all be different, or some the same and some different.
3.9 Closed Operators

Here we discuss an important design issue related to the type of set operations used to implement restrictions.

Consider a general polygon, i.e. a collection of polygonal components, where each component is connected.

**Definition 3.5** A general polygon \( P \) is **regularized** if it is equal to the closure of its interior.

Figure 3.30b depicts a regularization of the non-regularized general polygon of Figure 3.30a. Regularized polygons are not sufficient for our application. Using only regularized set operations can cause the algorithm to incorrectly conclude during a restriction that no solution exists. This can occur if the intersection of two polygons is either a line segment or a point. In fact, this situation can occur often for tight fits. One example is a stack of rectangles inside a rectangular container, where each rectangle to be placed has the same height as the container (see Figure 3.31).

![Figure 3.30: Regularization example](image)

A non-regularized polygon does allow isolated points and line segments (see Figure 3.30a). Unfortunately, it also allows isolated points or line segments to be removed from its interior. Thus, a non-regularized polygon need not be closed, i.e. equal to its closure - its boundary plus the closure of its interior. Since the configuration space we operate on is closed, we must deal with closed sets. We therefore need closed polygons. Union and intersection operations on closed sets are defined in the normal way, because the union and intersection of two closed sets is closed. However, the complement of a closed set is open, so we take the closure of the complement of a closed polygon.

Correct implementation of the algorithm requires the use of robust, closed set operations. The set operations must be implemented robustly in order to minimize error accumulation during the steady-state restriction process. Closed operators are important to both the geometric and linear programming restrictions. They are more crucial to geometric restrictions, since these rely more heavily on set operations.

![Figure 3.31: Two rectangles with degenerate intersection](image)
For our first implementation of SSGR, we used robust intersection, union, and complement operators which handled only regularized general polygons. Our robust set operators are based on geometric rounding [LM93b, MN89, Mil93a, Mil93b]. In order to shrink polygons to their boundaries, we added operators which can handle line segments and points as well as lists of polygons. However, our underlying polygon operators were still regularized. We are currently redesigning these operators so we will have a complete set of operators for closed polygons.

3.10 Summary and Conclusions

In this chapter, we have made several general observations about restrictions for translational containment. For example, we have observed that replacement polygons often have components which intersect the boundary of the original polygon, and shown when and why this occurs. These properties represent important considerations for designers of translational containment algorithms.

We have discussed the valid restriction SSGR in detail. We have evaluated the effectiveness of SSGR and of Milenkovic’s SSLPR. We conclude that SSGR is often much more effective on containment examples than SSLPR. SSLPR performs best when the configuration spaces are similar to their convex hulls. SSGR performs best when a restrictive triple of items exists. Both restrictions perform poorly for loose fits, identical items, and other circumstances that cause the configuration spaces to be closed curves containing the origin. Both restrictions are effective in propagating the effects of selecting a single (nearly convex) component of a polygon for consideration. SSGR and SSLPR are complementary. There are cases for which SSGR can restrict more than SSLPR, and cases for which SSLPR can restrict more than SSGR.

We have also introduced subset, union, and characterization-based restrictions, and discussed Milenkovic’s symmetry-breaking restriction.
4.1 Introduction

In this chapter we present evaluation and subdivision methods. Recall from Chapter 2 that, given a restricted hypothesis, the evaluation step tries to find a valid configuration within the hypothesis. If no solution is found by the evaluation step, then the subdivision step divides the current hypothesis into two sub-hypotheses such that if a solution exists within the current hypothesis, then a solution must exist within one of the two sub-hypotheses. To divide the current hypothesis, we partition one of the configuration spaces into two parts. The algorithm then recurses on each part while keeping the other configuration spaces constant.

Section 4.1.1 gives an overview of our work on evaluation. Section 4.1.2 outlines our work on subdivision. Section 4.1.3 gives an overview of this chapter.

4.1.1 Evaluation

In order to evaluate a hypothesis, we can invoke an exact algorithm for kNN at this point, or perhaps a MIP algorithm such as that of [DML94]. However, these alternatives are costly. For the purposes of our general containment algorithm given in Section 2.4, all we really require is a fast test which generates a configuration and tests to see if it is a valid configuration: \( t_j - t_i \in U_{ij}, \ 0 \leq i < j \leq k \). If the test succeeds, then we have a solution.

As explained in Section 2.4, the evaluator decides “yes”, “no”, or “maybe”. It decides “yes” if it finds a valid configuration within the current hypothesis. It decides “no” if it determines that there cannot be a valid configuration within the current hypothesis. It decides “maybe” if it fails to find a valid configuration in the current hypothesis but it is unable to rule out the possibility that one exists. In the “maybe” case, additional subdivision is required.

Our first evaluation method is a greedy one which operates directly on the configuration spaces. It shrinks each \( U_{ij} \) to a point, in turn, then propagates the effects of that shrinking via SSGR. If any \( U_{ij} \) becomes empty during this process, then that combination of \( U_{ij} \) points is not part of a valid configuration, and so the evaluator does not find a solution. In this case, it returns “maybe”. If, however, no \( U_{ij} \) becomes empty, then the collection
of single-point $U_{ij}$s determines a valid configuration and the evaluator returns “yes”. The key to the effectiveness of this procedure is a good choice of a shrinking point in a $U_{ij}$. We present several selection methods. One of them produces valid configurations, without using subdivision, in 66% of our test cases. This method maximizes the intersection of translations of the configuration spaces.

The disadvantage of the greedy evaluation method is that it cannot operate simultaneously on the configuration spaces. As $k$ increases, this becomes a more serious limitation. An evaluation method such as the indirect one of Milenkovic, which is based on a containment linear program (CLP), overcomes this limitation (see the discussion of the CLP in Section 3.4 and [Mil, DM95]). It approximates the $U_{ij}$s by their convex hulls and uses linear programming to check if there exists a configuration which is valid with respect to the convex hulls. If not, then the current $U$ cannot have a valid configuration. Thus, this evaluator is able to say “no”, in contrast to the greedy evaluator. If a valid configuration is found with respect to the convex hulls, and if each point found by the linear program is within its associated $U_{ij}$, then the configuration is a valid one. Otherwise, the configuration is an overlapping one. We apply the work of Li and Milenkovic on overlap reduction [Li94, ML, LM93c, LM93a] to improve this evaluation method.

Both of these evaluation techniques are powerful enough so that they can, in combination with restrictions, sometimes solve containment problems without any subdivision.

A containment algorithm based on CLP evaluation can solve a containment problem using a single CLP evaluation if every $U_{ij} \in U$ is convex. We present a theorem which specifies a condition under which a single CLP evaluation can solve a containment problem for a restricted $U$. This result has important implications. It provides a test which can be used to decide when to switch from a direct to an indirect approach within a containment algorithm. It also leads, in Section 4.5.3, to a theorem which guarantees that the number of worst-case subdivisions in certain containment algorithms is not increased by the use of restrictions.

### 4.1.2 Subdivision

In this chapter we also discuss subdivision methods. We define a set of possible goals for subdivision methods, and then discuss several subdivision methods which have one or more of these goals. One goal is to reduce the “size” of a $U_{ij}$, where size can be interpreted in a variety of ways. We present a size-based subdivision method which reduces the maximum dimension of the bounding box of a $U_{ij}$. Another goal is to reduce the number of combinatorial possibilities for a valid configuration. We discuss two such combinatorially-based subdivision methods which partition a $U_{ij}$ by extending one of its edges. These methods “move away” from the current configuration if it is overlapping. A different way to move away is offered by our new distance-based subdivision method.

### 4.1.3 Overview

Section 4.2 presents our greedy evaluation strategy. Section 4.3 describes the overlap reduction evaluation method. Section 4.4 gives a condition for which a single CLP evaluation can solve a containment problem for a restricted $U$. Section 4.5, presents subdivision techniques. Section 4.5.1 defines subdivision and discusses goals of subdivision. Section 4.5.2 discusses a size-based subdivision technique. Section 4.5.3 describes two
combinatorially-based subdivision techniques. Section 4.5.4 discusses our distance-based subdivision method. Section 4.6 summarizes this chapter and states conclusions.

4.2 Greedy Evaluation

4.2.1 Algorithm

The method we present here is a greedy strategy which shrinks each $U_{ij}$ to a point, in turn, then propagates the effects of that shrinking via SSGR. If any $U_{ij}$ becomes empty during this process, then the chosen combination of single $U_{ij}$ points cannot be part of a valid solution. If, on the other hand, we succeed in shrinking each $U_{ij}$ to a point and then applying SSGR without any $U_{ij}$ becoming empty, then a valid configuration exists (see proof of Theorem 4.2.1). Section 4.2.2 gives heuristics for selecting a shrinking point for each $U_{ij}$. In the pseudo-code below, the tolerance test varies depending on the subdivision method used.

**GREEDY-EVALUATE**($\mathcal{U}$, $k$

if ($\mathcal{U}$ fails tolerance test)
   return **BUILD-CONFIGURATION**($\mathcal{U}$, $k$

while (no $U_{ij} \in \mathcal{U} = \emptyset$ and some $U_{ij}$ has not been shrunk to a point)
   select smallest $U_{ij} \in \mathcal{U}$ which is not a point
   select a point $u_{ij} \in U_{ij}$
   $U_{ij} \leftarrow \{u_{ij}\}$
   $\mathcal{U} \leftarrow \text{SSGR}(\mathcal{U}, k$
   if (no $U_{ij} \in \mathcal{U} = \emptyset$)
      return **BUILD-CONFIGURATION**($\mathcal{U}$, $k$
   return **MAYBE**

**BUILD-CONFIGURATION**($\mathcal{U}$, $k$

$t_0 = (0, 0)$

for ($j = 1$ to $k$

   select $t_j \in U_{0j}$
   $\tau = (t_0, t_1, \ldots, t_k)$

return **YES**

**Theorem 4.2.1** If each $U_{ij}$ has been shrunk to a point and a subsequent application of SSGR does not cause any $U_{ij}$ to be empty, then **GREEDY-EVALUATE** yields a valid configuration for $\mathcal{P}$.

**Proof:** By definition, a valid configuration for $\mathcal{P}$ is a set of translations $t_i$, $0 \leq i \leq k$, such that $t_j - t_i \in U_{ij}$, $0 \leq i < j \leq k$. Since $U_{0j} \neq \emptyset$, $1 \leq j \leq k$ and consists of a single point, let $t_j$ be that point. Since $t_0 = \{0, 0\}$, we have $U_{0j} = t_j - t_0$. Now, for $U_{ij}$, $1 \leq i \neq j \leq k$, $\text{SSGR}(\mathcal{U}, k \neq 0 \rightarrow U_{ij} \subseteq U_{ih} \oplus U_{kj}$. In particular, for $h = 0$, $u_{ij} = u_{0j} - u_{0i}$, where $u_{ij} \in U_{ij}$, $u_{0j} \in U_{0j}$, and $u_{0i} \in U_{0i}$. Since $u_{0j} = t_j - t_0$ and $u_{0i} = t_i - t_0$, $u_{ij} = t_j - t_i$. Thus, the points of $\mathcal{U}$ constitute a valid configuration, and further restrictions cannot alter that fact.}$
4.2.2 Heuristics for Selecting \( u_{ij} \)

We present several heuristics for choosing a shrinking point \( u_{ij} \in U_{ij} \). In Section 4.2.3 we compare the effectiveness of these methods in practice. The most naive heuristic is to select a point at random. Our other heuristics are based on the following claim, which gives a necessary but insufficient condition for the existence of a valid configuration.

**Claim 4.2.2** If a configuration \( \{t_1, \ldots, t_k\} \) for \( \mathcal{P} \) is valid, then:

\[
U_{hi} \cap \bigcap_{j \neq i} (U_{hj} + (t_i - t_j)) \neq \emptyset
\]

**Proof:** For fixed \( h \), for each \( i \), \( U_{hi} + t_h - t_i \) contains the origin. The claim follows by translating each \( U_{hj} + t_h - t_j, i \neq j \), by \( (t_i - t_h) \). \( \square \)

Claim 4.2.2 means that a solution exists only if there are choices for the \( u_{ij} \) which cause the translated \( U_{ij} \)s to intersect. SSGB enforces pairwise intersection of the translated \( U_{ij} \)s. However, it does not guarantee the multiple intersection in Claim 4.2.2. This is illustrated by Figure 4.1 and Figure 4.2. Figure 4.1 shows a containment problem for three small convex items. Figure 4.2 depicts the three \( U_{0j} \) polygons for this example (unrestricted and scaled up). \( U_{01} \) is not translated, \( U_{02} \) is translated by a vector in \( U_{21} \), and \( U_{03} \) is translated by a vector in \( U_{31} \). In this case, a solution is not found because the polygons do not have a common intersection.

![Figure 4.1: 3CN example which does not yield a common intersection of translated \( U_{0j} \)s](image)

Avnaim and Boissonnat’s 3NP algorithm works because whenever the container is a parallelogram the \( U_{0j} \)s are parallelograms with sides parallel to the container. Avnaim and Boissonnat prove that for parallelograms with parallel corresponding sides, pairwise intersection implies multiple intersection. This version of Helly’s Theorem holds even if \( k > 3 \).

Note: One might be tempted to conclude from this that our algorithm provides an exact polynomial-time solution for \( k \)NP. However, this is highly unlikely to be true because \( k \)NP is NP-complete (see Section 2.3). The key to this paradox is that Claim 4.2.2 is necessary
but not sufficient. For 3NP, we can select $u_{12}$ and know that $u_{23}$ and $u_{31}$ exist and satisfy Claim 4.2.2. Unfortunately, for $k > 3$, we can’t necessarily obtain a self-consistent set of $u_{ij}$s for which $u_{ij} = u_{ik} + u_{kj}$ for all $h \neq i \neq j$.

Convexity of Union

Theorem 4.2.3 gives sufficient conditions for a set of $k$ polygons (convex or nonconvex) to have a common intersection\(^1\).

\textbf{Theorem 4.2.3} Given a set of polygons $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$, if $P_i \cup P_j$ is convex, $i \neq j$, $1 \leq i < j \leq k$, then

$$\bigcap_{i=1}^{k} P_i \neq \emptyset$$

\textbf{Proof:} If $P_k$ is nonconvex, let $p$ be a point which is on the boundary of $P_k$ but is not on the boundary of the convex hull of $P_k$, which we denote by $H(P_k)$. We claim $p \in P_i$, $1 \leq i \leq k$. Since $P_k$ is a closed polygon it contains its boundary; thus $p \in P_k$. Now, let $Q = H(P_k) \setminus P_k$. Because $P_i \cup P_k$ is convex, $H(P_k) \subseteq (P_i \cup P_k)$. This implies that $Q \subseteq P_i$. Since $P_i$ is a closed polygon it contains the boundary of any subset of itself. Therefore $P_i$ contains the boundary of $Q$, which contains $p$. This establishes the claim that $p \in P_i$, $1 \leq i \leq k$.

\(^1\)The generalization from 3 to $k$ polygons was suggested by Dan Roth.
Now suppose \( P_k \) is convex. We prove this case by induction. The base case for \( k = 1 \) is trivially true. The induction hypothesis is that our claim is true for \( k - 1 \). Now we consider \( P_k \). We claim that \( P_i \cup P_k \) is convex implies that \( P_i \cap P_k \neq \emptyset \) for \( 1 \leq i \leq k \). This is true because of the more general fact that if the union of two closed sets is connected, then they have a point in common. To establish this fact, we argue as follows. Let \( P_i \cup P_k \) be connected, \( p_i \in P_i \) and \( p_k \in P_k \). If either \( p_i \in P_k \) or \( p_k \in P_i \), we are done. Otherwise, since \( P_i \cup P_k \) is connected, there exists a path \( \gamma(t) \), \( 0 \leq t \leq 1 \), where \( \gamma(0) = p_i \) and \( \gamma(1) = p_k \) and either \( \gamma(t) \in P_i \) or \( \gamma(t) \in P_k \) for \( 0 < t < 1 \). Consider \( t^* = \sup \{ t | \gamma(t) \in P_i \} \). The point \( \gamma(t^*) \) is a limit point of \( P_i \cap \gamma(t) \) as \( t \) increases from \( 0 \). Let \( t^{**} = \inf \{ t | \gamma(t) \in P_k \} \). We claim \( t^{**} \leq t^* \). For, if \( t^* < t^{**} \) then there exists \( t' \) such that \( t^* < t' < t^{**} \) and \( \gamma(t') \) does not belong to either \( P_i \) or \( P_k \). Therefore \( \gamma(t^*) \in P_k \). This establishes \( P_i \cap P_k \neq \emptyset \) for \( 1 \leq i \leq k \).

Now we want to apply the induction hypothesis to the set of \( k - 1 \) nonempty polygons: \( \{(P_1 \cap P_k), \ldots, (P_{k-1} \cap P_k)\} \) to establish that \( \bigcap_{i=1}^{k} P_i \neq \emptyset \). We claim we can do this because \((P_i \cap P_k) \cup (P_j \cap P_k)\) is convex \( 1 \leq i < j < k \). This is true because

\[
(P_i \cap P_k) \cup (P_j \cap P_k) = P_k \cap (P_i \cup P_j)
\]

This is the intersection of two convex sets and is therefore itself convex. This completes the induction and establishes the theorem. \( \square \)

**Corollary 4.2.4** Suppose \( P \) is a set of polygons satisfying the conditions of Theorem 4.2.3. Let \( P_0 \) be convex and intersect all the other polygons: \( P_0 \cap P_i \neq \emptyset \), \( 1 \leq i \leq k \). It follows that,

\[
\bigcap_{i=0}^{k} P_i \neq \emptyset.
\]

**Proof:** In the proof of Theorem 4.2.3 for the case where one of the polygons is convex, we only used the property that \( P_i \cap P_k \neq \emptyset \). That property is satisfied here because of the conditions of the theorem. \( \square \)

Note that we have an exact solution to 3NN if \( V_1, V_2 + u_{21}, \) and \( V_3 + u_{31} \) satisfy the conditions of Theorem 4.2.3 or Corollary 4.2.4. However, these results cannot give an exact solution to kNN (see the note in the previous section). The results are still useful because they suggest a heuristic we can use in our kNN algorithm for selecting \( u_{ij} \). Theorem 4.2.3 suggests we choose \( u_{ij} \) which maximizes the convexity of \( U_{0i} \cup (U_{0j} + u_{ij}) \). The problem of finding the set of translations which maximizes the convexity of the union of a set of polygons is an open one (see Section 10.2.1). In our implementation we find an approximate answer (see Section 4.2.3).

**Maximizing Intersection Area**

In GREEDY-EVALUATE, shrinking a particular \( U_{ij} \) to a single point \( \{u_{ij}\} \) fixes the translation of \( U_{hj} \) with respect to \( U_{hi} \). Subsequent applications of SSGR can therefore cause the associated polygons in \( U \) to shrink. A containment solution can only be reached, however, if each polygon in \( U \) can be shrunk to a point without causing any polygon in \( U \) to vanish.

Suppose we choose the shrinking point for a \( U_{ij} \) to be the point which shrinks \( U_{hi} \) and \( U_{hj} \) the least. We might then have a better chance of preventing the polygons in \( U \) from

87
vanishing. This suggests the following heuristic for selecting the shrinking point: choose \( u_{ij} \) to be the point in \( U_{ij} \) such that the area of \( U_{hi} \cap (U_{kj} + u_{ji}) \) is maximized. If \( U_{hi} \) and \( U_{kj} \) are simple polygons with \( a \) and \( b \) vertices, respectively, then the optimal \( u_{ij} \) can be found exactly in polynomial time using a brute-force configuration space method, by solving \( O(a^2b^2) \) two-dimensional convex optimization problems [Mit94].

Solutions to some closely related problems often provide large intersection area in practice. One possibility is to align the centroids of the polygons. Another is to maximize the Hausdorff distance between the polygons. An inexpensive approximation to this is given in [ABB91]. Their result states that if you superimpose the lower left corner of the bounding boxes of two polygons, the Hausdorff distance between them is \( \leq (1 + \sqrt{2})\delta \), where \( \delta \) is the minimum Hausdorff distance possible for the two polygons. Of course, there is nothing special about the lower left corner. The proof works for any corresponding pair of bounding box corners.

In practice, we use a method which (approximately) maximizes the area of \( U_{hi} \cap (U_{kj} + u_{ji}) \). We overlay a grid on each component of \( U_{ij} \) and, at each grid point \( u_{ij} \), we measure the area of \( U_{hi} \cap (U_{kj} + u_{ji}) \). Section 4.2.3 describes this in more detail.

### 4.2.3 Comparing Heuristics for Selecting \( u_{ij} \)

We experimented with three different heuristics for selecting a shrinking point \( u_{ij} \in U_{ij} \): 1) a random point, 2) a point which (approximately) maximizes the area of \( U_{hi} \cap (U_{kj} + u_{ji}) \), and 3) a point which (approximately) maximizes the convexity of \( U_{hi} \cup (U_{kj} + u_{ji}) \).

Heuristics (2) and (3) use a two-dimensional grid to find an approximate maximum. For (2), we overlay a grid on each component of \( U_{ij} \) and, at each grid point \( u_{ij} \), measure the area of \( U_{hi} \cap (U_{kj} + u_{ji}) \). We choose a grid resolution for each component which gives ten grid points along the longest \((x \text{ or } y)\) dimension of the component. For each component, we find the average area and the maximum area. From the component with the highest average value, we select the point yielding the maximum value.

For (3), we again overlay a grid on each component of \( U_{ij} \). In this case, however, we measure the area of \( H(U_{hi} \cup (U_{kj} + u_{ji})) \setminus (U_{hi} \cup (U_{kj} + u_{ji})) \). For each component, we find the average area and the minimum area. From the component with the lowest average value, we select the point yielding the minimum value.

The goal of our experiments was to see which of the three heuristics was able to solve the most containment problems using only geometric restrictions and greedy evaluation. We performed several experiments. Our early experiments with 3NN examples, using geometric restrictions but not a steady-state loop, suggested that choices of \( u_{ij} \) leading to valid configurations often have regions of large intersection area between the translated \( U_{ij} \)'s. For that experiment we tested 30 4CN and 4NN examples, using method (2). A solution was found without subdividing in 66% of the 30 examples.

Once we had the steady-state restriction process in place, we ran another experiment using 30 test examples for \( 4 \leq k \leq 6 \). The results appear in Table 4.1.

Table 4.1 shows that maximizing the area of intersection (heuristic (2)) performs best, solving 60% of the problems without subdivision. Maximizing convexity (heuristic (3)) solved 36% of the problems, and randomization (heuristic (1)) only 20%.

Although the intersection method usually outperforms the convexity method, there are cases in which convexity is able to find a solution without subdivision when intersection cannot. For example, Figure 4.3 shows three \( U_{0j} \) polygons for the example of Figure 4.1.
<table>
<thead>
<tr>
<th>Type</th>
<th>Number of Examples</th>
<th>Random</th>
<th>Intersection</th>
<th>Convexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>4CC</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4NN</td>
<td>13</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5CC</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5CN</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5NN</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6NN</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Totals:</td>
<td>30</td>
<td>6</td>
<td>18</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 4.1: Number of solutions found without subdivision

The three polygons *almost* have a triple intersection when \( u_{ij} \) is chosen by the intersection method. When the convexity method is used for the same example, Figure 4.4 shows that a triple intersection results. Note in Figure 4.4 that the algorithm has matched each pair of \( U_{ij} \) polygons along one side in an attempt to maximize the convexity of the union of each pair.

The grid analysis method for finding an approximate maximum for intersection appears to be effective. However, many set operations are performed in order to choose a single \( u_{ij} \) shrinking point. We would like a method which is as effective but is faster (see Section 10.2.1). Our initial experiments with inexpensive methods for encouraging large intersection area (such as aligning centroids and bounding box corners (see Section 4.2.2)) suggest that these faster methods are not as effective as grid analysis.

### 4.3 Overlap Reduction Evaluation

The disadvantage of the greedy evaluation method of Section 4.2 is that it cannot act simultaneously on the \( U_{ij} \)s. Thus, our choices of shrinking points for the \( U_{ij} \)s do not necessarily produce a consistent set of points leading to a valid configuration. This limitation can be overcome by approximating the configuration spaces by their convex hulls.

The method we discuss here is based on Milenkovic’s *containment linear program*, called the CLP (see Section 3.4 and [Mil, DM95]). To create the CLP, temporarily replace every element \( U_{ij} \) of \( U \) by its convex hull \( H(U_{ij}) \). The set of constraints

\[
t_j - t_i \in H(U_{ij}) \quad 1 \leq i < j \leq k
\]

can be expressed as a linear program.

The use of CLP as an evaluator predates the use of the CLP for restriction. If the CLP is infeasible, then no solution exists. However, if the CLP is feasible, then an arbitrary feasible solution may or may not be a valid configuration. If \( U_{ij} \in U \) is not convex, then it is possible that \( t_j - t_i \) lies in the convex hull \( H(U_{ij}) \), but not in \( U_{ij} \) itself. In that case, the configuration is feasible with respect to the CLP, yet \( P_i + t_i \) and \( P_j + t_j \) overlap.

We suspected that applying an *overlap reduction* algorithm to the overlapping configuration would greatly improve this evaluation method. We tested this conjecture by directly invoking an overlap reduction module which was developed for another task within
our project, Li and Milenkovic have a method for overlap reduction based on linear programming [Li94, ML, LM93c, LM93a]. This was recently generalized and improved by Milenkovic\textsuperscript{2}. We decided to apply this algorithm to improve the performance of the “naive” CLP. Thus, our first attempt directly invoked the overlap reduction module for each overlapping configuration. This was a breakthrough for our containment work. Previously, the CLP algorithm had only been fast for three or fewer polygons. We found that reducing overlaps allowed this algorithm to solve containment, for up to ten polygons, often without subdivision.

However, there was still room for improvement. The overlap reduction module operates directly on the original (i.e. unrestricted) $U_{ij}$s, and so it cannot take into account the restrictions that SSGR or SSLPR find for $U$. Our initial success motivated Milenkovic to develop a new overlap resolution algorithm as described in [MD95]. We briefly summarize the key features of this algorithm here. Full details appear in [MD95].

Finding an overlap minimum, even a local minimum, is a nonconvex problem. Fortunately, it is possible to rapidly find a local minimum in practice through iterated linear programs. For the problems we encounter in practice, minimization appears to require at most two or three iterations. The linear program used in this case is an extension of the CLP called the overlap linear program (OLP). Building the OLP requires constructing a set of inner convex approximations to the $U_{ij}$s, in a fashion similar to Li’s use of a locality heuristic for compaction [Li94]. The OLP minimizes the maximum distance from the $U_{ij}$ to those inner approximations which simultaneously satisfy the CLP.

\textsuperscript{2}The implementation was done by Iraklis Kourtidis, a Harvard undergraduate.
This evaluation technique is very powerful. Like the greedy evaluation method, it can sometimes solve containment problems without any subdivision. It is especially adept, for obvious reasons, at finding a valid configuration when a “loose-fitting” one exists. We defer further discussion of its performance to Section 5.3.

4.4 Evaluation of a Restricted $\mathcal{U}$

A containment algorithm based on CLP evaluation can solve a containment problem using a single CLP evaluation if every $U_{ij}^{\ast} \in \mathcal{U}$ is convex. In this section we specify a condition under which a single CLP evaluation can solve a containment problem for a restricted $\mathcal{U}$.

Following the notation of Chapter 3, in Theorem 4.4.1 below, we denote the replacement of $U_{ij}$ in a valid restriction of $\mathcal{U}$ by $U_{ij}^{\ast}$, and the collection of $U_{ij}^{\ast}$s by $\mathcal{U}^\ast$. We also denote the convex hull of $U_{ij}$ by $H(U_{ij})$ and we use $U_{ij}^{\text{orig}}$ to denote $U_{ij}$ before any restriction has been applied to $\mathcal{U}$.

**Theorem 4.4.1** If, for every $U_{ij}^{\ast} \in \mathcal{U}^\ast$, $H(U_{ij}^{\ast}) \subseteq U_{ij}^{\text{orig}}$, then a single CLP evaluation solves the containment problem for $\mathcal{U}^\ast$.

**Proof:** If the CLP is infeasible, then the containment problem for $\mathcal{U}^\ast$ is infeasible. If the CLP is feasible, then because, for every $U_{ij}^{\ast} \in \mathcal{U}^\ast$, $H(U_{ij}^{\ast}) \subseteq U_{ij}^{\text{orig}}$, every point of the form

Figure 4.4: Common intersection after maximizing convexity of the union
$t_j - t_i$ returned by the CLP is within $U_{ij}^{\text{orig}}$. In this case, the configuration $(t_0, t_1, \ldots, t_k)$ is valid.

This result has two important implications. We show in Section 4.5.3 that Theorem 4.4.1 can be used to guarantee that incorporating restrictions into a CLP-based algorithm does not increase the worst-case number of subdivisions.

In addition, Theorem 4.4.1 provides a test which can be used to decide when to switch from a direct to an indirect approach within a restrict/evaluate/subdivide containment algorithm. Inside any evaluator, one can test if the conditions of Theorem 4.4.1 hold. If so, one can call a CLP evaluator which tests $t_j - t_i$ against $U_{ij}^{\text{orig}}$ to solve the containment problem for the current hypothesis.

## 4.5 Subdivision

We now turn to a discussion of subdivision techniques. Section 4.5.1 defines subdivision and discusses goals of subdivision. Subsequent sections discuss specific subdivision methods. Section 4.5.2 describes our size-based method. Section 4.5.3 discusses combinatorially-based methods. Section 4.5.4 gives a new distance-based method.

### 4.5.1 Definition and Goals

**Subdivision** divides the hypothesis into two sub-hypotheses. Specifically, subdivision splits one $U_{ij} \in \mathcal{U}$ into two parts $U_{ij}^0$ and $U_{ij}^\prime$. It creates two sub-hypotheses by replacing $U_{ij}$ with $U_{ij}^0$ or $U_{ij}^\prime$. That is, it creates $\mathcal{U}' = (\mathcal{U} \setminus \{U_{ij}\}) \cup \{U_{ij}^0\}$ and $\mathcal{U}'' = (\mathcal{U} \setminus \{U_{ij}\}) \cup \{U_{ij}^\prime\}$.

Subdivision can have a variety of goals:

1. reduce the “size” of the $U_{ij}$s, which can be interpreted as the number of vertices, the number of reflex vertices, the number of components, the area, or the maximum dimension of the bounding box, for example;

2. make the convex hull $H(U_{ij})$ “stick out” of $U_{ij}^{\text{orig}}$ less, where $U_{ij}^{\text{orig}}$ is $U_{ij}$ before restrictions have been applied to $\mathcal{U}$. In particular, if we define the maximum overlap of $H(U_{ij})$ to be,

   $$\max_{u \in H(U_{ij})} \text{dist}(u, U_{ij}^{\text{orig}}),$$

   then subdivision should make the maximum overlap of $H(U_{ij})$ and $H(U_{ij}^0)$ smaller;

3. move away from the current configuration;

4. reduce the number of combinatorial possibilities for a valid configuration.

Depending on the containment algorithm, one or more of the goals applies.

The first goal can be accomplished by almost any reasonable splitting algorithm. The goal of reducing the maximum dimension of the bounding box is appropriate for a direct algorithm. Reducing the number of reflex vertices is appropriate for an indirect algorithm which approximates the $U_{ij}$s by their convex hulls. We call a subdivision method which focuses on the first goal a *size-based* subdivision method.

The last three goals make sense if the subdivision method is used together with an evaluation method which always produces a configuration. If the configuration is an overlapping...
one, then subdivision is invoked. The direct, greedy evaluation method of Section 4.2 does not produce a configuration when it answers "maybe". However, the indirect overlap reduction evaluation method of Section 4.3 does produce an overlapping configuration when its answer is "maybe".

The second goal is appropriate for an indirect algorithm which approximates the $U_{ij}$s by their convex hulls. This goal arises because if $H(U_{ij}) \subseteq U_{ij}^{\text{orig}}$, then $U_{ij}$ need not be split further. Recall that Theorem 4.4.1 guarantees that if this condition holds, then a single CLP evaluation solves the containment problem for the current hypothesis. Recall also that Theorem 4.4.1 can be used within any evaluator to identify a situation in which a single CLP evaluation solves the containment problem for the current hypothesis. This implies that the second goal of subdivision can be viewed as an appropriate one for any containment algorithm within our restrict/evaluate/subdivide paradigm.

Both the third and fourth goals have the effect of driving the evaluator "away" from the current overlapping configuration. Without this goal there is no guarantee that a single subdivision will produce sub-hypotheses which are easier to evaluate. Evaluation might investigate the same overlapping tentative solution in one of the sub-hypotheses. Metaphorically, evaluation may get "stuck in a rut," and a large number of subdivisions may be required to move it out of the "rut."

A subdivision method which emphasizes the third goal by maximizing the distance from the current configuration is a distance-based method. A method based on the fourth goal is a combinatorially-based method.

### 4.5.2 Size-Based Method

As discussed above, a size-based subdivision method reduces the "size" of the $U_{ij}$s, which can be interpreted as the number of vertices, the number of reflex vertices, the number of components, the area, or the maximum $(x$ or $y)$ dimension of the bounding box, for example.

We present here one size-based method which reduces the maximal dimension of the bounding boxes of the $U_{ij}$s. Our method operates on the $V \in \{U_{0j}|1 \leq j \leq k\}$ which has the maximum bounding box dimension. It cuts $V$ with either a vertical or horizontal line, depending on the aspect ratio of $V$.

```plaintext
SIZE-SUBDIVIDE(U, k)
V ← element of \{U_{0j}|1 \leq j \leq k\} with maximum bounding box dimension
B ← bounding box of V
/* Assume x is the longest dimension of B */
U' ← \{V \cap \text{left half of } B\} \cup (U \setminus \{V\})
U'' ← \{V \cap \text{right half of } B\} \cup (U \setminus \{V\})
return U', U''
```

### 4.5.3 Combinatorially-Based Methods

Here we discuss two subdivision methods which satisfy the third and fourth goals. Each moves away from the current configuration, and does so in a way that reduces the number of combinatorial possibilities. Both can be used with an indirect evaluation method which approximates the $U_{ij}$s by their convex hulls. The first part of Section 4.5.3 discusses a method which extends an edge of a $U_{ij}$. This method was designed by Milenkovic to work
with the CLP evaluator [Mil, DM95]. The second part of Section 4.5.3 describes an edge extension technique, developed by Milenkovic [MD95], which depends on feedback from an overlap reduction evaluator.

Each of these combinatorially-based methods has an associated worst-case number of subdivisions which depends on the number of vertices in the $U_{ij}s$. If restrictions are applied to an hypothesis, then the number of vertices in $U$ may increase. In the third part of Section 4.5.3 we show that if we use the edges of the $U_{ij}^{0}$s instead of the $U_{ij}^{*}$s within the evaluator and subdivision methods, then any valid restriction may be applied to an hypothesis without increasing the worst-case number of subdivisions.

**Extending an Edge**

The subdivision method of [Mil, DM95] selects a $U_{ij} \in U$ and extends an edge $e$ of $U_{ij}$ into a line $L_e$. The line $L_e$ is used to cut the polygon. This method is used with the CLP linear programming evaluator. In the absence of restrictions, the worst-case number of subdivisions is given by Theorem 4.5.1. In Theorem 4.5.1, (and later in Theorem 4.5.2), $LP(a, b)$ is the time required to solve a linear program with $a$ variables and $b$ constraints.

**Theorem 4.5.1 (MD95)** If subdivision is implemented using cutting lines of the form $L_e$ described above, then a $k$NN algorithm which uses CLP evaluation runs in

$$O((mn)^{2k+1}LP(2k, O(2kmn + k^2m^2)))$$

**time.**

Milenkovic’s proof of Theorem 4.5.1 rests on the fact that there are only $O(mn)$ cutting lines of the form $L_e$. The number of hypotheses which must be evaluated is shown to be the product of the number of cutting lines (each of which corresponds to a cutting plane in the $2k$-dimensional configuration space for the hypothesis) times the number of different possible vertices in the $2k$-dimensional configuration space. The number of different possible vertices is $O((mn)^{2k})$ because each vertex is the intersection of $2k$ cutting planes.

The configurations associated with a hypothesis are contained within a convex cell of the $2k$-dimensional configuration space. The cell is bounded by the cutting planes for the hypothesis. This subdivision method produces two sub-hypotheses associated with combinatorially different configurations, because their configurations are inside convex cells bounded by different sets of cutting planes.

**Extending the Closest Edge**

This section summarizes a different subdivision method, designed by Milenkov [MD95], that drives the evaluator to a configuration which is combinatorially different. This algorithm, in combination with SSLPR and a modification of the OLP of Section 4.3, gives containment a provable running time bound, but it sacrifices practicality for the sake of this bound.

The technique receives a point $u_{max}$ from the OLP evaluator. The point $u_{max}$ corresponds to the pair of polygons with the most overlap. The cutting line is chosen to be the extension of the edge of the $U_{ij}$ which is closest to $u_{max}$.

This type of combinatorially-based subdivision requires some modifications to evaluation. The modified OLP has only a single distance constraint for each $U_{ij}$, and this
constraint is determined by the closest edge of $U_{ij}$ to the current value of $t_j - t_i$ inside the convex hull $H(U_{ij})$. We impose an arbitrary constant bound on the number of iterations of overlap minimization to prevent it from solving an exponential number of OLPs.

With this subdivision method, the containment algorithm requires

$$O\left(\frac{(6kmn + k^2m^2)^{2k+1}}{k!} \text{LP}(1 + 2k + k(k + 1)/2, O(kmn + k^2m^2))\right)$$

time [MD95]. Here $O((6kmn + k^2m^2)^{2k+1}/k!)$ is the number of local minima visited by the algorithm, and $\text{LP}(1 + 2k + k(k + 1)/2, O(kmn + k^2m^2))$ is the time for an evaluation.

If we modify the containment algorithm, we can claim the following running time, which has an exponent of $2k$ instead of $2k + 1$:

**Theorem 4.5.2 ([MD95])** There exists a $k$NN algorithm which uses combinatorially-based subdivision and requires

$$O\left(\frac{(6kmn + k^2m^2)^{2k}}{(k-2)!} \text{LP}(2k, 6kmn + k^2m^2)\right)$$

time.

Milenkovic’s proof is sketched in [MD95]. For the proof, the containment algorithm’s notion of a hypothesis is modified. Instead of using $U$ as a hypothesis, it uses $F = \{F_{ij}, 1 \leq i < j \leq k\}$, where $F_{ij}$ begins as the convex hull of the original $U_{ij}$. All restriction and subdivision acts on $F$ without modifying the $U_{ij}$ polygons. The restriction is basically the SSLPR of Section 3.4, but we do not replace $F_{ij}$ by $H(F_{ij} \cap U_{ij})$, and, as a result, restriction automatically terminates after one iteration. Finally, when we split a hypothesis, we immediately restrict both sub-hypotheses before recursing. Evaluation acts on the $F_{ij} \cap U_{ij}$ polygons, and the “splitting edge” is selected from one of these (as usual).

This may seem like a small improvement over the “simple”

$$O((mn)^{2k+1} \text{LP}(2k, O(kmn + k^2m^2))))$$

I.P-based algorithm [DM95]: the exponent on $mn$ is smaller but the number of variables is larger. However, the new algorithm visits only local overlap minima (inside convex cells bounded by cutting lines). There are reasons to expect the number of such minima to be independent of $m$ and $n$ and depend only on an exponential function in $k$.

**Incorporating Restrictions**

A containment algorithm which uses either of these combinatorially-based methods has an associated worst-case number of subdivisions which depends on the number of vertices in the $U_{ij}$s. If restrictions are applied to a hypothesis, then the number of vertices in $U$ may increase, thus increasing the worst-case number of subdivisions. We show here that, in each case, the same small modifications to evaluation and subdivision allow us to use restrictions without increasing the worst-case number of subdivisions.

These modifications are based on Theorem 4.4.1. The first modification is within the linear programming evaluator. We test each $t_j - t_i$ for inclusion in $U_{ij}^\text{orig}$, not $U_{ij}^\ast$. The second modification is that the cutting lines used for the subdivision are extensions of the edges of the $U_{ij}^\text{orig}$s, not the $U_{ij}^\ast$ s. These modifications preserve the running times of the algorithms, as shown by the following theorem and corollary.
Theorem 4.5.3 Theorem 4.5.1 holds for $U^*$ if the evaluator tests each $t_j - t_i$ for inclusion in $U_{ij}^{\text{orig}}$, and subdivision uses extensions of the edges of the $U_{ij}^{\text{orig}}$s.

Proof: Consider nonconvex polygon $U_{ij}^{\text{orig}}$. In the algorithm referred to in Theorem 4.5.1, in the worst case, $U_{ij}^{\text{orig}}$ is subdivided at every reflex vertex. The result is a set of convex polygons. Now use the same partitioning lines to subdivide $U_{ij}^{+} \subseteq U_{ij}^{\text{orig}}$. Each subpolygon of $U_{ij}^{+}$ is inside exactly one convex subpolygon of $U_{ij}^{\text{orig}}$; hence the convex hull of each subpolygon of $U_{ij}^{+}$ is in $U_{ij}^{\text{orig}}$. Now Theorem 4.4.1 guarantees that a CLP evaluation which tests $t_j - t_i$ for inclusion in $U_{ij}^{\text{orig}}$ solves the containment problem for $U^*$. Thus, the number of subdivisions for the worst case is the same.

Corollary 4.5.4 Theorem 4.5.2 holds for $U^*$ if the evaluator tests each $t_j - t_i$ for inclusion in $U_{ij}^{\text{orig}}$ and subdivision uses extensions of the edges of the $U_{ij}^{\text{orig}}$s.

Theorem 4.5.3 provides a way to incorporate restrictions into the exact, indirect $k$NN algorithm of [DM95, Mil] without affecting the worst-case running time. The subdivision method which results from applying Corollary 4.5.4 to the method associated with Theorem 4.5.2 appears in [MD95].

4.5.4 Distance-Based Method

Distance-based sub-division attempts to move the tentative configuration as far as possible from the current configuration. As before, let $P_i + t_i$ and $P_j + t_j$ be a pair of polygons with maximum overlap under the current configuration. The subdivision algorithm selects a line $L$ through $t_{ij} = t_j - t_i$ and uses $L$ to cut $U_{ij}$ into $U_{ij}^t$ and $U_{ij}^r$. It selects the line $L_{\text{max}}$ that maximizes the minimum distance $d$ from $t_{ij}$ to $H(U_{ij}^t)$ and $H(U_{ij}^r)$. This guarantees that all points within distance $d$ of $t_{ij}$ are removed from further consideration in the algorithm (see Figure 4.5). In other words, it eliminates some of the overlapping configurations which

![Figure 4.5: Distance-based subdivision](image-url)
are “close to” the current overlapping one. The line $L_{\text{max}}$ can be computed in $O(n_{ij} \log n_{ij})$ time, where $n_{ij} = |U_{ij}|$, using a fairly intricate sweep algorithm. We summarize here a simpler and more practical approximate algorithm with essentially the same order running time.

First compute the visibility polygon\(^3\) $V$ for $H(U_{ij}) \cap \overline{U_{ij}}$ with respect to $t_{ij}$. If $t_{ij}$ cannot see the boundary of $H(U_{ij})$ in any direction, then the maximum distance is zero, and the algorithm chooses the line $L$ through the nearest point to $t_{ij}$ on the boundary of $U_{ij}$. Otherwise, let $ab$ be an arc of the boundary of $H(U_{ij})$ that is visible from $t_{ij}$. Parameterize $ab$ by piecewise linear $\gamma(\delta)$ for $\delta \in [0, 1]$. Let the cut line be $L(\delta) = \gamma(\delta)t_{ij}$; it partitions $U_{ij}$ into $U_{ij}^l$ and $U_{ij}^r$. Let $U_{ij}^l$ be the portion of $U_{ij}$ containing $a$, and $U_{ij}^r$ be the portion of $U_{ij}$ containing $b$. For each value of $\delta$, let $d'(\delta)$ and $d''(\delta)$ be the distances of $t_{ij}$ to $H(U_{ij}^l)$ and $H(U_{ij}^r)$, respectively.

**Theorem 4.5.5** For $\epsilon \geq 0$ and $\delta \in [0, 1]$, $d''(\delta + \epsilon) \leq d''(\delta)$ and $d'(\delta + \epsilon) \geq d'(\delta)$.

**Proof:** Let $p$ be the first point beyond $t_{ij}$ where the ray $\gamma(\delta)t_{ij}$ intersects $V$. Similarly, let $q$ be the first point beyond $t_{ij}$ where the ray $\gamma(\delta + \epsilon)t_{ij}$ intersects $V$. We show that $d''(\delta + \epsilon) > d''(\delta)$ cannot occur. For $\delta$, let $r$ be the closest point of $H(U_{ij}^r)$ to $t_{ij}$. We show that $r \in H(U_{ij}^r)$ for $\delta + \epsilon$. Now, all points of $U_{ij}$ on or to the right of $L(\delta + \epsilon)$ are in $H(U_{ij}^r)$ for $\delta + \epsilon$. The points of $U_{ij}$ on or to the right of $L(\delta)$ are on or to the right of $\gamma(\delta)t_{ij}q$. Let $w_1$ be the set of points on or to the right of $\gamma(\delta)t_{ij}q$. Therefore, all points of $U_{ij}$ on or to the right of $L(\delta)$ are on or to the right of $\gamma(\delta)t_{ij}q$. Because $t_{ij}$ can see all points on the arc $ab$, no points of $U_{ij}$ are in the interior of $w_1$. Therefore, all points of $U_{ij}$ on or to the right of $L(\delta + \epsilon)$ are also on or to the right of $L(\delta + \epsilon)$. If $r$ is a point of $U_{ij}$, then it must therefore be in $H(U_{ij}^r)$ for $\delta + \epsilon$. If $r$ is not a point of $U_{ij}$, then it is on a line between two points of $U_{ij}$, and thus is also in $H(U_{ij}^r)$ for $\delta + \epsilon$. Therefore, $d''(\gamma + \epsilon) \leq d''(\gamma)$. A similar argument establishes that $d'(\gamma + \epsilon) \geq d'(\gamma)$.

Because $d'(\gamma)$ increases monotonically and $d''(\gamma)$ decreases monotonically, the maximum distance can be found to any degree of accuracy by binary search on $\gamma$. The running time of the practical algorithm is $O(n_{ij} \log n_{ij} + n_{\text{arcs}} \log \epsilon^{-1} \log n_{ij})$, where the accuracy is $\epsilon$ and the number $n_{\text{arcs}}$ of arcs is $\leq$ the number of components of $U_{ij}$, which is generally much less than $n_{ij}$.

Subdivision is only invoked if the OLP fails to resolve all overlaps; in this case, it is often true in practice that $t_{ij} - t_i$ is very close to the boundary of $U_{ij}$. It is often true that the overlap is between two polygons, as opposed to between a polygon and the container. Thus, the situation often resembles that shown in Figure 4.6. In this figure, the shaded regions are the components of $U_{ij}^*$.

In Section 5.3.2 we show that distance-based subdivision, together with the overlap reduction evaluation technique, allow our hybrid algorithm to detect infeasibility better than the naive CLP-based exact algorithm.

---

\(^3\)This can be found in $O(n)$ time [GA81, Lee83, JS87], or in $\Omega(n \log n)$ time if the polygon has holes [OR87]. (Restrictions tend to fragment $\mathcal{U}$, so an individual $U_{ij}$ may have many components; this means holes may be encountered.)
4.6 Summary and Conclusions

In this chapter, we presented several evaluation and subdivision methods. We offer two evaluation methods. Our greedy evaluator operates directly on the configuration spaces and employs SSGR. The key to its effectiveness is a good choice of a shrinking point for each $U_{ij}$. Selecting a point which maximizes the intersection area of translated $U_{ij}$s appears to work well; in our experiments this strategy solves 60% of the containment problems without any subdivision.

The other evaluation method is an overlap reduction evaluator. This is based on improvements to an indirect containment linear program (CLP) evaluator. A configuration which is feasible for the CLP evaluator may be overlapping. If the overlaps can be eliminated, a valid configuration can be found. An overlap reduction evaluator is sometimes able to solve containment problems without any subdivision. It is especially adept at finding a valid configuration when a “loose-fitting” one exists.

Our discussion of evaluation methods concludes with the observation that, if the convex hull of every $U_{ij}$ is within its unrestricted $U_{ij}$, then a single CLP evaluation solves the containment problem for the associated hypothesis. This test can be used to decide when to stop subdividing a hypothesis inside any containment algorithm which is based on our paradigm.

We offer four different possible subdivision goals: 1) reduce the “size” of the $U_{ij}$s, 2) make the convex hull of a $U_{ij}$ protrude less from its original $U_{ij}$ (this is motivated by the observation of the previous paragraph), 3) move away from the current configuration, and 4) reduce the number of combinatorial possibilities for a valid configuration. We present a size-based subdivision technique which abides by the first goal and reduces the maximal dimension of the bounding boxes of the $U_{ij}$s. We discuss two combinatorially-based methods, developed by Milenkovic, which satisfy the third and fourth goals. We show that a containment algorithm which uses either of these can use restrictions without increasing
the worst-case number of subdivisions. We introduce a distance-based subdivision method which satisfies the third goal. This practical method eliminates some of the overlapping configurations which are "close to" the current overlapping one.
5.1 Introduction

Our techniques can be combined to form a variety of two-dimensional translational containment algorithms which follow the restrict/evaluate/subdivide paradigm introduced in Section 2.4. As we noted in Section 2.4, termination of the algorithm depends on the compatibility of the techniques. In this chapter we present two ways of combining techniques into $k$NN containment algorithms.

Our first algorithm is an approximate algorithm. That is, it finds an $\epsilon$-approximate configuration for $k$NN (refer to Definition 2.4 in Section 2.2). The algorithm uses the geometric restrictions (SSGR) of Section 3.3, the greedy evaluator of Section 4.2, and the size-based subdivision method of Section 4.5.2. All of these methods are direct, and so this algorithm is a direct, approximate algorithm.

The approximate algorithm has additional structure beyond our paradigm. It uses the containment characterization of Section 2.7 to create a set of $k$ boundary-restricted containment subproblems. Each of the subproblems is solved using restriction, evaluation, and subdivision. Our approximate algorithm appears in [DM95, DM] as well as in this chapter.

The second algorithm is an exact algorithm. It uses both the geometric restrictions of Section 3.3 and the linear programming restrictions (SSLPR) of Section 3.4. It uses the overlap reduction evaluation of Section 4.3. It employs the distance-based subdivision of Section 4.5.4, so that when the polygons of a hypothesis are subdivided into more convex pieces, this is done with the goal of moving the configuration as far as possible from the current configuration. We consider this exact algorithm to be a hybrid algorithm because it uses an indirect evaluation method, and both direct and indirect methods for restriction. Our hybrid algorithm is also described in [MD95]. This algorithm currently yields our best practical results, primarily due to its powerful evaluation method.

Ironically, the approximate algorithm operates directly on the actual $U_{ijs}$, whereas the exact algorithm uses approximations of the $U_{ijs}$.

Both of our algorithms have practical running times for small numbers of items. The container and items may be nonconvex. Most of our examples are drawn from apparel
For our apparel examples, the number of vertices in the container ranges from 100 to 300, and the number of vertices in each item ranges from 4 to 100. We produce a feasible containment problem from an apparel layout by removing a group of neighboring items from the layout. To create an infeasible problem, we move some of the remaining items a small distance into the container.

For ten or fewer items, the hybrid algorithm typically requires several minutes or less on a 50MHz SPARCstation. The current implementation of approximate algorithm is fast for up to four items. For \( k > 4 \), the current implementation of SSGR, which is based on polygon set operations, slows the algorithm down considerably. Increasing the speed of our SSGR implementation is a subject of future work (see Section 10.2.1). In our experiments, SSGR evaluates fewer hypotheses than the faster, hybrid algorithm, and so we believe the approximate algorithm has the potential to be competitive with the hybrid algorithm.

Implementation note: Both of our algorithms require robust geometric techniques [LM93b, MN89, Mil93a, Mil93b] in order to function well in practice.

5.1.1 Overview

This chapter contains one section for each of our two containment algorithms. Section 5.2 discusses our approximate algorithm and gives experimental results. Section 5.3 describes our hybrid algorithm and presents experimental results. Section 5.4 summarizes this chapter and states conclusions.

5.2 An Approximate Containment Algorithm

5.2.1 Introduction

We present a numerically approximate algorithm for finding an \( \epsilon \)-approximate configuration for \( k \)NN (refer to Definition 2.4 in Section 2.2). The value of \( \epsilon \) is an input to the algorithm. In industrial cutting applications, the containment solution acts as a guide to a machine cutting out polygonal shapes from a sheet of material. If one chooses \( \epsilon \) to be a fraction of the cutter’s accuracy, then the solution to the approximate containment problem is sufficient for industrial purposes. In fact, for some cutting applications, CAD vendors round polygon vertex coordinates to integers. In such cases, an overlap of up to one unit, corresponding to .01 inches is acceptable. For the applications we consider, the ratio of the largest value to the smallest nonzero value is usually under 10,000; this makes an approximate algorithm a practical choice.

Given a containment problem, we use our characterization of translational containment to create a collection of \( k \) containment subproblems (see Section 2.7). We solve each subproblem by first applying our steady-state geometric restrictions (see SSGR in Section 3.3) and then using the greedy evaluator of Section 4.2. If necessary, we subdivide the configuration space, using size-based subdivision, to generate new subproblems. The running time of our algorithm is \( O\left(\left(\frac{1}{\epsilon}\right)^k \log \left(\frac{1}{\epsilon}\right) k^6 s \log s\right) \), where \( s \) is the largest number of vertices in a polygon restricted by SSGR (see the discussion of size analysis in Section 3.2.4). In the worst case \( s \) can be exponential in the size of the input, but, in practice, it is usually not more than quadratic.

In a sense, our algorithm substitutes subdivision for the iteration used in Avnaim’s exact 3NN algorithm (see [Avn89] and the discussion of related work in Section 2.6).
tice, much iteration is required in an exact algorithm before hitting the right combination of edges. If the answer is “no”, one must check all combinations of edges. Our SSGR alone can often detect infeasibility (see Section 3.3.5). SSGR together with our greedy evaluation method can often find a valid configuration without requiring the algorithm to use subdivision (see Section 5.2.3). While restriction and subdivision often perform better than iteration, iteration can have a much better theoretical worst-case running time bound. Even when subdivision is unnecessary, the running times of our algorithm are in \(O(m^{28}n^{24} \log mn)\) for 3CN and in \(O(m^{60}n^{40} \log mn)\) for 3NN. These seem much worse than the iteration-based methods, but in terms of the size \(s\) of the largest polygon generated by SSGR, the running times are both in \(O(s \log s)\) as opposed to \(O(s^5 \log s)\) for the iteration-based algorithms. As mentioned in Section 2.6, it is not clear that the worst-case results for consecutive polygon operations are attainable. The tests we have run indicate that the running times of our algorithm are much faster than the best times we would estimate for the iteration-based solutions to 3NN.

The high-level description of our algorithm in the second part of this subsection draws its structure from our characterization of containment, because the characterization allows us to decompose \(k\)NN into a set of subproblems. The algorithm solves each subproblem by first restricting \(U\) using SSGR, testing for a solution using the greedy evaluation of Section 4.2, and subdividing the configuration space if a solution is not found. Section 5.2.2 establishes the correctness and running time of our algorithm. Section 5.2.3 discusses our implementation and also presents containment examples.

### Applying the Characterization

Our philosophy is to avoid iteration as much as possible because iteration contributes to the worst-case cost of the algorithm, particularly in terms of \(s\)-analysis. For instance, Avnaim and Boissonnat’s algorithm for 3NP [AB87, Avn89] has running time \(O(s \log s)\), and Avnaim’s algorithm for 3NN [Avn89] has running time \(O(s^5 \log s)\). The algorithm for 3NP is faster because it avoids iterating over the edges and vertices of polygons. Like the algorithm for 3NP, we replace iteration with restriction as much as possible.

There is another type of iteration one must consider. This iteration is based on the characterization of Theorem 2.7.1. To apply the entire characterization, one needs to choose contacting pairs in a number of ways which is exponential in \(k\), roughly \(k^{4k}\). If one were to generalize Avnaim’s algorithm for 3NN [Avn89] to an algorithm for \(k\)NN, the factor of \(k^{4k}\) would not be a significant part of the cost compared to the factor dependent on \(m\) and \(n\) (roughly \(m^{2k^3}mn^{2k}k\)). For our approach, however, a factor of \(k^{4k}\) is significant.

It is possible to use restriction and subdivision and avoid iterating over any contacting pairs. In fact, this is what we initially implemented. However, we feel that considering contacting pairs is potentially very useful. For instance, if we invoke a condition weaker than the second part of Theorem 2.7.1 to assume that a particular \(P_j\) is in contact with \(P_0\), then we can replace \(U_{0,j}\) with its boundary. Note that this boundary restriction covers the cases where \(P_0\) has either a single or a double contact with some \(P_j\). Although we restrict \(U_{0,j}\) to its boundary, we are not iterating over vertices or edges of \(U_{0,j}\), unlike Avnaim’s 3NN algorithm.

This partial characterization implies a significant restriction. However, the disadvantage is that we must iterate over all \(k\) choices of \(j\) by solving \(k\) subproblems\(^4\). Our current

---

\(^4\) However, as noted in Section 2.7, these subproblems can be solved in parallel. In our current implemen-
conjecture is that iterating over $O(k)$ choices of (partial) characterization will allow us to avoid subdivision often enough to offset the extra factor of $k$ in cost. We present data in Section 5.2.3 to support this conjecture in the case of $k = 4$.

The boundary restriction does not add an extra factor of $k$ in cost for containment problems in which all the items are identical. In this case, all $k$ subproblems are identical, and so only one subproblem must be solved.

The Algorithm

In the algorithm below, $\mathcal{P}$ and $\mathcal{U}$ are set initially to the values given in Section 2.2. If at any point a valid configuration is found, execution terminates with success. We assume that the tolerance test TOO-SMALL() is used within GREEDY-EVALUATE(). The $\epsilon$ used in TOO-SMALL() is the same $\epsilon$ used within SSGR. BUILD-CONFIGURATION is given in Section 4.2.1. Note that in order to perform a boundary restriction on a $U_{0,j}$, we need only intersect it with the boundary of $U_{0,j}^{\text{orig}}$. $\mathcal{U}^j$ represents a copy of $\mathcal{U}$ for the $j$th processor, and $U_{0,j}^j$ is processor $j$’s configuration space for $P_j$ with respect to $P_0$.

```
APPROX-CONTAIN($\mathcal{U}$, $k$)
$\mathcal{U}^{\text{orig}}$ ← $\mathcal{U}$
$\mathcal{U}$ ← SSGR($\mathcal{U}$, $k$)
if (some $U_{ij} \in \mathcal{U} = \emptyset$) return NO
if (TOO-SMALL($\mathcal{U}$, $k$))
    return BUILD-CONFIGURATION($\mathcal{U}$, $k$)

/* Search for a boundary-restricted solution */
for (j = 1 to $k$) do in parallel
    $\mathcal{U}^j$ ← $\mathcal{U}$
    $U_{0,j}^j$ ← $U_{0,j} \cap$ BOUNDARY-OF($\mathcal{U}^{\text{orig}}$)
    return SOLVE-SUBPROBLEM($\mathcal{U}^j$, $k$)
```

```
SOLVE-SUBPROBLEM($\mathcal{U}$, $k$)
$\mathcal{U}$ ← SSGR($\mathcal{U}$, $k$)
if (some $U_{ij} \in \mathcal{U} = \emptyset$) return NO
status ← GREEDY-EVALUATE($\mathcal{U}$, $k$)
if (status = YES) return YES
if (status = NO) return NO
$\mathcal{U}'$, $\mathcal{U}''$ ← SIZE-SUBDIVIDE($\mathcal{U}$, $k$)
return SOLVE-SUBPROBLEM($\mathcal{U}'$, $k$) or SOLVE-SUBPROBLEM($\mathcal{U}''$, $k$)
```

```
TOO-SMALL($\mathcal{U}$, $k$)
$\mathcal{V}$ ← element of $\{U_{0,j}|1 \leq j \leq k\}$ with maximum bounding box dimension
$\mathcal{B}$ ← bounding box of $\mathcal{V}$
if (height($\mathcal{B}$) $< \frac{1}{\sqrt{2}}$ and width($\mathcal{B}$) $< \frac{1}{\sqrt{2}}$) return YES
return NO
```

In the approximation we solve them sequentially.
5.2.2 Analysis

Correctness

Claim 5.2.1 If a valid configuration exists, then the algorithm will find an \(\epsilon\)-approximate solution.

Proof: The algorithm converges because each subdivision reduces the height or width of the bounding box of \(V\) by a factor of two, stopping when the height and width reach the tolerance \(\frac{1}{\sqrt{2}}\).

If a solution exists, then our characterization guarantees that one of the boundary-restricted subproblems has a solution. We claim that our algorithm does not throw away any solutions to these subproblems. This is because Theorem 3.3.1 guarantees that SSGR is a valid restriction.

Now we argue that any solution which is found is an \(\epsilon\)-approximate solution. If GREEDY-EVALUATE finds a solution, then Theorem 4.2.1 guarantees that solution contains no overlap; hence it is a valid configuration. A valid configuration is an \(\epsilon\)-approximate solution.

The only other way the algorithm can find a solution is through a call to BUILD-CONFIGURATION. This occurs when the height and width of the bounding box of \(V\) are both \(< \frac{1}{\sqrt{2}}\). In that case, we argue that no point of any polygon is more than \(2\epsilon\) inside of any other polygon's boundary. None of the polygons overlap the container, because each \(t_j\) chosen in BUILD-CONFIGURATION is in \(U_0\). Thus, we need only bound the overlap between a pair of polygons \(P_i\) and \(P_j\), \(1 \leq i \neq j \leq k\).

Any \(u_{ij} \in U_{ij}\), having survived SSGR, can be expressed as \(t_j - t_i\), where \(t_i \in U_{0i}\) and \(t_j \in U_{0j}\). If \(P_i\) is placed at \(t_i\) and \(P_j\) is placed at \(t_j\), \(P_i\) and \(P_j\) do not overlap. Now let \(v_i\) and \(v_j\) be the points of \(U_{0i}\) and \(U_{0j}\) chosen in BUILD-CONFIGURATION. Let \(D(p, q)\) denote the distance between points \(p\) and \(q\). Since the diameter \(d\) of any \(U_0\) is less than \(\epsilon\), we have \(D(v_i, t_i) < \epsilon\) and \(D(v_j, t_j) < \epsilon\). Translating \(P_i\) from \(t_i\) to \(v_i\) can therefore cause a point of \(P_i\) to be at most a distance of \(\epsilon\) inside of \(P_j\). Translating \(P_j\) from \(t_j\) to \(v_j\) can at most double this overlap. Therefore, the maximum overlap between two polygons in the configuration \(\{t_1, t_2, \ldots, t_k\}\) is \(< 2\epsilon\). \(\square\)

Running Time

Here we calculate the asymptotic running time of our algorithm. We assume:

- \(s\) is the largest number of vertices of any polygon generated by SSGR (see the discussion in Section 3.2.4),
- \(2\epsilon\) is the maximum overlap between two polygons,
- \(\alpha\) is the fractional shrinkage in area required by each iteration of SSGR.

Theorem 5.2.2 The running time of our algorithm is in \(O\left(\frac{1}{\epsilon^2}\log\left(\frac{1}{\epsilon}\right)k^6s\log s\right)\).

Proof: Let \(S\) be the number of subdivisions and \(R\) be the running time of SSGR. The running time is dominated by \(Sk^3R\). To find \(S\), we first assume that the maximum height or width of the bounding box of \(V\) is \(q\). The height of the subdivision tree is therefore
\(O(k(\log_2 q - \log_2 \varepsilon))\). The \(k\) term corresponds to the total number of polygons that are subdivided in the worst case. The number of nodes in the subdivision tree is therefore

\[
O(2^{\log_2 \left(\frac{q}{\varepsilon}\right)^k}) = O(2^{\log_2 \left(\frac{1}{\varepsilon}\right)^k}) = S
\]

From Section 3.3.3, \(R\) is in \(O(\log \left(\frac{1}{\varepsilon}\right)^k \cdot s \log s)\).

The total running time is therefore:

\[
S k^3 R = O\left(\left(\frac{1}{\varepsilon}\right)^k \log \left(\frac{1}{\varepsilon}\right)^k \cdot s \log s\right)
\]

\(\square\)

### 5.2.3 Results

This section describes our implementation of the algorithm for approximate containment. We set \(\alpha\) equal to 0.1 and \(\varepsilon\) to 0.5. SSGR is used for \(k > 3\). We have found that for \(k \leq 3\) the algorithm runs so quickly and so rarely requires subdivision that one iteration of geometric restriction is all that is needed.

#### General Results

Table 5.1 contains running times for our algorithm for a variety of examples for \(3 \leq k \leq 10\). All running times given in the table are to the nearest second on a 28 MHz SPARCstation\(^2\). The setup time to calculate \(U_{ij}\) polygons is not included in the running time. Infeasible examples are identified by \((\ast)\). All infeasible examples are ones for which each pair of polygons fits into the container, and the area of the container is larger than the total area of the polygons. Our restriction process enables the algorithm to often detect infeasibility very quickly.

Figure 5.1 through Figure 5.4 show configurations achieved for four examples using our algorithm. The tightness of the layouts is typical. In each figure, the items placed by our containment algorithm have dark shading. Figure 5.1 and Figure 5.2 are 3NN examples. Figure 5.3 is a 4NN example. Figure 5.4 is a 5NN example.

![Figure 5.1: 3NN example: time = 6 seconds](image)

\(^2\)SPARCstation is a trademark of SPARC International, Inc., licensed exclusively to Sun Microsystems, Inc.
<table>
<thead>
<tr>
<th>Type</th>
<th>Time</th>
<th>Type</th>
<th>Time</th>
<th>Type</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3CC</td>
<td>1</td>
<td>3CC</td>
<td>1</td>
<td>3CC</td>
<td>0</td>
</tr>
<tr>
<td>3CC*</td>
<td>0</td>
<td>3CC</td>
<td>1</td>
<td>3CN</td>
<td>3</td>
</tr>
<tr>
<td>3CN</td>
<td>1</td>
<td>3CN</td>
<td>1</td>
<td>3CN</td>
<td>1</td>
</tr>
<tr>
<td>3CN</td>
<td>2</td>
<td>3CN</td>
<td>3</td>
<td>3CN</td>
<td>2</td>
</tr>
<tr>
<td>3CN</td>
<td>2</td>
<td>3CN</td>
<td>1</td>
<td>3CN</td>
<td>1</td>
</tr>
<tr>
<td>3CN</td>
<td>4</td>
<td>3CN</td>
<td>1</td>
<td>3CN*</td>
<td>2</td>
</tr>
<tr>
<td>3CN*</td>
<td>1</td>
<td>3CN*</td>
<td>0</td>
<td>3CN*</td>
<td>2</td>
</tr>
<tr>
<td>3CN*</td>
<td>1</td>
<td>3CN*</td>
<td>0</td>
<td>3CN*</td>
<td>1</td>
</tr>
<tr>
<td>3CN</td>
<td>2</td>
<td>3CN</td>
<td>2</td>
<td>3CN</td>
<td>3</td>
</tr>
<tr>
<td>3CN</td>
<td>4</td>
<td>3CN</td>
<td>2</td>
<td>3CN</td>
<td>1</td>
</tr>
<tr>
<td>3CN</td>
<td>1</td>
<td>3CN</td>
<td>2</td>
<td>3CN</td>
<td>2</td>
</tr>
<tr>
<td>3CN</td>
<td>2</td>
<td>3CN</td>
<td>2</td>
<td>3CN</td>
<td>3</td>
</tr>
<tr>
<td>3CN</td>
<td>2</td>
<td>3CN</td>
<td>1</td>
<td>3NN</td>
<td>5</td>
</tr>
<tr>
<td>3NN</td>
<td>5</td>
<td>4CN</td>
<td>26</td>
<td>4CN</td>
<td>30</td>
</tr>
<tr>
<td>4CN*</td>
<td>3</td>
<td>4CN</td>
<td>5</td>
<td>4CN</td>
<td>19</td>
</tr>
<tr>
<td>4CN</td>
<td>192</td>
<td>4CN</td>
<td>43</td>
<td>4NN</td>
<td>111</td>
</tr>
<tr>
<td>4NN</td>
<td>111</td>
<td>4NN</td>
<td>90</td>
<td>4NN*</td>
<td>10</td>
</tr>
<tr>
<td>5CC</td>
<td>126</td>
<td>5NN</td>
<td>60</td>
<td>5NN*</td>
<td>26</td>
</tr>
<tr>
<td>6NN</td>
<td>93</td>
<td>6NN</td>
<td>300</td>
<td>6NN*</td>
<td>49</td>
</tr>
<tr>
<td>7CC</td>
<td>823</td>
<td>10CC</td>
<td>600</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Running times in seconds ((*) denotes infeasible example)

Figure 5.2: 3NN example: time = 5 seconds
Figure 5.3: 4NN example: time = 90 seconds

Figure 5.4: 5NN example: time = 60 seconds
Table 5.2 below presents running times, in seconds, for the approximate algorithm on a different, small collection of examples for $5 \leq k \leq 7$. These tests are for a 50MHz SPARCstation. For each example, we also give the number of hypotheses examined by the approximate algorithm. Judging the approximate algorithm purely by its running time can be deceptive. Although the algorithm takes anywhere from 3 to 45 minutes in these examples, it is only examining 1 or 2 hypotheses. Most of the running time is consumed by SSGR, whose current implementation is roughly ten times more slow than that of SSLPR. Faster implementations of SSGR will directly lead to faster running times for the approximate algorithm. In fact, for an experiment in Section 5.3.2, the number of hypotheses examined by our fastest (hybrid) algorithm is greater than the number of hypotheses examined by the approximate algorithm. We therefore believe that, given fast enough polygon set operations, the approximate algorithm has the potential to be competitive with the hybrid algorithm.

<table>
<thead>
<tr>
<th>Type</th>
<th>Running Time</th>
<th>Number of Hypotheses</th>
</tr>
</thead>
<tbody>
<tr>
<td>5NN</td>
<td>428</td>
<td>1</td>
</tr>
<tr>
<td>5NN</td>
<td>189</td>
<td>1</td>
</tr>
<tr>
<td>5NN</td>
<td>317</td>
<td>1</td>
</tr>
<tr>
<td>6CN</td>
<td>2540</td>
<td>2</td>
</tr>
<tr>
<td>6CN</td>
<td>2238</td>
<td>1</td>
</tr>
<tr>
<td>7CC</td>
<td>823</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.2: Running time, in seconds, and number of hypotheses examined by approximate algorithm

Solving Subproblems

In our comparison of evaluation heuristics for choosing a $U_{ij}$ shrinking point, we reported in Table 4.1 that the method of maximizing the intersection area solved 60% of the 30 examples without subdivision. Here we maximize the intersection area of translated $U_{ij}$s, and we compare the effect of using $k$ subproblems versus not using subproblems. That is, we evaluate the effectivness of applying a partial containment characterization.

Our examples use $k = 4$. Figure 5.5 shows an example of a feasible 4NN problem solved using the characterization. The darkly shaded items numbered 5, 62, 63, and 61 were placed. Note that three of the four items are in contact with the boundary of the container.

We first compare the sequential cost, in terms of the number of hypotheses examined, for infeasible as well as feasible examples. Results are shown in Table 5.3. For the 46 examples, the subproblem approach examines a total of 162 hypotheses, whereas the non-subproblem approach examines 345 hypotheses. These results suggest that, for $k = 4$, applying the partial characterization to the sequential algorithm is beneficial.

When all the items are identical, the $k$ subproblems generated by the characterization are all identical, and so we need only solve one of them. The results above contain 21 problems for which all items are identical. The remaining 25 problems are for non-identical items. The results for identical item problems are listed separately in Table 5.4. The total number of hypotheses examined in the characterization case is 115, whereas the number
Figure 5.5: 4NN feasible example solved using partial characterization
<table>
<thead>
<tr>
<th>Type</th>
<th>Subproblems</th>
<th>No Subproblems</th>
<th>Type</th>
<th>Subproblems</th>
<th>No Subproblems</th>
</tr>
</thead>
<tbody>
<tr>
<td>4CC</td>
<td>1</td>
<td>1</td>
<td>4CC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CC</td>
<td>1</td>
<td>1</td>
<td>4CN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>1</td>
<td>2</td>
<td>4NN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>3</td>
<td>1</td>
<td>4CC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>1</td>
<td>1</td>
<td>4CN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>3</td>
<td>1</td>
<td>4CN</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>1</td>
<td>1</td>
<td>4CN</td>
<td>1</td>
<td>116</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>1</td>
<td>4CN</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>8</td>
<td>26</td>
<td>4CN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>6</td>
<td>4CN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>1</td>
<td>4CN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>1</td>
<td>4CN</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4CN</td>
<td>2</td>
<td>1</td>
<td>4CN</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>4</td>
<td>4NN</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>2</td>
<td>1</td>
<td>4CN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CN*</td>
<td>5</td>
<td>5</td>
<td>4CN*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4CN*</td>
<td>7</td>
<td>14</td>
<td>4CN*</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>4CN*</td>
<td>1</td>
<td>11</td>
<td>4CN*</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>4NN*</td>
<td>3</td>
<td>3</td>
<td>4NN*</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CC*</td>
<td>1</td>
<td>1</td>
<td>4NN*</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4NN*</td>
<td>1</td>
<td>1</td>
<td>4CC*</td>
<td>65</td>
<td>93</td>
</tr>
<tr>
<td>4NN*</td>
<td>2</td>
<td>2</td>
<td>4CN*</td>
<td>3</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 5.3: Number of hypotheses examined by sequential algorithm ((*) denotes infeasible example)
examined in the non-characterization case is 174. For the non-identical item problems, the total number of hypotheses examined in the characterization case is 47, whereas the number examined in the non-characterization case is 171. Thus, for identical as well as non-identical item problems, the characterization examines fewer hypotheses for \( k = 4 \) in our experiments.

<table>
<thead>
<tr>
<th>Type</th>
<th>Subproblems</th>
<th>No Subproblems</th>
<th>Type</th>
<th>Subproblems</th>
<th>No Subproblems</th>
</tr>
</thead>
<tbody>
<tr>
<td>4CN</td>
<td>1</td>
<td>1</td>
<td>4CN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>1</td>
<td>4CN</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4CN</td>
<td>2</td>
<td>1</td>
<td>4CN</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>4</td>
<td>4NN</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>2</td>
<td>1</td>
<td>4CN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CN*</td>
<td>5</td>
<td>5</td>
<td>4CN*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4CN*</td>
<td>7</td>
<td>14</td>
<td>4CN*</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>4CN*</td>
<td>1</td>
<td>11</td>
<td>4CN*</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>4NN*</td>
<td>3</td>
<td>3</td>
<td>4NN*</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4NN*</td>
<td>2</td>
<td>2</td>
<td>4CN*</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4CC*</td>
<td>65</td>
<td>93</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: Number of hypotheses examined for identical items (** denotes infeasible example)

Intuitively, we expect that a single boundary-restricted subproblem will not examine more hypotheses than the non-subproblem approach because we are eliminating so much of the search space. Often this is the case. However, our experiments reveal that it is possible for a boundary-restricted subproblem to evaluate (a few) more hypotheses than a problem in which the boundary restriction is not applied. We believe that this phenomenon is related to the evaluation method. Recall that our evaluation method maximizes the intersection area of \( U_{ij} \) polygons. If a general polygon contains polygonal regions plus isolated edges and points, the intersection heuristic only considers the polygonal regions. If no polygonal regions are present, we currently resort to a simple alternative heuristic. Thus, if the boundary restriction causes many of the \( U_{ij} \)'s to consist of only isolated edges and points, we currently lose the power of the evaluation heuristic. Section 10.2.1 suggests a possible remedy for this.

Assessing the cost-effectiveness of applying the partial characterization for \( k > 4 \) is a subject for future work. We believe that, as \( k \) increases, the partial characterization will continue to be cost-effective for identical item problems because only one subproblem must be examined. The partial characterization is less likely to have an advantage in sequential tests for non-identical items as \( k \) increases, due to the increase in the number of subproblems. Once the critical value of \( k \) is identified for which this advantage is lost, a detailed comparison of the parallel cost will be necessary in order to justify using the partial characterization for values of \( k \) larger than the critical one. Section 2.4 discussed one way in which course-grained parallelism can be applied to our algorithm. A queue of hypotheses can be maintained. Each hypothesis represents an independent containment subproblem which is created via subdivision. Whenever a processor is available, it operates on the
next hypothesis in the queue. The subproblems generated by the characterization are also independent from each other, and so the hypotheses they generate are independent and can be added to the queue. Note that applying the characterization creates subproblems which are all available at the start of the algorithm.

We remark that the number of subproblems which must be examined is smaller, in theory, for \(k\)CN examples than for \(k\)NN. This is due to the following theorem and corollary.

**Theorem 5.2.3** If an instance of \(k\)CN for \(k \geq 3\) has a valid configuration, then that instance has a valid configuration in which at least three items are in contact with the boundary of the container.

**Proof:** Consider a valid configuration for an instance of \(k\)CN, for \(k \geq 3\). Milenkovic shows [Mil] that, given a set \(P\) of non-overlapping polygons, at least one of them can move unobstructed to infinity in the direction \((-1,0)\). This “disassembly lemma” works for any direction. Thus, for every direction \(d\), there is some polygon \(P \in P\) such that \(P\) can move to infinity in direction \(d\). Let \(S\) be the subset of polygons in \(P\) which can move to infinity in at least one direction. Let \(S'\) be a minimal subset of \(S\); one which covers all directions using the smallest number of polygons. We now show that \(S'\) contains at least two polygons. First, we observe that either every polygon in \(P\) can move to infinity in some direction (in which case we are done), or at least one polygon \(Q\) cannot. It is easy to show that one convex polygon alone cannot “surround” another convex polygon, and so it takes at least two polygons to prevent \(Q\) from moving to infinity. Thus, \(S'\) contains at least two polygons. Since \(S'\) contains at least two polygons and all directions are covered by polygons in \(S'\), at least two polygons \(P_1\) and \(P_2\) in \(S'\) share a common direction \(d\). This means that we can move both \(P_1\) and \(P_2\) in direction \(d\) until each hits the boundary of the container. Because the direction \(-d\) is independent of direction \(d\) and there are at least three polygons, we can apply the disassembly lemma to \(P \setminus (\{P_1\} \cup \{P_2\})\) to move a third polygon \(P_3\) in direction \(-d\) until it hits the container. \qed

We note that three items is “tight”, because there are sets of four convex polygons for which only three can be moved to infinity.

**Corollary 5.2.4** When solving an instance of \(k\)CN using the partial characterization, examining \(k - 2\) subproblems is sufficient.

Corollary 5.2.4 implies that examining one subproblem is sufficient for \(3\)CN examples. This, however, does not yield the fastest known algorithm for \(3\)CN. Milenkovic [Mil] observed that, for a valid \(3\)CN configuration, each of the three polygons can be moved to the boundary of the container. This forms the basis of his \(O(m^3 \log mn)\) time \(3\)CN algorithm. Below we compare the performance of this very fast \(3\)CN algorithm with our approximate algorithm on \(3\)CN examples.

**Comparison with Exact Algorithm for Convex Items**

Table 5.5 compares running times for Milenkovic’s \(3\)CN exact algorithm [DML94, Mil] with our approximate algorithm on \(3\)CN examples. Infeasible examples are identified by (*). The exact algorithm tests separating line orientations between pairs of convex polygons. Because both algorithms are so fast, timing comparisons are based on 100 calls to
each. For feasible examples, the exact algorithm performs about seven times faster than the approximate algorithm, on average. For infeasible examples, the exact algorithm is only about twice as fast, on average. This is no accident. When the exact algorithm was first designed, the approximate algorithm’s powerful restriction process gave it a considerable advantage over the exact algorithm for infeasible examples. The exact algorithm was redesigned with the goal of detecting infeasibility quickly in mind. The major part of the redesign was a multiple resolution method for evaluating sets of separating lines.

<table>
<thead>
<tr>
<th>Example</th>
<th>Exact</th>
<th>Approximate</th>
<th>Example</th>
<th>Exact</th>
<th>Approximate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>79</td>
<td>6*</td>
<td>15</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>107</td>
<td>7*</td>
<td>21</td>
<td>68</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>171</td>
<td>8*</td>
<td>16</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>230</td>
<td>9*</td>
<td>15</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>55</td>
<td>10*</td>
<td>30</td>
<td>38</td>
</tr>
</tbody>
</table>

Table 5.5: Exact vs. approximate running times for 3CN examples (100 calls each)

**Subdivision**

The vertical and horizontal cut method of Section 4.5.2 has been implemented. For our applications, CAD vendors round polygon coordinates to integers, so an \( \epsilon \) of .5 units is sufficiently small. It is rare for the algorithm to subdivide down to tolerance. Often, no subdivision is required at all to find a valid configuration. This is because we have a heuristic for selecting a shrinking point for each \( U_{ij} \) which works well in practice (see Section 4.2.3). Subdivision is rarely required for infeasible situations, because our restriction process often detects infeasibility early in the algorithm.

**5.3 A Hybrid Containment Algorithm**

**5.3.1 Introduction**

Our hybrid algorithm is the result of significant improvements to the restriction, evaluation, and subdivision techniques of an existing indirect algorithm of Milenkovic [Mil]. Milenkovic’s indirect algorithm, in its original form, was slower than our direct algorithm for infeasible problems, and its practical upper bound on \( k \) was \( k \leq 3 \). The improved indirect algorithm is currently the fastest containment algorithm for \( 4 \leq k < 10 \).

Our first improvement to the indirect algorithm improves its ability to detect infeasibility by proving that restrictions, when used in the indirect algorithm, do not increase the algorithm’s worst-case running time if we perform an appropriate test within the evaluation step and subdivide using an appropriate set of cutting lines. This improvement is discussed in detail in Section 4.5.3. This improvement allows us to use both SSGR (Section 3.3) and SSLPR (Section 3.4) in the algorithm without increasing the worst-case running time. We use some SSGR restriction at the top level, and SSLPR at the hypothesis level.

The second improvement to the indirect algorithm improves the ability of the evaluator to locate a valid configuration. We apply the work of Li and Milenkovic on overlap reso-
olution [Li94, ML, LM93c, LM93a] to improve the evaluation step. Their overlap resolution algorithm is an indirect algorithm. It uses inner convex approximations, an approach which complements the outer approximations of the indirect containment algorithm. The success of this experiment led Milenkovic to develop the overlap reduction evaluator described in Section 4.3. We use that evaluator in the hybrid algorithm.

A combinatorially-based subdivision algorithm yields a running time of

$$O \left( \frac{(6kmn + k^2m^2)^{2k}}{(k-2)!} \text{LP}(2k, 6kmn + k^2m^2) \right)$$

where LP($a$, $b$) is the time to solve a linear program with $a$ variables and $b$ constraints, although the actual running time depends on the number of local minima, which is likely to be far smaller than $(mn)^{2k}$. An alternative heuristically-based subdivision algorithm (the distance-based subdivision method of Section 4.5.4) gives containment an even better running time in practice, and is guaranteed to not create new local overlap minima.

Our practical distance-based subdivision technique is our third improvement to the indirect algorithm. This method attempts to maximize the distance from the current evaluation to the convex approximations of the subdivided configuration spaces. This technique is described in Section 4.5.4. It moves the evaluator away from the current configuration so that evaluation does not get “stuck in a rut”. It does this by eliminating some of the overlapping configurations which are “close to” the current overlapping one.

We call our improved algorithm a hybrid algorithm because it uses indirect methods for evaluation, developed by Li and Milenkovic, and both direct and indirect methods for restriction. This algorithm is also described in [MD95].

The hybrid algorithm, in addition to being a formidable containment algorithm, has the property that a partial containment solution from it is valuable. This is because each evaluation step produces either a non-overlapping solution or an overlapping arrangement for which a linear program has reached a local minimum. As a result, the hybrid algorithm, when terminated early, is the most powerful heuristic known for generating a local overlap minimum.

In our experiments, the new algorithm clearly outperforms purely geometric containment algorithms, such as the approximate algorithm of Section 5.2. However, it does this by combining rather than replacing geometric techniques with linear programming. This “hybrid” algorithm demonstrates the manner in which linear programming can greatly increase the power of geometric algorithms.

**High-Level Algorithm Description**

The algorithm starts with a queue $Q$ of hypotheses containing a single hypothesis: the set of original $U_{ij}$, $0 \leq i < j \leq k$, as calculated using Equation 2.1. We assume that this hypothesis has been restricted using some geometric restrictions.
HYBRID-CONTAIN\((Q, k)\)
\[\text{while } (Q \text{ is not empty}) \]
\[\mathcal{U} \leftarrow \text{DEQUEUE}(Q)\]
\[\mathcal{U} \leftarrow \text{SSLPR}(\mathcal{U}, k)\]
\[\text{if (some } U_{ij} \in \mathcal{U} = \emptyset) \text{ continue while loop}\]
\[\text{status} \leftarrow \text{OVERLAP-EVALUATE}(\mathcal{U}, k)\]
\[\text{if (status = YES) return YES}\]
\[\text{if (status = NO) continue while loop}\]
\[\mathcal{U}', \mathcal{U}'' \leftarrow \text{DISTANCE-BASED-SUBDIVIDE}(\mathcal{U}, k)\]
\[\text{ENQUEUE}(\mathcal{U}', Q)\]
\[\text{ENQUEUE}(\mathcal{U}'', Q)\]
\[\text{return NO}\]

5.3.2 Results

The examples we selected are primarily from the apparel industry. We use SSLPR only at the deepest levels of hypothesis splitting, for two reasons. The first is cost. Although SSLPR is several times faster than SSGR, it is not necessarily cost-effective to invoke it prior to evaluating each hypothesis. The second is that SSLPR restricts polygons more as their convex hulls become better approximations to them.

Comparison with Other Algorithms

We compared the performance of our hybrid algorithm with two algorithms: the approximate algorithm of Section 5.2 and the naive exact kNN algorithm of [DM95, Mil], which is based on the CLP. Table 5.6 gives running times for feasible containment problems. Running times are expressed in seconds on a 50 MHz SPARCstation. Table 5.7 gives running times for infeasible containment problems. Our new algorithm clearly outperforms both of the other algorithms. For five or more polygons, the difference is dramatic. Figure 5.6 and Figure 5.7 depict configurations obtained by our algorithm.

The approximate algorithm uses SSGR, whose current implementation runs about ten times more slowly than SSLPR. A more fair comparison between the approximate and hybrid algorithms counts the number of hypotheses examined instead of simply the running time. Table 5.8 shows the results of such an experiment. The approximate algorithm examines a total of 76 hypotheses, whereas the hybrid algorithm examines 100. From this perspective, the approximate algorithm appears more powerful. For the 7CC example, both algorithms find a solution without any subdivision; each evaluates only one hypothesis. However, the approximate algorithm requires 823 seconds, whereas the hybrid algorithm finds the solution in only 7 seconds. The time difference here is due to the difference in the speed of SSGR and SSLPR, and to the fact that the greedy evaluator invokes SSGR each time a \(U_{ij}\) is shrunk to a point.

SSGR vs. SSLPR

Here we discuss the effectiveness of SSGR and SSLPR within our new algorithm.

We tested our algorithm on 21 different feasible and infeasible containment examples for \(3 \leq k \leq 8\). For each example, we varied the type of restriction used: 1) no restriction, 2)
<table>
<thead>
<tr>
<th>Type</th>
<th>APPROX</th>
<th>NAIVE LP</th>
<th>HYBRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>3NN</td>
<td>19</td>
<td>22</td>
<td>9</td>
</tr>
<tr>
<td>3NN</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4CC</td>
<td>25</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4CC</td>
<td>21</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>4CN</td>
<td>113</td>
<td>84</td>
<td>4</td>
</tr>
<tr>
<td>4NN</td>
<td>155</td>
<td>19</td>
<td>33</td>
</tr>
<tr>
<td>4NN</td>
<td>80</td>
<td>24</td>
<td>35</td>
</tr>
<tr>
<td>4NN</td>
<td>35</td>
<td>43</td>
<td>15</td>
</tr>
<tr>
<td>4NN</td>
<td>319</td>
<td>89</td>
<td>10</td>
</tr>
<tr>
<td>5CC</td>
<td>126</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>5NN</td>
<td>&gt;5min.</td>
<td>&gt;5min.</td>
<td>24</td>
</tr>
<tr>
<td>5NN</td>
<td>428</td>
<td>-</td>
<td>16</td>
</tr>
<tr>
<td>5NN</td>
<td>189</td>
<td>-</td>
<td>36</td>
</tr>
<tr>
<td>5NN</td>
<td>317</td>
<td>-</td>
<td>44</td>
</tr>
<tr>
<td>6CC</td>
<td>75</td>
<td>53</td>
<td>5</td>
</tr>
<tr>
<td>6CN</td>
<td>&gt;5min.</td>
<td>&gt;5min.</td>
<td>184</td>
</tr>
<tr>
<td>6CN</td>
<td>2540</td>
<td>-</td>
<td>372</td>
</tr>
<tr>
<td>6CN</td>
<td>2238</td>
<td>-</td>
<td>1031</td>
</tr>
<tr>
<td>6NN</td>
<td>&gt;5min.</td>
<td>&gt;5min.</td>
<td>86</td>
</tr>
<tr>
<td>6NN</td>
<td>&gt;15min.</td>
<td>&gt;15min.</td>
<td>593</td>
</tr>
<tr>
<td>7CC</td>
<td>823</td>
<td>162</td>
<td>7</td>
</tr>
<tr>
<td>7CN</td>
<td>&gt;5min.</td>
<td>&gt;5min.</td>
<td>72</td>
</tr>
<tr>
<td>7NN</td>
<td>&gt;10min.</td>
<td>&gt;10min.</td>
<td>176</td>
</tr>
<tr>
<td>8CC</td>
<td>&gt;5min.</td>
<td>&gt;5min.</td>
<td>34</td>
</tr>
<tr>
<td>10CC</td>
<td>-</td>
<td>-</td>
<td>14</td>
</tr>
<tr>
<td>10NN</td>
<td>-</td>
<td>-</td>
<td>183</td>
</tr>
</tbody>
</table>

Table 5.6: Running times for feasible examples in seconds
SSGR, 3) SSLPR, and 4) SSGR followed by SSLPR. In all cases, we began the containment algorithm by applying Equation 3.1, which bounds the unbounded $U_{ij}$s.

Let $N$ be the number of hypothesis evaluations required if we use no restrictions at the individual hypothesis level. Let $L$ be the number of evaluations if we use SSLPR for each hypothesis. Let $G$ be the number of evaluations if we use SSGR for each hypothesis. Finally, let $GL$ be the number of evaluations if we follow SSGR by SSLPR for each hypothesis. Table 5.9 shows values of $N$, $L$, and $G$ for some of our examples.

The table omits $GL$ because it is almost always true that $G = GL$. However, in one case, SSLPR eliminated something SSGR could not eliminate. This occurred after many levels of subdivision.

In all 21 cases, $\min(G, L, GL) \leq N$. This is true of both feasible and infeasible cases. The difference is often dramatic (e.g., for one 4NN infeasible example, $N = 237$, $L = 21$, and $G = 7$). It is always the case that both $G \leq N$ and $L \leq N$. It is often true that $|L - G|$ is small. In 17 of 21 cases, $G < L$. The total of $G$ for the 21 examples is only 46, whereas the total for $L$ is 117.

Sometimes $G$ is slightly larger than $L$. We believe that this phenomenon is due to a subdivision method we experimented with for these tests. This method isolates an individual component of a $U_{ij}$. If restriction produces many components, just isolating one at a time can create many hypotheses. In this case, a cutting algorithm which moves the configuration the maximum possible distance from the current configuration might be preferable. It is possible that the best subdivision strategy might depend on which type of restriction is used at the hypothesis level. For example, since SSLPR is most effective when a polygon is similar to its convex hull, subdivision in this case should maximize the convexity of $U_{ij}$ and $U_{ij}''$. For SSGR, perhaps subdivision should minimize the amount of “closure” of the two subpolygons. We are still evaluating subdivision strategies for our hierarchical containment algorithm.

Our results suggest that both SSGR and SSLPR can dramatically reduce the number of hypotheses which must be evaluated. Using SSLPR causes the algorithm to evaluate more hypotheses than SSGR in our experiments. However, for our current implementations of SSGR and SSLPR, SSLPR is the most cost-effective restriction in this context because SSLPR is approximately ten times faster than SSGR.

**Subdivision**

In order to gauge the success of our subdivision strategy, we compared the number of hypotheses evaluated by the naive exact algorithm versus the hybrid algorithm on infeasible examples. For this comparison, we disabled the high-level geometric restriction as well as the hypothesis-level restrictions. The subdivision method we tested uses distance-based subdivision when the $U_{ij}$ consists of a single component. If the $U_{ij}$ to be subdivided has multiple components, we isolate the component which is nearest to the “bad point”. The results are shown in Table 5.10. In every case the hybrid algorithm requires fewer evaluations than the naive exact algorithm. This suggests that a subdivision strategy which uses the information from a local overlap minimum can be effective.
Figure 5.6: 4NN example: time = 15 seconds

Figure 5.7: 7NN example: time = 176 seconds
5.3.3 Conclusion

We now have both geometric and linear programming-based methods for restriction, evaluation, and subdivision. Each appears to have strengths in different areas, and they can be combined to achieve good results. In particular, geometric restrictions are very helpful at the "top level" when the $U_{ij}$ are very nonconvex, and we definitely apply them there. This work in no way shows the superiority of either approach, but instead shows how well they can work together.

5.4 Summary and Conclusions

This chapter has presented two $k$NN algorithms based on our restrict/subdivide/evaluate paradigm. The first is a numerically approximate algorithm because it produces an $\varepsilon$-approximate configuration. It uses a partial containment characterization to produce a set of $k$ subproblems. It uses SSGR, a greedy evaluator, and size-based subdivision to solve each subproblem.

The second algorithm is a hybrid algorithm. It uses both geometric and linear programming restrictions, an overlap reduction evaluator, and distance-based subdivision. SSLPR is used at the hypothesis level. Our experiments suggest that distance-based subdivision helps the hybrid algorithm detect infeasibility. This algorithm is superior to the original CLP-based algorithm of Milenkovic.

We give running times for both of these algorithms for a variety of $k$CC, $k$CN, and $k$NN containment examples, for $3 \leq k \leq 10$. The hybrid algorithm is currently faster than the approximate algorithm for $k > 4$. The hybrid algorithm typically solves our containment examples in several minutes or less on a 50 MHz SPARCstation, for $3 \leq k \leq 10$, $4 \leq m \leq 100$, and $100 \leq n \leq 300$. We believe that the evaluation methods of the two algorithms are of roughly the same effectiveness, and that this difference in speed is primarily due to the fact that the current implementation of SSLPR is roughly ten times faster than the current implementation of SSGR. This conclusion is supported by the fact that the number of hypotheses evaluated by the approximate algorithm in our experiment is less than the number of hypotheses evaluated by the hybrid algorithm. Speeding up our implementation of SSGR is a subject of future work (see Section 10.2.1). We believe that, given fast enough polygon set operations, the approximate algorithm has the potential to be competitive with the hybrid algorithm.
<table>
<thead>
<tr>
<th>Type</th>
<th>APPROX</th>
<th>NAIVE LP</th>
<th>HYBRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4NN</td>
<td>47</td>
<td>152</td>
<td>135</td>
</tr>
<tr>
<td>4NN</td>
<td>138</td>
<td>3037</td>
<td>493</td>
</tr>
<tr>
<td>4NN</td>
<td>&gt; 10 min.</td>
<td>&gt; 10 min.</td>
<td>416</td>
</tr>
<tr>
<td>5NN</td>
<td>&gt; 10 min.</td>
<td>&gt; 10 min.</td>
<td>103</td>
</tr>
</tbody>
</table>

Table 5.7: Running times for infeasible examples in seconds

<table>
<thead>
<tr>
<th>Type</th>
<th>APPROX</th>
<th>HYBRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4CC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4CN</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>4CN*</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>4CN*</td>
<td>23</td>
<td>13</td>
</tr>
<tr>
<td>4NN</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4NN*</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>4NN*</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>4NN*</td>
<td>23</td>
<td>15</td>
</tr>
<tr>
<td>5CN</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5NN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5NN</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5NN</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6CN</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>6CN</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>7CC</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.8: Number of hypotheses examined ((*) denotes infeasible example)
<table>
<thead>
<tr>
<th>Type</th>
<th>N</th>
<th>L</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>3NN</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3CN*</td>
<td>61</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4CN</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4CN</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4CN</td>
<td>21</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4CN*</td>
<td>25</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4CN*</td>
<td>15</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>4CN*</td>
<td>205</td>
<td>27</td>
<td>3</td>
</tr>
<tr>
<td>4CN*</td>
<td>85</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>13</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4NN</td>
<td>18</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4NN</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>26</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4NN</td>
<td>257</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4NN*</td>
<td>237</td>
<td>19</td>
<td>7</td>
</tr>
<tr>
<td>4NN*</td>
<td>81</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>5CN</td>
<td>10</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5CN</td>
<td>73</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6CN</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8CC</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.9: Number of hypotheses evaluated for restriction options (* denotes infeasible example)

<table>
<thead>
<tr>
<th>Type</th>
<th>NAIVE LP</th>
<th>HYBRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4CN</td>
<td>32</td>
<td>27</td>
</tr>
<tr>
<td>4NN</td>
<td>493</td>
<td>153</td>
</tr>
<tr>
<td>4NN</td>
<td>153</td>
<td>103</td>
</tr>
<tr>
<td>4NN</td>
<td>42</td>
<td>11</td>
</tr>
<tr>
<td>4NN</td>
<td>1736</td>
<td>311</td>
</tr>
<tr>
<td>4NN</td>
<td>436</td>
<td>323</td>
</tr>
</tbody>
</table>

Table 5.10: Number of hypotheses evaluated for infeasible problems (no hypothesis restriction)
Part II

Applications of Containment to Layout
6.1 Introduction

This part of the thesis gives examples of layout problems which can be solved using two-dimensional translational containment for irregular polygons. The problems are of two types: 1) minimal container, and 2) maximal item. These applications support our claim that being able to quickly solve containment problems for up to ten items allows a variety of layout problems to be solved more effectively.

Recall that our containment approach is a "bottom-up" one in which we maximize the number of items for which algorithms are practical. Clustering techniques for layout also use a bottom-up approach. Some of them even allow the items to rotate. However, the methods which cluster more than two arbitrary nonconvex polygons are greedy heuristics. Our containment-based solutions to minimal container problems do not allow the items to rotate, but, in contrast to the heuristics, we establish an actual lower bound on the size of the container for multiple items.

In addition, we believe that some potentially promising layout techniques have been abandoned by the layout community because they lack strong bottom-up methods. The primary example of this is pre-packing. Recall that a pre-packing strategy generates feasible groups for each container and then assigns groups to containers. If the layout world had a practical method for generating groups of items which fit into each container, this, plus an appropriate assignment strategy, would make pre-packing viable. We demonstrate, through our containment-based solutions to maximal item problems, that pre-packing can work.

We also solve a problem which arises in multi-stage layout problems, and which must be solved before a strategy such as pre-packing can be applied. The problem is to transform a single, highly nonconvex container into a collection of containers.

We remark that, although the primary use of containment is for automatic solutions to layout problems, containment can also be part of a collection of computer-aided layout tools for humans as well as assist in training humans who solve layout tasks manually.
6.1.1 Overview of Chapter

Section 6.2 discusses our work on the container transformation problem. Section 6.3 gives an overview of our work on minimal container problems. Section 6.4 discusses our work on maximal item problems. Section 6.5 explains the structure of this part of the thesis.

6.2 Overview of Container Transformation Work

The motivation for our container transformation work is the trim placement problem in apparel manufacturing (see Section 1.4). Trim placement is the second stage of a two-stage layout process for pants markers. The items to be placed are either large pattern pieces or small trim pieces. The first stage places the large pattern pieces. The “container” for the second stage is the complement of the large placed pieces within the first stage container. It is typically highly nonconvex. Rather than operate on this unwieldy container directly, we transform it into a collection of containers. The trim placement phase places as many of the small trim pieces as possible into the containers.

An ideal transformation of the container both limits and decomposes it. The limiting process removes unnecessary regions which are not reachable by any of the items to be placed. We use restrictions (see Chapter 3) to remove regions which are clearly unreachable. The decomposition process produces a collection of containers which may overlap each other. Overlap among containers makes it possible for placed items to overlap each other. We show how to balance the conflicting goals of maximizing the number of containers while minimizing the overlap between them. Maximizing the number of containers produces manageable containment problems. We allow the user to select the probability that the containers will be overlapping.

For efficiency purposes, we introduce the notion of a breaking polygon. This is a polygon which can be translated to fit inside any of the items to be placed. In order to find a breaking polygon, we find the maximum area axis-parallel rectangle (MAAPR) in a polygon. Our theoretical work on the MAAPR problem uses fast matrix searching techniques from the computer science literature and gives the fastest known asymptotic running time for this problem: $O(n \log^2 n)$ time for an $n$-vertex polygon.

Our work on container decomposition is applicable, in general, to multi-stage layout problems because the second stage of a multi-stage layout problem may involve a large, highly nonconvex container which needs to be decomposed. Highly nonconvex containers also appear in industries in which defects in material cause the material to be partitioned into “quality zones” (see Heistermann and Lengauer’s work with leather [HL92]).

6.3 Overview of Minimal Container Work

From the discussion of related work in layout in Section 1.3.3, we see that solutions to minimal enclosure problems are often required for the first stage of a multi-stage layout problem. Minimal container problems also arise when finding the best stock sheet size for a given set of items, or the best cloth width in garment manufacturing. Current solutions to minimal enclosure problems for more than two arbitrary nonconvex items often allow the items to rotate, but they use greedy heuristics. The heuristics cannot produce a lower bound on the size of the enclosure.
We give four examples of minimal container problems: 1) strip packing, 2) finding minimal square enclosures, 3) finding minimal rectangular enclosures, and 4) finding tight convex and nonconvex enclosures. Before discussing our work on these problems in Section 6.3.2, we summarize related work on enclosure problems in Section 6.3.1.

6.3.1 Related Work

Attempts to treat multiple polygon enclosure problems in the literature typically use greedy heuristics. Adamowicz and Albano [AA76], Haims and Freeman [HF70], Amaral, et al. [ABJ90], Dagli and Tatoğlu [DT87], and Bailleul et al. [BTS83] pack nonconvex items into rectangles to form composite items, and then place the rectangles. In [AA76], Adamowicz and Albano give an algorithm for pairwise clustering of irregular items in a rectangle, and describe limited clustering options for more than two items. They allow polygons to rotate. They perform pairwise clustering, using an object called the No Fit Polygon (NFP)\(^1\). They also perform limited clustering for more than two items, by first clustering two large items and then placing small items within the bounding box of that cluster. Amaral, et al. [ABJ90] adopt the approach of [AA76] for clustering items.

Dagli and Tatoğlu [DT87] cluster pairs of items in a greedy fashion and allow the items to rotate. They first cluster two items, make them a unit, cluster this unit with another item, and so on. Prasad and Somasundaram [PS91] and Babaev [Bab82] form an efficient cluster of items, then build a repeating pattern in order to nest sheet-metal blanks. Ismail and Hon [IH92] use a genetic algorithm to cluster two similar items together; the items are allowed to rotate. Dighe and Jakiela [DJ95] use a hierarchical genetic algorithm to cluster items. The items are allowed to rotate. Their algorithm uses a binary tree in which each item (together with values for its three parameters: \(x\), \(y\), and \(\theta\)) constitutes a leaf, and a node represents a search for the minimal enclosing rectangle of its two children. Thus, the genetic algorithm seeks the best binary tree for a set of items. Milenković, et al. [MDL92] solve the specialized problem of building 4-item stacks of pants panels for layouts in apparel manufacturing.

In computational geometry, algorithmic work appears on various types of enclosure problems for single items. Finding a minimal enclosure for a single polygon can also be viewed as a polygon approximation problem, as noted in [CY86].

Chang and Yap define inclusion and enclosure problems [CY86]. They define an enclosure problem as follows: \(\text{Enc}(P, Q, \mu)\): Given \(P \in P\), find the \(\mu\)-smallest \(Q \in Q\) that encloses \(P\), where \(P\) and \(Q\) are families of polygons, and \(\mu\) is a real function on polygons such that:

\[
\forall Q, Q' \in Q, \quad Q' \subseteq Q \quad \Rightarrow \quad \mu(Q') \leq \mu(Q)
\]

Chang and Yap [CY86] review work on a variety of such enclosure problems. We briefly mention two examples of enclosure work. Melissaratos and Souvaine [MS90a] find the minimum concave quadrilateral circumscribed about a simple polygon, and O'Rourke [O'R85] finds the minimal box circumscribing a set of three-dimensional points\(^2\).

---

\(^1\)The NFP of polygon \(A\) with respect to polygon \(B\) is really just \(B \oplus -A\) (see Section 2.2).

\(^2\)Chang and Yap’s definition of \(P\) could be extended to point sets for this case.
6.3.2 Our Minimal Container Work

Our containment-based solutions to minimal container problems do not allow the items to rotate, but, in contrast to heuristics, we establish an actual lower bound on the size of the container for multiple items. Our general approach to finding minimal enclosures is to find an optimal value for a parameter via binary search. We remark that the binary search can be accelerated by using compaction techniques, such as those developed by Li and Milenkovic [Li94, ML, LM93c, LM93a], to improve a feasible solution and thereby help update the upper bound. Our approach to these problems follows the common operations research approach of tightening upper and lower bounds to obtain an optimal value. Containment provides the lower bounds in our case. Our hybrid containment algorithm is appropriate for minimal enclosure problems. It is extremely fast for loose fits, and so the initial stages of the search are fast.

For the problem of finding tight convex and nonconvex enclosures, we show that removing unreachable portions of a tight rectangular enclosure can yield tight convex and nonconvex enclosures which preserve all the containment solutions for the rectangle.

6.4 Overview of Maximal Item Work

We give four examples of maximal item problems. For this discussion, let \( k_{\text{practical}} \) be the largest number of items for which current containment algorithms are feasible. Our problems are: 1) given a container and a collection of items, finding maximally scaled copies of items which fit in the container, 2) given a container and a collection of \( \leq k_{\text{practical}} \) items for which the sum of the item areas does not exceed the area of the container, finding a “large” sub-collection which fits in the container, 3) given a container and a collection of items for which either the sum of the item areas exceeds the area of the container or the number of items is \( > k_{\text{practical}} \), finding a “large” sub-collection of \( \leq k_{\text{practical}} \) items which fits in the container, and 4) problem (3) for a collection of containers.

Section 1.3 contains an extensive survey of the relevant literature on layout and packing problems. It also reviews the limited layout literature on the assignment problem, which arises as part of problem (4).

We discuss our work on maximal item problems below in Section 6.4.1.

6.4.1 Our Maximal Item Work

For problem (1) we use binary search combined with containment. In problem (2) the given items cannot be scaled; they can only be included in or excluded from the set. The key to solving this problem is to carefully choose which item to eliminate from an infeasible subset. We reject the most “troublesome” item, which is identified using the results of the overlap reduction evaluation method of the hybrid containment algorithm. For problem (3), we randomly construct a set of items and solve problem (2). This yields a rich collection of sets which fit into the container. For each set of size \( n \) which fits, we automatically have \( 2^n \) subsets which fit and which need not be individually represented. This work provides the first practical method for constructing sets of up to ten polygons which fit in a container.

For problem (4), we solve problem (3) for each container, and then apply a look-ahead assignment strategy. This is a pre-packing strategy. Containment thus allows us to transform a geometric optimization problem into a combinatorial optimization problem. Our
results challenge the existing belief that pre-packing methods are impractical for layout [Arb93].

6.5 Overview of Part Two

Chapter 7 discusses our work on container decomposition. Our container decomposition work also appears in [DM94]. Our theoretical work on the MAAPR problem is summarized in Chapter 7, and also appears in [DMR93, DMR]. Chapter 8 solves four minimal container problems. Chapter 9 solves four maximal item problems, including pre-packing.
Chapter 7

Limited Gaps

7.1 Introduction

In this chapter we consider the following problem: given a complex container and a set of items to be placed in the container, transform the container by limiting and decomposing it. The limiting process removes unnecessary regions which are not reachable by any of the items to be placed. The decomposition process produces a collection of containers. Maximizing the number of containers produces manageable containment problems.

The motivation for our container transformation work is the trim placement problem in apparel manufacturing (see Section 1.4). Trim placement is the second stage of a two-stage layout process for pants markers. The items to be placed are either large pattern pieces or small trim pieces. The first stage places the large pattern pieces. The “container” for the second stage is the complement of the large placed pieces within the first stage container. This container is highly nonconvex and may have many components. An example is shown in Figure 7.1. In that figure, the darkly shaded regions together form the second stage container. They represent gaps formed by neighboring pants panels. The items to be placed during the second stage appear stacked at the bottom of the figure. These items are trim pieces. Rather than operate on this unwieldy container directly, we transform it into a collection of containers. The trim placement phase places as many of the small trim pieces as possible into the containers.

Our work on container decomposition is applicable to multi-stage layout problems, in general, because the second stage may involve a large, highly nonconvex container. Highly nonconvex containers also appear in industries in which defects in material cause the material to be partitioned into “quality zones” (see Heistermann and Lengauer’s work with leather [HL92]).

An ideal transformation of the container both limits and decomposes it. We show how the container can be limited with the help of restrictions (see Chapter 3). The limiting process removes regions which are clearly not reachable by any item. (Identifying all unnecessary regions is an NP-hard problem.) The need for eliminating unreachable regions has been suggested by Amaral, et al. [ABJ90]: “It is ... necessary to implement algorithms that identify those gaps which cannot contain any shape, due to its dimensions or geometry.”
Karasic [Kar89] conjectures that humans perform this function intuitively when they place items in a container.

The decomposition process produces a collection of containers which may overlap each other. Overlap among containers makes it possible for placed items to overlap each other. We show how to balance the conflicting goals of maximizing the number of containers while minimizing the overlap between them. As mentioned earlier, maximizing the number of containers produces manageable containment problems. Our method allows the user to select the probability that the containers will be overlapping.

For efficiency purposes, we introduce the notion of a breaking polygon. This is a polygon which can be translated to fit inside any of the items to be placed. In order to find a breaking polygon, we find the maximum area axis-parallel rectangle (MAAPR) in a polygon. Our theoretical work on the MAAPR problem uses fast matrix searching techniques from the computer science literature and gives the fastest known asymptotic running time for this problem: $O(n \log^2 n)$ time for an $n$-vertex polygon.

This chapter describes our efforts to provide a sound mathematical basis and a set of algorithms for container limiting and decomposition. We call each component of the limited and decomposed container a limited gap.$^1$

7.1.1 Decomposing the Container into Gaps

Given a collection of $k$ polygonal items $\mathcal{P}$ and a container $C$, we seek a gap set $G = \{g_1, g_2, \ldots, g_l\}$, such that $\bigcup g_i = C$, with the containment property; if $p \in \mathcal{P}$ is placed inside the container, then it is contained entirely inside some $g \in G$. Smaller gaps and more gaps are desirable; overlap among gaps is undesirable.

The set of connected components of the container has the containment property, but it usually has too few elements to improve tractability much in practice. We will define a subset $R_\mathcal{P}$ of the container called the reachable region of $\mathcal{P}$. The connected components of $R_\mathcal{P}$ form a much better gap set than the container itself, but $R_\mathcal{P}$ still is often insufficiently decomposed. However, it is the largest cardinality gap set we know of with non-overlapping gaps.

If we permit gaps to overlap, then it is possible for polygons placed in different gaps to overlap. However, if we construct the gaps properly, then the probability of overlap

$^1$After creating the term limited gap, we noticed the double connection of the name to the retail side of the apparel business.
will be small. We will give a rigorous notion of this probability. Furthermore, for any given probability $\pi$, we will show how to construct a gap set, called limited gaps, such that the probability that items in different gaps overlap is less than $\pi$, assuming a uniform distribution.

There is a tradeoff between the number and size of gaps and the probability. Many small gaps imply small $k$ for a containment problem for an individual gap, but the probability of overlap between items in different gaps is higher. Since $\pi$ is a parameter to the construction of the limited gaps, we allow the possibility of finding the best value of $\pi$ in the tradeoff.

7.1.2 Overview

Section 7.2 describes the container decomposition methods we experimented with before arriving at the notion of the limited gaps. Section 7.3 defines the limited gaps in terms of the probability $\pi$ and gives an algorithm for constructing them. Section 7.4 presents two methods for improving the efficiency of limited gap construction. These methods introduce the new concepts of breaking polygons and inking polygons. Section 7.5 summarizes our work on finding the maximum area axis-parallel rectangle in a polygon. This is used to construct a breaking polygon. Section 7.6 summarizes this chapter and states conclusions.

7.2 The Challenge of Constructing Gap Sets

Before defining the limited gaps, we review the other container decomposition methods we experimented with. Our methods assume that the container is the product of the first stage of a two-stage layout problem. The first stage places a collection of items which become obstacles in the second stage. That is, the container for the second stage is the region within the original container which is the complement of the obstacles. We also assume that a transformation of the obstacles which improves the first-stage layout is legal.

7.2.1 Compaction

Li and Milenkovic have developed a compaction tool [MDL92, LM93d, Li94, ML, LM93c, LM93a] which plans a motion for a set of placed items under a set of applied "forces." We attempted to eliminate holes and narrow "necks" in the container by applying a leftward force to each obstacle polygon. In theory, this process is guaranteed to eliminate holes because each obstacle will be in contact with at least one other obstacle. However, we found that this strategy has three shortcomings: 1) it fails to close all the narrow necks, 2) although the container components become simply connected, unnaturally large regions still exist, and 3) contact detection suffers from numerical instability. Through the use of a configuration space approach, the compaction algorithm avoids the question of contact detection in order to run quickly and numerically robustly. However, in order to generate a good decomposition, it is necessary to explicitly detect contacts, and this proves to be numerically unstable.

7.2.2 Partitioning Squares

Our second attempt involved choosing a narrowness tolerance $\gamma$. We formed a partitioning square with side of length $\gamma$ and used this to partition the container. The partitioning
process is conceptually analogous to centering a partitioning square at each vertex of each obstacle, and then removing those squares for which the intersection with the set of obstacles is connected (see Figure 7.2; this illustrates a square which is not removed). The difficulties with this approach are: 1) it introduces artifacts (pieces of squares) into the layout, 2) it fragments the container in a way that leaves many small regions which are not usable by the set of items to be placed, and 3) choosing a universal $\gamma$ is inherently impossible. The fragmentation issue can be resolved by testing each region, but this is computationally expensive\(^2\). Choosing $\gamma$ is not possible because $\mathcal{P}$ does not naturally give rise to a narrowness value. Adopting a value corresponding to regions which no item can fit through is insufficient, because an item might be able to reach into an arbitrarily narrow neck from both sides, thus making the neck usable (see Figure 7.3). The only “natural” solution is to let the left and right regions overlap.

Figure 7.3: A neck which is usable although items cannot pass through it

7.2.3 Other Methods

We considered other methods, such as computing a Voronoi diagram of the obstacles, using edges of the obstacles as sites [For87]. However, such methods also rely on a narrowness tolerance $\gamma$, which is a fundamentally flawed approach.

7.3 Limited Gaps

We first define the set of limited gaps for $\mathcal{P}$ when $P_i = P_j$ for $1 \leq i < j \leq k$. Then we give a straightforward extension of the theory to noncongruent shapes. For ease of notation, we denote $P_j$ by $p$. We also denote $U_{0j}^*$ by $V_p$ (recall from Section 3.1.2 that $U_{0j}^*$ is the replacement of $U_{0j}$ in a valid restriction of $\mathcal{U}$).

7.3.1 Reachable Regions

**Definition 7.1** The set $R_p = V_p \oplus p$ is the reachable region of $p$ with respect to $\mathcal{P}$.

\(^2\)One could use a breaking polygon (see Section 7.4.1) to reduce the expense.
Note that if we use $U_{0}^{\text{orig}}$, the unrestricted $U_{0,j}$, in place of $V_{p}$, the reachable region becomes what is known in mathematical morphology and image analysis as the opening of a point set [Ser82, Ser88]. At the other extreme, if we use $U_{0,j}$ in place of $V_{p}$, we obtain $V_{p}^{*}$. The set $V_{p}^{*} \oplus p$ is a tight upper bound on the reachable region under set inclusion.

A gap set $G_{R}$ consisting of the connected components of $R_{p}$ has the containment property, and the components have the added advantage of being non-overlapping. Unfortunately, components of $R_{p}$ can contain arbitrarily narrow necks. Ideally, each component of $R_{p}$ is generated by a single component of $V_{p}$. Unfortunately, dilation by $p$ can blend together two components of $V_{p}$ for cases in which $p$ cannot fit through a narrow neck of the container but two copies of $p$ can touch if they reach into the neck from different sides, as in Figure 7.3.

By dilating each individual component of $V_{p}$, we can create a gap set $G_{V}$ with higher cardinality than $G_{R}$ and with no narrow necks. Unfortunately, these gaps can be highly overlapping. The limited gap set will be somewhere “in between” $G_{V}$ and $G_{R}$.

One advantage of $R_{p}$ is that it provides a nontrivial upper bound on the efficiency\(^{3}\) of a container, as follows. The efficiency of packing copies of $p$ in a container $C$ cannot exceed the ratio of $\text{area}(R_{p})$ to $\text{area}(C)$. Most packing density bounds in the literature apply only when the polygons are convex, but this one works for any type of polygon.

Karasic [Kar89] conjectures that human layout experts employ the opening as a heuristic criterion. He notes that the opening can be used not only to evaluate the quality of a polygon’s placement, but also for selecting the next polygon to place. The area of $R_{p}$ can also be used within a greedy layout algorithm as a criterion for judging the placement of polygons. Larger $R_{p}$ is better in this case, and the algorithm chooses a placement position for each item which results in the most usable space for the remaining unplaced items.

### 7.3.2 Defining Limited Gaps using Probability of Interference

Let $v, v' \subseteq V_{p}$ be two connected components. If $v \oplus p$ and $v' \oplus p$ do not intersect, then we can independently place copies of $p$ into the container using translations from $v$ and $v'$ without interference. If $(v \oplus p) \cap (v' \oplus p) \neq \emptyset$, then we want to compute a probability $\text{Prob}(v, v')$ that $(p + t) \cap (p + t') \neq \emptyset$. We assume a uniform distribution on the connected components of $V_{p}$, so $t \in v$ and $t' \in v'$ are chosen uniformly. Assuming we can compute $\text{Prob}(v, v')$, we can finally define limited gaps. Given a probability $\pi$, components $v$ and $v'$ interfere with each other if $\text{Prob}(v, v') \geq \pi$. Partition the components of $V_{p}$ according to the transitive closure of the interference relation into interference sets. Finally, dilate each interference set by $p$ to generate a gap. The resulting set $G_{L}(\pi)$ of gaps is the limited gap set for probability $\pi$. There are variations on this definition, but the point is that one can select the probability $\pi$ of interference for the gap set.

We calculate $\text{Prob}(v, v')$ as follows. For each $t \in v$, the set of $t' \in v'$ such that $(p + t) \cap (p + t') \neq \emptyset$ is $((p + t) \oplus (p)) \cap v'$, and the probability that this choice of $t$ results in interference is $\text{Prob}([t], v') = \text{area}((p + t) \oplus (p)) \cap v')/\text{area}(v')$. The integral of $\text{Prob}([t], v')$ over all $t \in v$ gives us $\text{Prob}(v, v')$.

We do not have an exact algorithm for this integral, but we can approximate it by sampling $t \in v$. We can improve the efficiency by sampling only the region $v \cap ((v' \oplus p) \oplus -p)$ that can result in a non-zero value of $\text{Prob}([t], v')$. After computing the average value of

\(^{3}\)Efficiency is the ratio of used area to total area.
this probability, we normalize by multiplying it by area\((v \cap ((v' \oplus p) \oplus -p)) / \text{area}(v)\).

Finally, if \(P\) contains noncongruent shapes, then we can similarly define and compute \(\text{Prob}(v, v')\) for \(v \subseteq V_p\) and \(v' \subseteq V'_{p'}\). Let \(V_{P'}\) be the set of all components of the \(V_p\) regions. We partition \(V_{P'}\) according to interference. In this case, \(v\) and \(v'\) can only interfere if they come from different \(V_p\) and \(V'_{p'}\). For each partition set \(\Pi \subseteq V_{P}\), we dilate each component \(v \in \Pi\) by the appropriate \(p \in P\), and then take the union \(\bigcup_{v \in \Pi} (v \oplus p)\) to generate the gap corresponding to \(\Pi\).

### 7.4 Efficient Computation of Limited Gaps

In order to efficiently construct the limited gaps, we want to quickly partition \(V_{P}\) into interference sets. We present two helpful partitioning techniques. The first technique involves finding a breaking polygon: a polygon \(B\) that can be translated to fit inside any \(p \in P\).\(^4\)

The second technique uses the notion of inking polygons: \(q\) inks \(p\) if the reachable region of \(q\) inside \(p\) is all of \(p\). Section 7.4.1 discusses the breaking polygon technique. Section 7.4.2 introduces the more powerful concept of inking polygons.

#### 7.4.1 Breaking Polygons

Let us denote a breaking polygon of \(P\) by \(B\). To simplify the presentation, we assume that each \(p \in P\) has its origin positioned so that \(B \subseteq p\) without translation. Since \(B \subseteq p\), it follows that every polygon in \(V_{P}\) is a subset of some polygon in \(V_B\).

We can use \(B\) to quickly group together components of \(V_{P}\) which might intersect each other. Clearly, two components of \(V_{P}\) cannot intersect each other unless they are in the same component of \(V_B\). Furthermore, since each connected component of \(V_{P}\) is entirely within a connected component of \(V_B\), the inclusion test consists only of a single point-in-polygon test.

Since we want to maximize the number of components of \(V_B\), we want to maximize the area of \(B\). Finding the polygon of maximum area which can be translated to lie inside of each \(p \in P\) is an interesting open problem which we do not solve here. In practice, we find the maximum area axis-parallel rectangle (MAAPR) of each \(p \in P\). Finding the MAAPR quickly is a challenging problem in its own right and is the subject of [DMR93, DMR]. We briefly summarize our MAAPR work in Section 7.5.

We reposition each \(p\) so that its origin is at the center of its MAAPR. We choose \(B\) to be the intersection of all \(p \in P\). This guarantees that \(B\) contains a rectangle with the minimum width and minimum height of all the MAAPRs. This, in turn, provides a lower bound on the area of the best breaking polygon.

Unfortunately, the reachable region \(R_B\) of the breaking polygon is not necessarily a superset of \(R_p\) for each \(p \in P\). Consequently, we are not guaranteed that if two components \(v_1\) and \(v_2\) in \(V_{P}\) are in different connected components of \(V_B\), then they do not interfere. In fact, there can be a significant amount of interference.

#### 7.4.2 Inking Polygons

We say polygon \(q\) inks polygon \(p\) if and only if \((p \ominus (-q)) \oplus q = p\). A set of polygons \(Q\) is an inking set of a set of polygons \(P\) if, for every \(p \in P\), \(\bigcup_{q \in Q} ((p \ominus (-q)) \oplus q) = p\).

\(^4\)We use the term breaking polygon because \(V_B\) “breaks”, or partitions, \(V_{P}\).
An inking set is valuable because \( R_Q = \bigcup_{q \in Q} R_q \) is a superset of each reachable region \( R_p \). Therefore, for \( p, p' \in \mathcal{P} \), if \( v \subseteq V_p \) and \( v' \subseteq V_p' \) are connected components, then \( v \oplus p \) and \( v' \oplus p' \) can intersect (and hence \( \text{Prob}(v, v') > 0 \)) only if they lie in the same component of \( R_Q \). Furthermore, since \( v \oplus p \) either lies entirely inside or entirely outside each component of \( R_Q \), we can simply perform a point-in-polygon test of any point of \( v \oplus p \). It is easy to generate this point without computing all of \( v \oplus p \).

For an inking set to be useful, \( R_Q \) should not be too much larger than \( R_P \). On the other hand, \( R_Q \) should be easier to compute. The first condition implies that we desire the inking set whose smallest member has maximum possible area. The second condition implies that we desire the inking set of minimum complexity (total number of vertices). The inking set of maximal area is the set \( \mathcal{P} \) itself. The inking set of minimum complexity is a single point, but if we specify non-zero area, then this is an open question. We desire the answer to a doubly open question: of the inking sets of minimum complexity, find the one whose smallest member has maximum area. The next section gives a lower bound on the complexity of an inking set based on the notion of a hitting set.

**Hitting Sets**

Let \( a, b, c \) be three consecutive vertices of some polygon \( p \in \mathcal{P} \) such that angle \( abc \) opens into the interior of \( p \). If an inking set \( Q \) inks \( p \), then there must be a set of consecutive vertices \( a'b'c' \) on some \( q \in Q \) such that angle \( a'b'c' \) is a subset of \( abc \): if \( q \) is translated to place \( b' \) on top of \( b \), then \( a' \) and \( c' \) are on or in the interior of angle \( abc \). We say that angle \( a'b'c' \) hits angle \( abc \).

We can map angle \( abc \) to an interval (arc) on the unit circle by translating \( b \) to the center of the circle and taking the arc in the interior of \( abc \). Clearly, \( a'b'c' \) hits \( abc \) if and only if its interval is a subset. We refer to the set of angles or intervals of \( \mathcal{P} \) as that set of arcs generated by all triples of consecutive vertices \( abc \) of polygons \( p \in \mathcal{P} \).

Given a set of intervals on the unit circle, a hitting point-set \( S \) is a set of points such that each interval contains at least one point of \( S \) in its interior. Clearly the complexity of \( Q \) is at least as great as the size of the minimum hitting point-set of the angles of \( \mathcal{P} \).

In the following discussion, let \( M \) be the total number of vertices in \( \mathcal{P} \).

**Theorem 7.4.1** A minimal hitting point-set for \( M \) intervals on the unit circle can be found in \( O(M^2) \) time.

**Proof:** We first give an \( O(M \log M) \) time algorithm for finding the minimal hitting set for a set of \( M \) intervals of the real line. Then we show how to use this algorithm as part of an \( O(M^2) \) time algorithm for intervals on a circle.

The first step in the line interval algorithm is to initialize the hitting set \( \mathcal{H} \) to \( \emptyset \). We then sort the intervals by increasing \( x \)-coordinate of their right endpoints, and process them in this order. For each interval \( i \), if \( \mathcal{H} \) contains a point that hits \( i \), then we do not change \( \mathcal{H} \). Otherwise, we add the right endpoint of \( i \) to \( \mathcal{H} \).

This algorithm runs in \( O(M \log M) \) time because we sort the intervals. It can be shown that each step of the algorithm yields the minimal hitting set (for the current set of intervals) whose rightmost point is furthest to the right.

[^5]: Actually, to avoid degeneracy, we add a point just slightly to the left of the right endpoint of \( i \).
Our algorithm for intervals on a circle first sorts the interval endpoints, creating a set of $2M$ event points and $2M$ “sub-intervals” (between event points). The algorithm has an outer loop which considers each of these $2M$ sub-intervals, in turn. We add a single hitting point to a sub-interval, and thus we can eliminate from consideration any interval that contains this sub-interval. We can now cut the circle anywhere in this sub-interval and then apply the line algorithm. This gives the minimal hitting set for this choice of sub-interval. After considering all $2M$ sub-intervals, we know which choice yields the minimum hitting set. We only have to sort once, so the running time is in $O(M^2)$.

We can generalize the notion of hitting point-set to hitting interval-set: recall an interval of $Q$ hits an interval of $P$ if it is a subset. Clearly, we can replace each point in the hitting point-set by the sub-interval in which that point lies. However, this might not yield the hitting interval-set whose minimum length interval is maximized. Since we do not want “skinny” inking polygons (to avoid inking narrow necks of the container), we want to maximize this minimum angle. This will also tend to maximize the minimum area of an inking polygon.

For a given arc length $L$, we can find the minimum hitting interval-set whose members are of length $L$. We can do this in $O(M^2)$ time using a modification of the point-set algorithm. The algorithm on the line is exactly the same, but “unraveling” the circle into a line becomes more intricate. For any given hitting set size $h$, we can do binary search on the value of $L$ to find the smallest value of $L$ whose minimum hitting interval-set is $h$ or greater. Since $L$ can take on at most $2M(2M - 1)$ values, we only have to run the $O(M^2)$ hitting interval-set algorithm $O(\lg M)$ times.

**Forming Inking Sets using Hitting Sets**

Given the minimal sized hitting interval-set $Q$ of $h$ intervals for $P$ (with maximum minimum length $L$), we show here how to construct an inking set consisting of a set of $h$ isosceles triangles and a single square. The isosceles triangles ink regions near the vertices of $P$, and the rectangle inks the remaining area. If $h_{\text{min}}$ is the minimum complexity of the inking set, then this algorithm yields an inking set of complexity $3h_{\text{min}} + 4$.

We first discuss the isosceles triangles. We associate an isosceles triangle with each hitting set interval $i$ as follows. Consider the set $S_i$ of angles $abc$ of polygons $p \in P$ which are hit by $i$. For each angle $abc$, find the maximum height of a “corner” isosceles triangle whose top is $b$ and whose sides are aligned with $ba$ and $bc$ such that the triangle is in the interior of $p$. Let $H_i$ be the minimum height over all vertices in $S_i$. Now, form an inking set polygon $T_i$ which is the isosceles triangle of height $H_i/2$ and top angle determined by $i$. $T_i$ can ink a “corner” isosceles triangle for $abc$ of height $H_i/2$.

The size of the square $s$ is constrained by two factors: 1) the narrowest neck in $P$, and 2) the narrowest (and most diagonal) angle in $P$. To treat the first constraint, we build a Voronoi diagram of each polygon in $P$, under the $L_1$ metric [Lee80]. If we surround each site with the largest square which fits, then $s$ must be smaller than the smallest of these squares. Let $w_1$ be the resulting width. The second constraint requires that we calculate $w_2 = \min(H_i \sin(\alpha/2)/\sqrt{2})$ over all $abc$ in $P$ with angle $\alpha$ and hitting interval $i$. The width of $s$ is $\min(w_1, w_2)$. 

135
7.5 MAAPRs

This section considers the geometric optimization problem of finding the maximum area axis-parallel rectangle (MAAPR) in an \( n \)-vertex general polygon. This rectangle problem arises naturally in applications where a quick internal approximation to a polygon is useful. One of its applications is finding breaking polygons (see Section 7.4.1).

Section 7.5.1 summarizes our MAAPR work from [DMR93, DMR]. This work gives an \( O(n \log^2 n) \) time algorithm for an \( n \)-vertex general polygon. Section 7.5.2 describes a naive algorithm\(^6\) which we use in practice to find a breaking polygon.

7.5.1 Summary of MAAPR Results

Here we summarize our work on MAAPRs, which is described in detail in [DMR93, DMR]. We present the fastest algorithmic results for general polygons with holes: a \( O(n \log^2 n) \) time algorithm. We also prove a lower bound for this type of polygon of time in \( \Omega(n \log n) \). In order to develop this algorithm, we first characterize the MAAPR for general polygons by considering different cases based on the types of contacts between the rectangle and the polygon.

We present a general framework for solving the two-contact case of the MAAPR problem, which dominates the running time for a variety of polygon types. The framework involves transforming the polygon, via vertex projection and inner rectilinear approximation, into another polygon for which we can solve a related problem. In the related problem, we are given a point set \( S \) and two subsets \( L \) and \( R \) of \( S \). The task is to find the largest rectangle containing no point of \( S \) which has lower-left corner in \( L \) and upper-right corner in \( R \). This problem can be solved by transforming it into a monotone\(^7\) matrix searching problem. Fast methods exist in the literature for finding the maximum element of such matrices.

Using these methods leads to the following MAAPR results: 1) \( \Theta(n) \) time for an xy-monotone polygon\(^8\), 2) \( O(na(n)) \) time for an orthogonally convex polygon\(^9\), where \( a(n) \) is the slowly growing inverse of Ackermann’s function, 3) \( O(na(n) \log n) \) for horizontally (vertically) convex polygons, 4) \( O(n \log n) \) for a special type of horizontally convex polygon.

\(^6\)We thank Jacqueline Huang for implementing this algorithm while she was an undergraduate at Harvard University.

\(^7\)A matrix is monotone if the column index of the row maximum moves to the right as the row index increases.

\(^8\)A simple polygon consisting of two xy-monotone chains is an xy-monotone polygon. A chain is xy-monotone if it is monotone with respect to both the x and y axes. A chain is monotone with respect to a line \( l \) if a line orthogonal to \( l \) intersects the chain in exactly one point [PS85].

\(^9\)An orthogonally convex polygon is both horizontally and vertically convex. This class contains the class of convex polygons.
consisting of two \(y\)-monotone chains on opposite sides of a vertical line, and 5) \(O(n \log^2 n)\) for general polygons.

### 7.5.2 MAAPR Characterization and Naive Algorithm

Here we characterize the MAAPR contained in a general polygon \(P\) by considering different cases based on the types of contacts between the MAAPR and the boundary of \(P\), and outline a naive algorithm for finding the MAAPR based on this characterization. Others have used contact classification for algorithmic development (see, for example, [MS90a, MOS85, DKO87]). The first part of this subsection classifies the types of contacts between the MAAPR and the boundary of \(P\). The second part introduces the notion of a `determining set` of contacts. The third part shows how to solve maximization problems associated with certain determining sets of contacts. The fourth part presents our characterization theorem for MAAPRs. The fifth part gives a naive algorithm based on the characterization theorem.

#### Types of Contacts

Intuitively, if an axis-parallel rectangle is inside \(P\), it has four degrees of freedom (parameters) and can “grow” until each of its four sides is stopped by contact with the boundary of \(P\). Contacts between the MAAPR and \(P\) are of two types: 1) a side of the MAAPR with a vertex of \(P\), and 2) a corner of the MAAPR with an edge of \(P\). In order to discuss the first type, we require the notion of a `reflex extreme vertex`, introduced in [SRW91].

**Definition 7.2** A vertex \(v\) of \(P\) is a **vertical reflex extreme vertex** if there exists \(\epsilon > 0\) such that, for an open ball \(\beta\) of radius \(\epsilon\) about \(v\) and \(S = \text{exterior}(P) \cap \beta\), either no point of \(S\) has \(y\)-coordinate above \(v\) or no point of \(S\) has \(y\)-coordinate below \(v\). A horizontal reflex extreme vertex is defined similarly.

For type 1 contacts, a reflex extreme vertex of \(P\) touches a side of the MAAPR and stops growth in one direction; we call this a `reflex contact`. Each reflex contact can remove one degree of freedom. Two reflex contacts with adjacent sides of the MAAPR fix a corner of the MAAPR. For type 2 contacts, a corner of the MAAPR touches an edge of \(P\) forming an `edge contact`.

#### Determining Sets of Contacts

**Definition 7.3** A set of contacts \(C\) is a **determining set of contacts** if the MAAPR \(R\) satisfying \(C\) has finite area and if the MAAPR \(R'\) satisfying any proper subset \(C' \subset C\) has greater or infinite area.

For example, a set of four reflex contacts, one on each side of the rectangle, is a determining set.

Note: A determining set determines the area of the MAAPR, but it does not necessarily determine a unique rectangle or MAAPR.

Within a determining set, we distinguish between two different subtypes of edge contacts. An edge contact is `fixed` if the set of constraints uniquely determines the point of contact with the rectangle (not necessarily the MAAPR). Otherwise, it is a `sliding contact`.

A fixed contact can arise when there is no freedom to slide along an edge because a reflex contact fixes a coordinate. For example, in Figure 7.5(a), the reflex contact of the
determining set fixes the x-coordinate of the edge contact, which completely determines the location of the edge contact. An edge contact with an adjacent side that has either a reflex or fixed contact must also be a fixed contact.

Two sliding edge contacts are dependent if the position of one determines the position of the other; otherwise they are independent. An independent sliding contact requires that the two adjacent sides of the MAAPR do not have any contact with \( P \) (see Figure 7.5(b)). A sliding contact adjacent to another sliding contact is dependent, because the two contacts must share a coordinate (see Figure 7.5(c)).

Maximization Problems

Here we examine maximization problems associated with certain determining sets of contacts. Finding the MAAPR associated with a determining set of contacts requires solving a maximization problem if the set contains a sliding contact. For a given set of contacts, the number of degrees of freedom is the number of undetermined parameters of the rectangle. Degrees of freedom within a determining set can only arise from sliding contacts because any other degree of freedom would result in a rectangle of infinite area, and therefore the contacts would not form a determining set. It follows that if a determining set consists of only reflex or fixed edge contacts, no maximization is required. For each independent sliding contact in the set, we can parameterize the associated edge. The maximization problems can then be classified based on the number of parameters.

1-Parameter Problems:

The set of 1-parameter maximization problems can be further subdivided according to the number of dependent sliding contacts.

The Basic 1-Parameter Problem

The simplest 1-parameter problem involves no dependent sliding contacts, just a single independent one. This is the basic 1-parameter problem, and it arises when one corner of the MAAPR has a sliding contact and the opposite corner is fixed. The basic 1-parameter problem can be solved by parameterizing the edge associated with the sliding contact and maximizing a quadratic in one variable. This can be solved in \( O(1) \) time.

An alternate constant-time solution to the 1-parameter problem is based on the following lemma, which demonstrates that the slope of the MAAPR diagonal depends only on the slope of the polygon edge. We assume here that the edge is neither vertical nor horizontal.

Lemma 7.5.1 (Slope Lemma) Given a point \( p \) and a line \( L \) with slope \( s \), the MAAPR with one corner at \( p \) and opposite corner at point \( q \) on \( L \) has diagonal \( \overline{pq} \), where the slope
of \( \overline{pq} = -s \).

In other words, if one corner of a MAAPR is incident on \( L \), the slope of the MAAPR’s diagonal is the negative of \( L \)’s slope.

\[ \text{Figure 7.6: Slope lemma} \]

**Proof:** Assume \( p \) is at the origin, and that the line \( L \) is given by \( ax + by = d \) (see Figure 7.6). W.l.o.g. assume \( L \) intersects the +x and +y axes. (Note: we ignore the degenerate cases of horizontal and vertical lines, for which the MAAPR is undefined.) The family of hyperbolas given by \( xy = c \) represents curves corresponding to rectangles of constant area. Moving away from the origin into the first quadrant, the hyperbola branch which is tangent to \( L \) provides the area \( c \) of the MAAPR associated with \( L \). Let \( (x_c, y_c) \) be the coordinates of the point at which \( L \) is tangent to the hyperbola \( xy = c \).

Now, the tangent to \( xy = c \) at \( (x_c, y_c) \) is equal to \( L \), so the normal to \( xy = c \) at \( (x_c, y_c) \) is equal to the normal to \( L \) at \( (x_c, y_c) \). The normal to \( xy = c \) is \( (F_x, F_y) = (y, x) \); evaluated at \( (x_c, y_c) \) it is \( (y_c, x_c) \). The normal to \( L \) at \( (x_c, y_c) \) is \( (a, b) \). Therefore,

\[ \text{slope}(\overline{pq}) = \frac{y_c}{x_c} = \frac{a}{b} = -\text{slope}(L). \]

\[ \square \]

Note that, as a consequence of this lemma, if \( p \) moves downward, \( q \) moves downward along its edge.

**Two Dependent Sliding Contacts**

Consider a pair of dependent sliding contacts which share a coordinate (w.l.o.g. \( y \)), and have an opposite reflex contact which determines one coordinate of a side, (w.l.o.g. \( y' \)) (see Figure 7.7(a)).

To find the MAAPR, we parameterize edge \( \overline{p_2p_1} \) by \( t \), yielding the following quadratic in \( t \) to maximize:

\[ \mathcal{F}(t) = (x' - x)(y - y') \]
\[ = (A + Bt)(C + Dt) \quad (7.1) \]
Figure 7.7: 1-Parameter problems with two and three dependent sliding contacts

where:
\[
\begin{align*}
A &= \frac{y_2 - b}{m} - x_2 \\
B &= \frac{y_1 - y_2}{m} + (x_2 - x_1) \\
C &= y_2 - y' \\
D &= y_1 - y_2 \\
m &= \frac{(y_4 - y_3)}{(x_4 - x_3)} \\
b &= y_4 - mx_4
\end{align*}
\]

Note that \(x'\) is obtained by solving the line equation associated with edge \(p_4p_5\).

Three Dependent Sliding Contacts

For the case of three dependent sliding contacts, (see Figure 7.7b), we again parameterize edge \(\overline{p_2p_1}\) and express \(x'\) in terms of \(y\) for edge \(\overline{p_4p_3}\). Similarly, we express \(y'\) in terms of \(x\) for edge \(\overline{p_5p_6}\). We again obtain a quadratic in \(t\) to maximize. The difference between this and the previous case is in the values of \(C\) and \(D\):
\[
\begin{align*}
C &= y_2 - m'x_2 - b' \\
D &= y_1 - y_2 + m'(x_2 - x_1) \\
m' &= \frac{(y_6 - y_5)}{(x_6 - x_5)} \\
b' &= y_5 - mx_5
\end{align*}
\]

The 2-Parameter Problem:

There is only one type of 2-parameter problem. It has two independent sliding contacts. The following lemma allows us to reduce a 2-parameter problem to a set of 1-parameter problems.

**Lemma 7.5.2** Let \(e_1\) and \(e_2\) be non-intersecting line segments. Consider the set \(E\) of empty axis-parallel rectangles which have diagonally opposite corners on \(e_1\) and \(e_2\). There is a maximum area rectangle in \(E\) with at least one corner at an endpoint of \(e_1\) or \(e_2\).

**Proof:** One way to prove the claim is to parameterize each edge to obtain a quadratic in two variables. It is easily shown that the resulting surface is a saddle surface, so that the maximum is achieved along the boundary of the unit patch, implying that the MAAPR has a corner at a vertex of one of the edges.
A geometric argument is more intuitive, however. Let \( c_1 \) on \( e_1 \) and \( c_2 \) on \( e_2 \) be opposite corners of a MAAPR, let \( r \) be the diagonal connecting \( c_1 \) and \( c_2 \), and, by way of contradiction, assume that neither \( c_1 \) nor \( c_2 \) is at a vertex of \( e_1 \) or \( e_2 \). Now replace \( e_1 \) and \( e_2 \) by the lines \( l_1 \) and \( l_2 \) containing them. Consider the set \( S \) of all line segments that connect \( l_1 \) and \( l_2 \) and are parallel to \( r \). The length of these line segments as a function of their distance from \( r \) is monotonic. If \( l_1 \) and \( l_2 \) are parallel, the length is constant. Otherwise the length increases moving away from \( r \) in one direction, and decreases in the opposite direction. Move away from \( r \) in the direction of increasing length (either direction if \( l_1 \) is parallel to \( l_2 \)) until either \( c_1 \) or \( c_2 \) is at a vertex of \( e_1 \) or \( e_2 \), and let \( r' \) be the new diagonal. Since \(|r'| \leq |r|\), the area of the rectangle whose diagonal is \( r' \) is at least as large as the area of the rectangle whose diagonal is \( r \), contradicting the assumption that the latter is a MAAPR.

Having established that the MAAPR has a corner at a vertex in this case, we can find the MAAPR by considering, in turn, each of the four endpoints of \( e_1 \) and \( e_2 \), solving the associated 1-parameter problems, and then comparing the four resulting 1-parameter MAAPR areas.

**Characterization Theorem**

To characterize the MAAPR, we examine the possible determining sets of contacts. By enumerating the reflex contacts between the MAAPR and \( P \), we derive the set of five cases shown in Figure 7.8.

Figure 7.8: Determining sets of contacts for the MAAPR

**Theorem 7.5.3** The determining set of the MAAPR of a general polygon \( P \) conforms (up to symmetry) to one of the five cases in Figure 7.8.
Proof: A determining set has, by definition, at most one reflex contact with each side of the MAAPR. For each of the possible numbers of reflex contacts, we show that the determining set of a MAAPR of that type conforms (up to symmetry) to one of the configurations shown in Figure 7.8. In each case, we must eliminate degrees of freedom beyond those of sliding contacts. We observe that a determining set cannot contain both two adjacent reflex contacts and the fixed contact between them; this would be redundant.

Case 4: In this case there is one reflex contact with each side of the MAAPR. Since each reflex contact removes one degree of freedom from the rectangle, this set of contacts is sufficient to determine the MAAPR. Removing any one reflex contact allows the rectangle to grow, so all four contacts are necessary.

Case 3: Three reflex contacts are not sufficient to determine the MAAPR, since the fourth side can move outward. We must add an edge contact which is fixed because it is adjacent to a reflex contact. There is only one choice of position for this contact, up to symmetry. It fixes the remaining side of the rectangle.

Case 2: Two reflex contacts can be either adjacent or diagonally opposite. In both cases, edge contacts are needed to determine the MAAPR. Since two degrees of freedom remain, the determining set contains at most two edge contacts.

In the adjacent case, we examine the possibility of edge contacts at the corners of the rectangle, excluding the corner between the two reflex contacts. There are two ways that two edge contacts can appear. In both cases both contacts must be fixed, so they fix the remaining sides of the rectangle. If only one edge contact appears, it must be a sliding contact; otherwise a degree of freedom remains. The sliding contact is diagonally opposite to the corner fixed by the reflex contacts. This contact is sufficient because it represents a basic 1-parameter problem.

In the opposite case, any edge contact is fixed; hence there must be two of them. There are two possible ways they can be configured. In both cases, all four sides of the rectangle are fixed.

Case 1: One reflex contact is not sufficient to determine the MAAPR. We can choose to place edge contacts at any three of the four corners of the rectangle (four would be redundant). If there are three edge contacts, they are all fixed and they completely determine the MAAPR. There are two ways to configure three such contacts. If there are two edge contacts, they can be adjacent or diagonally opposite. If they are adjacent, they must both be opposite the reflex contact; otherwise a degree of freedom remains. These adjacent contacts are dependent sliding contacts, and they determine the MAAPR because they represent a 1-parameter problem. If they are diagonally opposite, one is fixed, and the other is a sliding contact; this is also a 1-parameter problem. One edge contact is not sufficient, together with the one reflex contact, to determine the MAAPR.

Case 0: In this case there are no reflex contacts. One sliding contact is not sufficient to determine the MAAPR. If there are two sliding contacts, they must be opposite if they are to determine the MAAPR. This is a 2-parameter problem. If there are three edge contacts, they are all dependent sliding contacts. They are sufficient to determine the MAAPR because they represent a 1-parameter problem.
four edge contacts, they yield four equations in four unknowns, for which the MAAPR is completely determined. This therefore forms a set of fixed contacts.

\[\square\]

**Corollary 7.5.4** Given a determining set \( C \) for a MAAPR of a general polygon, it follows that \( 2 \leq |C| \leq 4 \).

**A Naive Algorithm**

Based on the above characterization, we can find the MAAPR in a general polygon by finding the MAAPR under the constraints of each of the five cases and selecting the largest one.

**Theorem 7.5.5** For each determining set of contacts in Figure 7.8, the MAAPR can be found in constant time.

**Proof:** A reflex contact yields, in constant time, one of the four parameters of the rectangle, as does a fixed contact. In all cases we can process these contacts first. The remaining situations are all either 1-parameter problems or 2-parameter problems, which can be solved in constant time because they only require maximizing a constant number of quadratic forms.

\[\square\]

A naive MAAPR algorithm can use Theorem 7.5.5 and supply it, in each case, all possible determining sets for \( P \). These can be identified using an algorithm with up to four nested loops, one for each element of the determining set. For each MAAPR candidate, we can check if it is empty (i.e. contains no edges or vertices of \( P \)) in \( O(n) \) time. We conclude:

**Theorem 7.5.6** The MAAPR of an \( n \)-vertex general polygon can be found in \( O(n^5) \) time.

### 7.6 Summary and Conclusions

This chapter has discussed our container limiting and decomposition work. Limiting is based on the reachable regions of the items. A reachable region is found by using restrictions combined with the opening morphological operator. Our decomposition process allows the user to control the probability that gaps will overlap each other. For efficiency reasons, we introduce the notion of breaking polygons and inking polygons. We have implemented the breaking polygon approach and found it to be practical for forming limited gaps. Our implementation uses a naive algorithm for finding the maximum-area axis-parallel rectangle (MAAPR) in a polygon. Our theoretical work on the MAAPR problem yields the fastest known asymptotic running time for this problem: \( O(n \log^2 n) \) time for an \( n \)-vertex polygon. Figure 7.9 is a screen dump which shows limited gaps for an example from apparel manufacturing.

We note that in applications which operate on \( U_{ij} \)s and do not directly operate on the containers themselves, the methods we present for clustering together \( U_{ij} \)s can be used without carrying out the final construction of the limited gaps themselves.
Figure 7.9: Limited gaps example


8.1 Introduction

In this chapter we address four minimal container problems: 1) strip packing, 2) finding minimal square enclosures, 3) finding minimal rectangular enclosures, and 4) finding tight convex and nonconvex enclosures.

In Chapter 6 we cited the following definition of an enclosure problem given by Chang and Yap [CY86]. Enc(P, Q, μ): Given \( P \in \mathcal{P} \), find the \( \mu \)-smallest \( Q \in \mathcal{Q} \) that encloses \( P \), where \( \mathcal{P} \) and \( \mathcal{Q} \) are families of polygons, and \( \mu \) is a real function on polygons such that:

\[
\forall Q, Q' \in \mathcal{Q}, \quad Q' \subseteq Q \quad \Rightarrow \quad \mu(Q') \leq \mu(Q)
\]

This definition is not sufficient for our purposes. We need a more general definition of an enclosure problem. Our definition replaces \( P \) with \( \mathcal{P} \), where \( \mathcal{P} \) is a tuple of elements of \( \mathcal{P} \). We also let \( \mathcal{M} \) be a set of polygon transformations. We define:

Enc(\( \mathcal{P} \), \( \mathcal{M} \), \( \mathcal{Q} \), \( \mu \)): Given \( \mathcal{P} = (P_1, P_2, \ldots, P_k) \), where \( P_i \in \mathcal{P} \), find the \( \mu \)-smallest \( Q \in \mathcal{Q} \) that encloses \( P_{\mathcal{M}} \), where \( \mathcal{P} \) and \( \mathcal{Q} \) are families of polygons, \( \mathcal{M} \) is a set of polygon transformations, \( P_{\mathcal{M}} \) is the transformation of each item in \( \mathcal{P} \) by a (possibly different) transformation in \( \mathcal{M} \), and \( \mu \) is a real function on collections of polygons such that:

\[
\forall Q, Q' \in \mathcal{Q}, \quad Q' \subseteq Q \quad \Rightarrow \quad \mu(Q') \leq \mu(Q)
\]

Work on minimal enclosures in computational geometry uses the identity transformation for \( \mathcal{M} \) and does not allow a collection of polygons. The particular two-dimensional enclosure problems we address here let \( \mathcal{P} \) be a collection of polygons, \( \mathcal{M} \) be translations, \( \mu \) be the area function, and \( \mathcal{Q} \) vary depending on the problem. For example, in strip packing, \( \mathcal{Q} \) is the set of axis-parallel rectangles of fixed width.

From the discussion of related work in layout, we see that solutions to minimal enclosure problems are often required for the first stage of a multi-stage layout problem. Minimal container problems also arise when finding the best stock sheet size for a given set of items, or the best cloth width in garment manufacturing. Section 6.3.1 discusses computational geometry work on various types of enclosure problems for single items. Current solutions to
minimal enclosure problems for more than two arbitrary nonconvex items often allow the
items to rotate. However, because they use greedy heuristics, they cannot provide a lower
bound on the size of the enclosure. Popular approaches either place small items in gaps
inside the bounding box of two large items [AA76], or solve the problem incrementally by
adding one item at a time [DT87, DJ95].

Our containment-based solutions to minimal container problems do not allow the items
to rotate, but, in contrast to heuristics, we establish an actual lower bound on the size of
the container for multiple items. Our approach to finding minimal enclosures is to find an
optimal value for a parameter via binary search combined with containment. The idea of
solving a geometric optimization problem using binary search on a parameter is found, for
example, in Chazelle’s seminal paper on polygon containment [Cha83]. We observe that the
binary search can be accelerated by using compaction techniques, such as those developed
by Li and Milenkovic [Li94, ML, LM93c, LM93a], to improve a feasible solution and hence
update the upper bound. Our hybrid containment algorithm is appropriate for minimal
enclosure problems. It is extremely fast for loose fits, and so the initial stages of the search
are fast.

We propose the following general approach to minimal enclosure problems involving a
single parameter. Our approach is a binary search, with compaction\(^1\) used to update the
upper bound obtained by containment. Note that the binary search approach works only
if the function being minimized decreases monotonically as the parameter decreases. In
this binary search, the “function” can be thought of as a binary one which returns 1 if the
containment problem is feasible and 0 otherwise.

\[
\text{MINIMAL-ENCLOSURE}(P, k) \\
\text{do} \\
\quad t_{\text{max}} \leftarrow \text{large initial value} \\
\quad t_{\text{min}} \leftarrow 0 \\
\quad \epsilon \leftarrow \text{tolerance value} \\
\text{while } (t_{\text{max}} - t_{\text{min}} > \epsilon) \\
\quad t_{\text{current}} \leftarrow (t_{\text{max}} + t_{\text{min}})/2 \\
\quad Q \leftarrow \text{container of size } t_{\text{current}} \\
\quad \text{if } (\tau \leftarrow \text{CONTAINMENT}(P, k, Q)) \\
\quad \quad \text{place polygons according to configuration } \tau \\
\quad \quad t_{\text{max}} \leftarrow \text{COMPACTION}(P, k, Q) \\
\quad \text{else} \\
\quad \quad t_{\text{min}} \leftarrow t_{\text{current}} \\
\text{return } t_{\text{max}}
\]

8.1.1 Overview

In Section 8.2 we use binary search on length together with containment to solve a two-
dimensional strip packing problem. In Section 8.3, binary search on length is combined with
containment to find tight (axis-parallel) square enclosures for collections of items. We extend
this approach in Section 8.4 to finding tight (axis-parallel) rectangular enclosures, which is
a common preprocessing step in many layout algorithms which pack non-rectangular items.
It can also be applied to the problem of finding the best stock sheet size for a given set

\(^1\)Our current implementations do not use compaction; adding it is a subject of future work.
of items, or the best cloth width in garment manufacturing. In higher dimensions, this technique could help determine the best packaging for a product. In Section 8.5, we show that removing unreachable portions of a tight rectangular enclosure can yield tight convex and nonconvex enclosures which preserve all the containment solutions for the rectangle. Section 8.6 summarizes this chapter and states conclusions.

8.2 Strip Packing

Here we use binary search on length together with containment to solve a two-dimensional strip packing problem. We illustrate how containment can be used for strip packing by solving an incremental layout task from the apparel industry. Figure 8.1 shows an incremental layout task. A garment containing 8 items has already been laid out, and the task is to add a second garment to this layout while minimizing the length of the strip. This type of problem is becoming more important as the apparel industry moves in the direction of smaller lot sizes and customized garments.

![Figure 8.1: Incremental layout task](image)

Let the length be \( \ell \). Figure 8.2 shows a very loose initial layout corresponding to \( \ell = 101.24 \) inches. Note that only the six large items are placed. The two small ones can easily be inserted into the final layout in a number of places, so our search on length ignores them. Figure 8.3 shows the tightest layout obtained for this problem; \( \ell = 77.09 \) inches.

One alternative to this approach is to use an improvement strategy. Improvement methods for layouts of nonconvex items are surveyed by Dowsland and Dowsland [DD]. “Metaheuristic” techniques such as simulated annealing, genetic algorithms, and tabu search can be applied. However, they are unable to establish a lower bound on \( \ell \). Another improvement strategy uses the compaction techniques of Li and Milenkovic [Li94, ML, LM93c, LM93a, MDL92, LM93d]. Compaction is also unable to establish a lower bound. Compaction alone can be applied in a given direction to simulate a gravitational force. However, this technique can only produce a continuous motion of the items. It is unlikely to produce the tightest possible strip packing, especially if the items are nonconvex. In order to produce a tight
Figure 8.2: Loose layout

Figure 8.3: Final layout
packing, we must allow the items to move discontinuously as \( \ell \) decreases. This “jumping” behavior can be obtained using forms of compaction based on mixed integer programming, but current implementations are currently too slow to be practical.

We compare our solution based on binary search and containment with the solution obtained by first compacting the layout of Figure 8.2 and then placing the two small items. Compaction gives \( \ell = 85.48 \) (see Figure 8.4), whereas \( \ell = 77.09 \) for the containment approach.

Although the binary search technique obtains a tighter layout than compaction alone, compaction can still play an important role in this problem, and in all our enclosure problems. It can certainly help to update an upper bound quickly, thereby accelerating the binary search, as noted in Section 8.1.

### 8.3 Minimal Square Enclosures

In our second problem, binary search on a scaling factor can be combined with containment to find the minimal enclosing (axis-parallel) square for a set of polygonal items, where the orientation of each polygon is fixed. One parameter suffices to characterize the square. We denote this by \( \ell \); it is equal to \( 1/2 \) the length of a side of the square. We use binary search on \( \ell \) combined with containment.

We begin with a square which is guaranteed to be large enough to hold the items. Our first guess consists of a square whose side is the maximum of the height and width of the bounding box of the following shape:

\[
((P_1 \oplus -P_2) \oplus P_2) \ldots \oplus -P_k) \oplus P_k
\]

The \( k \) items can certainly fit into this square.

Because the area function \( \mu(\ell) \) decreases monotonically as \( \ell \) decreases, minimizing \( \ell \) yields the minimum area square enclosing the items. The same is true if we apply a pure scaling to any polygon. Thus, the same method can be used, for example, to find the minimal regular hexagon (of fixed orientation) enclosing a collection of items.
Figure 8.5: Minimal square enclosure for 3 nonconvex items
Figure 8.5 shows the minimal square enclosure for three items. An improvement method which only allows continuous motion of the items is incapable of finding this square enclosure because there is no continuous motion of the items which can “nest” the rectangle inside the “C”-shaped item.

Figure 8.6 shows a minimal square enclosure for four nonconvex apparel pattern pieces. This runs in 122 seconds on a 50MHz SPARCstation. Each of the four polygons has 65 vertices. The pieces are identical, except that two are rotated by π. The configuration produced by the containment algorithm is a repeating pattern. Indeed, one possible application of our minimal enclosure work is to identify good repeating patterns for identical nonconvex objects. Although much work has been done on lattice packings for convex objects, there is still room for work on finding good repeating patterns for identical nonconvex objects. It would be interesting to experiment with different types of regular polygonal enclosures (e.g. square, hexagon) for a given set of identical items to see which gives the best result.

Figure 8.7 shows the minimal square for five items; this takes 50 seconds on a 50 MHz SPARCstation.

---

\[\text{Figure 8.6: Minimal square enclosure for 4 items}\]

\[\text{We thank Matt Buonomano of Gerber Garment Technologies for suggesting this application.}\]
Figure 8.7: Minimal square enclosure for 5 nonconvex items
Implementation note: an appropriate form of compaction for the square enclosure problem is one which gravitates items toward the center of the square.

8.4 Minimal Rectangular Enclosures

We extend this approach to finding tight rectangular enclosures, which is a common preprocessing step in many current algorithms which pack non-rectangular items. It can also be applied to the problem of finding the best stock sheet size for a given set of items, or the best cloth width in garment manufacturing. For layout algorithms which build strips [AA76] or columns [MDL92] of items, this approach can help determine the best cluster. In higher dimensions, this technique could help determine the best packaging for a product.

Characterizing a rectangle requires two parameters. One could parameterize by aspect ratio $\alpha$ and a scaling factor $\iota$. Unfortunately, although $\mu(\iota)$ is monotonic with respect to $\iota$, it is not monotonic with respect to $\alpha$. Thus, we cannot perform binary search on $\alpha$. Instead, we can sample $\alpha$.

We choose instead to parameterize the rectangle by height $h$ and width $w$. In this case, as well, the area function is only monotonic with respect to one of the parameters. We choose to sample $w$ and perform binary search on $h$ for each sampled value of $w$. It is straightforward to obtain upper and lower bounds on $w$. Let $h_{\text{max}}$ be the maximum height of any item, and $w_{\text{max}}$ be the maximum width. Then, one bound on $w$ can be obtained by letting the width of the container be $w_{\text{max}}$, and using binary search on height combined with containment to find the minimum associated height. The other bound is achieved by letting the height of the container be $h_{\text{max}}$, and using binary search on width combined with containment to find the minimum width $w_{\text{min}}$. The minimal rectangular enclosure algorithm contains an outer loop which samples $w$ in between the two bounds. For each $w$, it invokes a procedure which performs binary search on $h$ for a particular value of $w$.

Implementation note: there are two ways to use compaction to accelerate the minimal rectangle algorithm. First of all, downward compaction can be used within the binary search on $h$ for fixed $w$. The second application of compaction relies on the fact that the minimal $h$ as a function of $w$ increases monotonically as $w$ decreases. “Steps” in the function can be identified using compaction, as follows. After the minimal $h$ is obtained for a given $w$, place the items within a $w$ by $h$ rectangle and apply leftward compaction to the right-hand boundary of the rectangle. Hopefully, compaction will move $w$ beyond the next sample point so that a binary search is avoided. For this reason, we suggest that $w$ be sampled in decreasing fashion.

We present three examples of finding minimal rectangles. The first example is three rectangles in Figure 8.8. This was found in seven seconds on a 50MHz SPARCstation. The second is three identical nearly nonconvex apparel pattern pieces in Figure 8.9. This required less than five minutes. The third is three nonconvex polygons in Figure 8.10.

Milenkovic has recently suggested [MD95] an alternative method for finding minimal rectangular enclosures of multiple items. He poses the following problem: given shapes $P_1, P_2, \ldots, P_k$ and area $\alpha$, is there a rectangle of area $\alpha$ which contains translated copies of the shapes? He solves the problem by modifying the hybrid containment algorithm. With the modification, one can find the minimum $\alpha$ numerically by binary search.
Figure 8.8: Tight rectangular enclosure for 3 convex items
Figure 8.9: Tight rectangular enclosure for 3 nearly identical nonconvex items
Figure 8.10: Tight rectangular enclosure for 3 nonconvex items
8.5 Tight Convex and Nonconvex Enclosures

Section 8.3 observed that one-parameter binary search together with containment can not only find the minimal enclosing square, but it can also find the minimal scaling factor applied to any polygonal enclosure.

In this section we introduce a different way to obtain tight convex and nonconvex enclosures. One can easily generate tight enclosures from a configuration of items within a tight square or rectangular enclosure \( C \). For example, one can use the convex hull of the items to obtain a convex enclosure.

However, if we want the enclosure to preserve all the valid configurations of \( C \), we need a different approach. We use the \textit{reachable region}, as defined in Chapter 7, to form a container. The procedure is simple. First, find \( U \) for the items with respect to \( C \). Next, apply restrictions to \( U \). Then form: \( C' = \bigcup_{1 \leq j \leq k} (U_{ij}^* \oplus P_j) \). This \( C' \) is a container which has all the valid configurations associated with \( C \). It reveals portions of \( C \) which cannot be reached by any of the items when they are placed simultaneously in \( C \).

Figure 8.11 shows a tight nonconvex container obtained from the rectangular enclosure of Figure 8.10. In Figure 8.11, the container is the black filled region inside the rectangle.
and the outlines of the original items are shown to the right for reference. This container was formed using reachable regions, which were, in turn, obtained from $U_j^s$. All of the white space inside the rectangle and outside the black region is unreachable, and can be removed with no consequences. In the figure, we can easily see that three of the four corners of the rectangle are superfluous for the three given items. The convex hull of this container is a convex container which has the same valid configurations as the rectangle.

This technique can be applied for container design. In some cases, it can be used to help humans solve containment problems. Observe that the container and items of Figure 8.11 immediately suggest a legal configuration to the human eye.

### 8.6 Summary and Conclusions

This chapter has addressed four minimal container problems: 1) strip packing, 2) finding minimal square enclosures, 3) finding minimal rectangular enclosures, and 4) finding tight convex and nonconvex enclosures. Our general approach to solving these problems is to use binary search combined with containment. The binary search can be accelerated using compaction techniques.

Previous work on minimal enclosures in computational geometry considers only one item. Work in the layout literature allows multiple items, but is heuristic (and typically greedy) for more than two items. Our work represents the first non-heuristic multiple-item minimal enclosure work for more than two items. Our containment-based solutions to minimal container problems do not allow the items to rotate, but, in contrast to heuristics, we establish an actual *lower bound* on the size of the container for multiple items.
Chapter 9

Maximal Item Problems

9.1 Introduction

In this chapter we give four examples of maximal item problems: 1) given a container and a collection of items, finding maximally scaled copies of items which fit in the container, 2) given a container and a collection of $\leq k_{practical}$ items for which the sum of the item areas does not exceed the area of the container, finding a “large” sub-collection which fits in the container, 3) given a container and a collection of items for which either the sum of the item areas exceeds the area of the container or the number of items is $> k_{practical}$, finding a “large” sub-collection of $\leq k_{practical}$ items which fits in the container, and 4) problem (3) for a collection of containers.

For problem (1) we use binary search combined with containment. In problem (2) the given items cannot be scaled; they can only be included in or excluded from the set. The key to solving problem (2) is to carefully choose which item to eliminate from an infeasible subset. We reject the most “troublesome” item, which is identified using the results of the overlap elimination evaluation method of the hybrid containment algorithm. For problem (3), we randomly construct a set of items and solve problem (2). This yields a rich collection of sets which fit into the container. For each set of size $n$ which fits, we automatically have $2^n$ subsets which fit and which need not be individually represented. This work provides the first practical method for constructing sets of up to ten polygons which fit in a container.

For problem (4), we solve problem (3) for each container, and then apply a look-ahead assignment strategy. This is a pre-packing strategy. Containment thus allows us transform a geometric optimization problem into a combinatorial optimization problem. Our results challenge the existing belief that pre-packing methods are impractical for layout [Arb93].

9.1.1 Overview

Section 9.2 solves problem (1). Section 9.3 solves problem (2). Section 9.4 solves problem (3). Section 9.5 solves problem (4). Section 9.6 summarizes this chapter and states conclusions.
9.2 Maximally Scaled Items

In this section we are concerned with positive homothets of items. A positive homothet of an item \( P_i \) is a scaled copy of \( P_i \) with scaling factor \( \gamma_i \geq 1 \). We do not allow rotation of the homothets. Given a container, a set of \( k \) items which has a valid configuration in the container, and a real number \( \zeta \), we pose the following general problem: find a valid configuration of positive homothets of the items in the container for which the sum of the areas of the homothets is at least \( \zeta \). We call this the maximal positive homothet problem (MAX-HOMOTHET) for a container. Maximizing the value of \( \zeta \) yields a “tight” packing of the container.

Claim 9.2.1 The maximal positive homothet problem is NP-hard.

Proof: We reduce an instance of the NP-complete PARTITION problem to an instance of MAX-HOMOTHET. An instance of PARTITION has a set \( A \) of items. Each \( a \in A \) has a positive integer size \( s(a) \). Let \( \sum_{a \in A} s(a) = Q \). We build a two-component container. Each component is a rectangle of height 1 and width \( Q/2 \). For each \( a \in A \), construct a rectangle of height \( 1/2 \) and width \( s(a)/2 \). Now, if \( |A| > 1 \), scale each of these rectangles by \((1 + \epsilon)\), where \( 0 < \epsilon < 1 \) and

\[
\sum_{a \in A \setminus \alpha} (s(a)(1 + \epsilon)) \leq Q
\]

where \( \alpha \) is the item in \( A \) of minimum width.

This guarantees that the scaled rectangles fit into the container. Let \( \zeta = Q \). Now solve MAX-HOMOTHET. We claim MAX-HOMOTHET has a solution if and only if PARTITION has a solution. For, if PARTITION has a solution, then there exists a valid configuration of rectangles of height 1 and width \( s(a) \), for \( a \in A \). This configuration has area = \( Q \), and it represents a scaling of at least 1 for each item we constructed for MAX-HOMOTHET, so it is a solution to our instance of MAX-HOMOTHET.

Now we argue that if MAX-HOMOTHET finds a solution, then that solution is a solution to PARTITION. Because we scaled by \((1 + \epsilon)\), no two items can be stacked vertically within a component of the container. This means that, in order to completely fill the container, each item must be restored to its full height. This yields a solution to PARTITION. \( \square \)

We examine heuristics for obtaining a good, but not necessarily optimal, solution to the problem of maximizing the value of \( \zeta \) for MAX-HOMOTHET. One possible approach is to force all items to have the same scaling factor. This is easy to solve using binary search on the scaling factor. It has the advantage that a solution can be found either by scaling the container or the pieces. If the number of vertices in the container is much smaller than the sum of the number of vertices in the items, then it is appropriate to scale the container.

Another possibility is a greedy approach. A greedy scaling algorithm would fix the scaling factor of one item during each iteration. The fixed item would be the one whose positive homothet can have maximum area.

Each of these strategies can be appropriate under different circumstances. To illustrate this, we consider a maximal homothet problem for a collection of identical circular items and a circular container. Suppose the container has a radius of 3, and there are 7 items. It is easy to show that if each of the given circles has radius less than .4, then the area of the
positive homothets is greatest under a greedy strategy. If, however, each circle has radius greater than \( \frac{1}{4} \), the greedy approach is inferior to applying the same scaling factor to each circle. The left portion of Figure 9.1 shows a common scaling factor applied to the circles, and the right portion shows a greedy scaling.

Yet another approach is an enhancement to the common scaling factor approach, suggested by Milenkovic [Mil95a]. In this approach one maintains a set \( Q \) of items whose scaling factors are fixed, and a set \( S \) of items whose scaling factors are not yet fixed. While \( S \neq \emptyset \), the algorithm scales items in \( S \) up as much as possible, by the same scaling factor \( \gamma \). (The scaled items of \( S \) must fit into the container along with the items in \( Q \).) Then, for each item of \( S \), if it cannot be scaled more than by \( \gamma \), we remove it from \( S \) and add it to \( Q \).

We present an approach to the maximal homothet problem which has most of the advantages of both the greedy and the enhanced common scaling factor approaches and is superior to both on some inputs. As in the enhancement to the common scaling factor algorithm, we maintain a set \( Q \) of items whose scaling factors are fixed, and a set \( S \) of items whose scaling factors are not yet fixed. We use the area of \( \text{SSGR}(U_0) \) as a rough measure of the “freedom” of an item \( P \). We associate a parameter \( \iota \) with the area of these \( U_0 \)'s. The value of \( \iota \) is a real number whose minimum and maximum values are the minimum and maximum areas, respectively, over all \( \text{SSGR}(U_0) \), for \( 1 \leq i \leq k \). For a given value of \( \iota \), we define a subset \( S(\iota) \) of \( S \): \( S(\iota) = \{ P_i \in S | \text{area}(\text{SSGR}(U_{0,i})) \geq \iota \} \). As stated, this causes both \( P_i \) and \( P_j \) to be in \( S(\iota) \) if \( \text{area}(\text{SSGR}(U_{0,i})) = \text{area}(\text{SSGR}(U_{0,j})) \geq \iota \). Thus, whenever an item is in \( S(\iota) \), all identical copies of it are also in \( S(\iota) \). To avoid this difficulty, we modify the area function slightly so that, for \( i \neq j \), \( \text{area}(\text{SSGR}(U_{0,i})) \neq \text{area}(\text{SSGR}(U_{0,j})) \).

As \( \iota \) ranges from \( \iota_{\text{max}} \) to \( \iota_{\text{min}} \), \( S(\iota) \) changes \( k \) times. For each of these \( k \) events, we scale \( S(\iota) \) up as much as possible, using a common scaling factor for the items in \( S(\iota) \). The maximum common scaling factor is found using binary search on a parameter \( \gamma \). For the maximum common scaling factor, we record the total area of the scaled (and unscaled) items. Let \( \gamma_{\text{max}} \) be the scaling factor associated with the event of maximum total area, and \( S(\iota)_{\text{max}} \) be the associated subset. We scale items in \( S(\iota)_{\text{max}} \) by \( \gamma_{\text{max}} \). If an item in \( S(\iota)_{\text{max}} \) cannot be scaled by more than \( \gamma_{\text{max}} \), we remove it from \( S \) and add it to \( Q \). We then update the set of events and repeat the process.

Pseudo-code for the complete algorithm is given below. The tolerance for the search on \( \gamma \) is \( \epsilon_{\gamma} \). We denote the scaling of each element of a set \( S \) by a scaling factor \( \gamma \) as follows:

\[
\]
$S \cdot \gamma$.

**MAX-HOMOTHET(P)**

$Q \leftarrow \emptyset$

$S \leftarrow P \setminus P_0$

$t_{\text{max}} \leftarrow \max_{P_i \in S} \{\text{area}(S \cap \text{GR}(U_0, P_i))\}$

$t_{\text{min}} \leftarrow \min_{P_i \in S} \{\text{area}(S \cap \text{GR}(U_0, P_i))\}$

Form event list

while $(S \neq \emptyset)$

$\gamma_{\text{best}} \leftarrow 1$; $t_{\text{best}} \leftarrow t_{\text{max}}$

for each event $i$ in event list

$S(i) \leftarrow \{P_i \in S | \text{area}(S \cap \text{GR}(U_0, P_i)) \geq t_i\}$

$\gamma \leftarrow \text{SCALABLE}(S(i), P_0, Q \cup (S \setminus S(i)))$

if (total area for $\gamma >$ total area for $\gamma_{\text{best}}$)

$t_{\text{best}} \leftarrow t_i$; $\gamma_{\text{best}} \leftarrow \gamma$

$S(t_{\text{best}}) \leftarrow \{P_i \in S | \text{area}(S \cap \text{GR}(U_0, P_i)) \geq t_{\text{best}}\}$

$S(t_{\text{best}}) \leftarrow S(t_{\text{best}}) \cdot \gamma_{\text{best}}$

$S(t_{\text{best}}) \leftarrow \{P_i \in S(t_{\text{best}}) | \text{SCALABLE}([P_i], P_0, Q \cup (S \setminus \{P_i\})) = 1\}$

$Q \leftarrow Q \cup S(t_{\text{best}})$

$S \leftarrow S \setminus S(t_{\text{best}})$

**SCALABLE(S', C, S')**

$\gamma_{\text{max}} \leftarrow \gamma_{\text{init}}$

$\gamma_{\text{min}} \leftarrow 1$

while $(\gamma_{\text{max}} - \gamma_{\text{min}} > \epsilon)$

$\gamma_{\text{current}} \leftarrow (\gamma_{\text{max}} + \gamma_{\text{min}})/2$

if $(S' \cdot \gamma_{\text{current}} \cup S') \in C$

$\gamma_{\text{min}} \leftarrow \gamma_{\text{current}}$

else

$\gamma_{\text{max}} \leftarrow \gamma_{\text{current}}$

return $\gamma_{\text{min}}$

Here is a simple way to establish $\gamma_{\text{init}}$. Set $\gamma_{\text{init}} = (\text{area}(C)/\sum_{P_i \in S' \cup S'} \text{area}(P_i)) + \epsilon$.

We stated above that this algorithm possesses most of the advantages of the greedy and the enhanced balanced strategies. Here we elaborate on this point. We first show how our algorithm is similar to the greedy algorithm. Our event list contains an event which corresponds to scaling up a single item. If we included an event for scaling up each $P_i$ individually, we would be testing all the possibilities considered by the greedy algorithm. In contrast, our algorithm has only one single-item scaling event. However, the scaled item is the $P_i$ whose $U_0$ has the largest area. This is a rough measure of the “freedom” of an item. When all items are identical, our algorithm clearly makes the same choice as the greedy algorithm. We expect that in many cases, in practice, it will make the same choice as the greedy algorithm. Figure 9.2 gives an example of MAX-HOMOTHET behaving in a greedy fashion for three items. In this example, only one of the items is scaled up; its scaling factor is 1.97.
Figure 9.2: Scaling up items in a greedy fashion
We now show how our algorithm is similar to the enhanced balanced strategy. Our event list contains an event which corresponds to scaling up all items simultaneously by the same scaling factor. This is also done by the enhanced balanced strategy. In addition, we only fix the scaling factor for an item if it cannot be scaled up further. This is also a feature of the enhanced balanced algorithm. Figure 9.3 gives an example of MAX-HOMOTHET behaving like the enhanced balanced strategy. In this example, all of the items are scaled up by a factor of 1.09.

There exist inputs for which our algorithm obtains a better solution than the greedy algorithm. A simple example is the previously mentioned circle problem, in which the container is a circle of radius 3, and each of 7 items is a circle of radius greater than 4. In general, if the solution found by the enhanced balanced strategy is better than the solution found by the greedy strategy for a given input, then our algorithm’s solution will be better.
than the greedy algorithm's solution for that input.

There also exist inputs for which our algorithm obtains a better solution than the enhanced balanced algorithm. One such input consists of a circular container of radius 3 and 7 circular items, each of radius less than .4. Because the items are identical, our algorithm makes the same choice as the greedy algorithm in this case.

Our algorithm not only combines the power of the greedy and enhanced balanced strategies. It also considers possibilities not considered by either of these two strategies. These possibilities correspond to events which scale up $j$ items simultaneously, for $1 < j < k$. Figure 9.4 is an example in which MAX-HOMOTHET scales up two items by the same factor of 1.47; the third item is not scaled.

![Figure 9.4: Scaling up two items](image)

Implementation note: In applications where a scaling factor is involved, one can use the subset restriction of Section 3.5 to attempt to speed up the containment algorithm.
9.3 An Elimination Technique

In this section we consider the following problem: given a container and a collection of \( k \leq k_{\text{practical}} \) items for which the sum of the item areas does not exceed the area of the container, find a large\(^1\) subset of the items which fits in the container. An example of such a problem is illustrated by Figure 9.5. In this figure, we are given nine items and must find a subset of them which fits into the container. The container is depicted as a black filled region. The given items cannot be scaled; they can only be included in or excluded from the set.

We present here a heuristic approach to this problem which is based on an elimination technique, and is therefore a top-down method. The flow of the procedure is shown below, for a container \( C \) and an initial collection of items \( S \). The advantage of this top-down approach is that a subset is found using at most \( k \) calls to containment. Of course, once a subset of size \( n \) is found, we obtain \( 2^n \) subsets for free, and need not represent them separately.

\[
\text{GENERATE-SUBSET}(S, C)
\]

\[
\text{while } S \text{ does not fit into } C \quad S \leftarrow S \backslash \text{CHOOSE}(S)
\]

\[
\text{return } S
\]

However, nothing is ever truly for free. The price we pay here is that we must resolve the difficult issue of how to choose an item to remove from the subset. If this is not done well, the following scenario can easily occur. Suppose we always eliminate the item of smallest area. If a small number of the largest items in the subset cannot fit together into the container, the subset produced by our procedure might only contain one or two items. This might not pack the container tightly.

![Figure 9.5: Find a “large” subset which fits](image)

Intuitively, it makes sense to ask why a given infeasible subset does not fit. If the bottleneck can be found, we can eliminate one of the items involved in the bottleneck. We know how to obtain a good answer to this question, because when the hybrid containment algorithm fails, it can give us information about the local overlap minimum it has obtained. Recall that our distance-based subdivision method of Section 4.5.4 selects a \( U_{ij} \) to subdivide based on feedback from the evaluation step. In that case, we subdivide the \( U_{ij} \) which is

\(^1\)One measure of size is the number of items; another is the total area of the items.
associated with the greatest overlap between two items. These two items are, in some sense, the most “troublesome”. In the current application, we can eliminate one of these two items from the subset. (We choose to eliminate the smaller of the two.) This approach works well. Figure 9.6 shows a solution to the subset problem of Figure 9.5. Our approach finds a collection of seven items which fit into the container. This packing is fairly tight.

Implementation note: To control running time, it is possible to limit the number of hypotheses examined by the hybrid algorithm. This causes it to give up early in some cases. However, because every hypothesis evaluation produces a local overlap minimum, we still obtain useful overlap information when the algorithm terminates early.

\section{9.4 Randomized Subset Construction}

Here we address the following problem: given a container and a collection of items for which either the sum of the item areas exceeds the area of the container or the number of items is \( \geq k_{practical} \), find a “large” sub-collection of \( \leq k_{practical} \) items which fits in the container.

Whenever the total area of the items exceeds the area of the container, the containment problem is infeasible, and there is no need to run a containment algorithm. Whenever the number of items exceeds \( k_{practical} \), it is also not appropriate to run a containment algorithm. In both of these cases, our goal is to find a subset \( S \) of the given set of items which satisfies the conditions of the problem addressed in Section 9.3. The elimination technique of Section 9.3 can then be applied to \( S \).

One way to produce \( S \) is to use randomness. Using randomness has the advantage that we can create a diverse collection of subsets by repeating the process. We then apply the elimination technique to each subset. We select the resulting subset which has the largest area and therefore provides the tightest packing of the container. One can randomly exclude items from the collection until the subset satisfies the requirements of \( S \). Alternatively, one can select items randomly in a bottom-up fashion. When the number of items is large (i.e. \( \gg k_{practical} \)), as in our applications, the latter approach is more efficient.

We offer two examples of the success of this randomized approach. In both examples, some of the items can be reflected in the \( x \)-axis or rotated by a multiple of \( \pi/4 \). Our construction process takes these possibilities into account by randomly selecting orientations. Figure 9.7 portrays a collection of items and a container (filled in black). Figure 9.8 shows a collection of three items which pack the container tightly. We generated the following ten
Figure 9.7: Items and container for randomized construction task

Figure 9.8: Randomly constructed 3-item group

Figure 9.9: Randomly constructed 5-item group
groups for this container: (21 37 57) (63 39 56) (34 17) (33 46) (36 43) (34 44) (36 54) (33 65) (34) (32). Figure 9.8 gives another example. In this case, a collection of five items is placed in the container. We generated 150 groups for this container.

Pseudo-code for a group generation procedure for a container $C$ and a collection of items $\mathcal{P}$ is given below. We maintain a “fit” list $G$ and a “misfit” list $M$, which are both initially empty. To generate a group $g$ which fits, we first initialize the current group $g = \emptyset$. Then, while the total area of items in $g$ does not exceed the container area and the number of items in $g$ is $\leq k_{\text{practical}}$, we randomly add items to $g$. Let random($\mathcal{P}$) return a randomly selected item $P_i$, $i \neq 0$, from $\mathcal{P}$. Now we attempt to disqualify $g$ by checking if any group in the misfit list is a subset of $g$. If $g$ is not disqualified, we obtain a subset of $g$ which fits by invoking GENERATE-SUBSET from Section 9.3. We add the subset to $G$. Let $\alpha$ be a stopping condition for generating groups for a container.

\begin{verbatim}
GENERATE-GROUPS(\mathcal{P}, C)
    G \leftarrow \emptyset
    \text{while} \ \alpha \ \text{not satisfied}
        g \leftarrow \emptyset
        \text{while} \ \sum_{P_i \in g} \text{area}(P_i) \leq \text{area}(C) \ \text{and} \ |g| \leq k_{\text{practical}}
            g \leftarrow g \cup \text{random}(\mathcal{P})
            \text{if} \ \text{there exists} \ m \in M \ \text{such that} \ m \subseteq g
                M \leftarrow M \cup g
            \text{else}
                g \leftarrow \text{GENERATE-SUBSET}(g, C)
                \text{Add} \ g \ \text{to} \ G
        \end{verbatim}

return $G$

The question of what constitutes an effective stopping criterion $\alpha$ is an open one. A running time/packing density tradeoff clearly exists. Our experience suggests that a good balance between these two factors can be achieved if the number of groups is inversely proportional to the capacity of the container. We have tested our group generation procedure on hundreds of examples, and verified that our stopping criterion produces tight packings.

Ultimately, the best stopping condition can depend on what the groups will be used for. We therefore defer further discussion of this to the next section.

9.5 Pre-packing

Given a packing problem for a collection of containers and a collection of items, a pre-packing approach has two steps: 1) generate groups of items for each container, and 2) assign groups to containers. The goal is to place a “maximal”\(^3\) subset of the items. Section 9.4 offers a group generation procedure which uses containment. This solves problem (1). Problem (2) is a combinatorial optimization problem. There is a rich literature on combinatorial optimization.

The optimal solution to the assignment problem can be obtained using a matching model based on mixed integer programming. We developed one such model. A Boolean variable

\(^2\)We did not implement the misfit list, so it is not clear whether or not it is cost-effective in general.

\(^3\)Area is one possible measure.
in our model represents the assignment of group \( k \) to container \( j \). A Boolean variable \( p_{ij} \) represents the assignment of item \( i \) to container \( j \). The model has three types of constraints. The first type of constraint is of the form:

\[
\sum_k g_{kj} \leq 1
\]

There is one constraint of this type for each container. Each constraint prevents a container from having more than one group assigned to it. The second type of constraint is:

\[
\sum_j p_{ij} \leq 1
\]

There is one constraint of this type for each item. Each constraint prevents an item from being assigned to more than one group. The third type of constraint forces all items in a group to be assigned together. This is of the form:

\[
p_{ij} - g_{kj} \geq 0
\]

There is one constraint of this type for each item in each group. One possible objective function is to maximize the following:

\[
\sum_{k,j} g_{kj}
\]

We originally intended to use this model under the assumption that each group contains at most three items. If we allow up to ten items in a group, it is not clear that this approach is practical, given the current state-of-the-art in mixed integer programming. Consider, for example, a typical pre-packing application in apparel marker making. A typical pants marker has approximately 50 containers. If containment produces a 10-item group for each container, this yields \( 2^{10} \) subgroups for each container. The number of Boolean variables could be as large as 50,000 in this case.

At the other extreme, a greedy algorithm can be used for assignment. There are many possible approaches in between mixed integer programming and a greedy strategy. Finding the ultimate assignment technique for pre-packing is beyond the scope of this containment-oriented thesis. However, we offer an intermediate approach which runs quickly, provides look-ahead, and is quite flexible. Our results challenge the belief that pre-packing strategies are impractical [Arb93].

9.5.1 Assignment Strategy

Our assignment strategy requires three functions. The first is a comparison function \( f(g_1, g_2) \) which accepts two groups \( g_1 \) and \( g_2 \), returns \( g_1 \) if \( g_1 \geq g_2 \) and returns \( g_2 \) otherwise. Let \( C \) be a collection of containers. The second function is an ordering function \( f'(C_1, C_2) \) for containers. The third is an evaluation function \( h(C, g) \) which accepts a container \( C \in C \) and a group \( g \) and returns a real number \( \gamma \).

The first step of our assignment process is to sort the groups for each container according to \( f \). Then, while some item remains unassigned, perform the following. For each container, select the first group in its ordering for which there are sufficient unassigned items. Apply
\( h(C, g) \) to obtain a \( \gamma \) value for this group. Select the \((C, g)\) pair with the best \( \gamma \) value, and perform this assignment.

The evaluation function \( h \) uses a greedy look-ahead strategy which requires the ordering function \( f^i \) for containers. For a given container and its first group, tentatively perform this assignment. Now examine each remaining container according to the container ordering. For each container, tentatively assign the highest group (according to \( f \)) whose items are currently (tentatively) unassigned. When tentative assignments have been performed for all containers, return \( \gamma \), which is the total area of all tentatively assigned items.

In the pseudo-code below, \( C \) is a collection of containers and \( G \) is the output of \( \text{GENERATE-GR UP S} \) for all the containers. \( C_i \in C \) is an individual container, and \( G_i \in G \) is the collection of groups produced by \( \text{GENERATE-GR UOPS} \) for \( C_i \). An available group \( g \in G_i \) is one whose items are all available (i.e. unassigned).

\[
\text{ASSIGN}(\mathcal{P}, C, G)
\]

\[
\quad \text{for each} \quad \text{container} \quad C_i \in C
\]

\[
\quad \text{Sort} \quad G_i \in G \quad \text{according to} \quad f
\]

\[
\quad \text{while} \quad \text{some} \quad P_i \in \mathcal{P}, \quad i \neq 0, \quad \text{is available}
\]

\[
\quad \text{for each} \quad \text{container} \quad C_i \in C
\]

\[
\quad g \leftarrow \text{first available group of} \quad G_i
\]

\[
\quad \gamma_i \leftarrow h(C_i, g)
\]

\[
\quad \text{BestI} = \text{index of container with best} \quad \gamma
\]

\[
\quad \text{Assign items of} \quad g \quad \text{for} \quad C_{\text{BestI}} \quad \text{to} \quad C_{\text{BestI}}
\]

\[
\text{9.5.2 Results}
\]

To justify our claim that pre-packing can be practical, we present the results of our experiments with applying pre-packing to a trim placement problem for pants markers in apparel manufacturing (see Section 1.4 for a description of the trim placement problem). Table 9.1 presents results of our group generation and assignment procedures for 18 examples. For each example we give the name of the marker (layout), the number of trim items which must be placed, and results for three trim placement strategies. The three strategies are: 1) First-Fit, 2) our assignment strategy (called “Match” in the table), and our assignment strategy followed by First-Fit on the unplaced items. For each strategy, we give the number of items left unplaced and the total area of the unplaced items.

Our First-Fit strategy is a greedy heuristic which places each item at the leftmost available position. The items are sorted by decreasing area and are placed in that order. The heuristic has two additional features which allow it to produce layouts whose waste percentage is, on the average, only about 3% larger than what experienced humans achieve. The first additional feature tries all legal flips and discrete rotations (by multiples of \( \pi/4 \)) for each item and selects the best one. The second feature uses a variation on compaction to “bump” each placed item in a particular direction in an effort to reduce fragmentation of the available space.

There is a number of ways to interpret this data. Comparing the total number of items left unplaced, assignment plus First-Fit places 53% more items than First-Fit alone. Averaging the percentage of the items which are left unplaced shows that assignment plus First-Fit leaves only an average of 10% of the trim items unplaced, whereas First-Fit leaves 19% of
<table>
<thead>
<tr>
<th>marker name</th>
<th>ntrim</th>
<th>nleft</th>
<th>area left</th>
<th>nleft</th>
<th>area left</th>
<th>nleft</th>
<th>area left</th>
<th>nleft</th>
<th>area left</th>
</tr>
</thead>
<tbody>
<tr>
<td>37504c</td>
<td>70</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>4.3e+6</td>
<td>2</td>
<td>5.9e+5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41132b</td>
<td>56</td>
<td>15</td>
<td>2.3e+6</td>
<td>7</td>
<td>1.6e+6</td>
<td>2</td>
<td>7.7e+5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41132b1</td>
<td>56</td>
<td>16</td>
<td>3.1e+6</td>
<td>8</td>
<td>2.9e+6</td>
<td>5</td>
<td>2.4e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45941b</td>
<td>56</td>
<td>5</td>
<td>7.2e+5</td>
<td>12</td>
<td>2.8e+6</td>
<td>4</td>
<td>1.4e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42461d</td>
<td>56</td>
<td>12</td>
<td>1.9e+6</td>
<td>9</td>
<td>2.7e+6</td>
<td>7</td>
<td>2.4e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37188c</td>
<td>42</td>
<td>10</td>
<td>1.9e+6</td>
<td>5</td>
<td>1.5e+6</td>
<td>2</td>
<td>1.1e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2275k1t001</td>
<td>24</td>
<td>5</td>
<td>2.6e+6</td>
<td>3</td>
<td>2.3e+6</td>
<td>2</td>
<td>2.1e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2275k1t016</td>
<td>24</td>
<td>3</td>
<td>2.3e+6</td>
<td>4</td>
<td>2.5e+6</td>
<td>3</td>
<td>2.4e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2275k1t016a</td>
<td>24</td>
<td>7</td>
<td>3.0e+6</td>
<td>4</td>
<td>2.4</td>
<td>4</td>
<td>2.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2215k2t013</td>
<td>22</td>
<td>4</td>
<td>2.5e+6</td>
<td>5</td>
<td>2.8e+6</td>
<td>2</td>
<td>2.2e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37969a</td>
<td>18</td>
<td>2</td>
<td>3.0e+5</td>
<td>1</td>
<td>1.4e+5</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37969a1</td>
<td>18</td>
<td>2</td>
<td>1.1e+6</td>
<td>2</td>
<td>3.2e+5</td>
<td>2</td>
<td>3.2e+5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41487a</td>
<td>18</td>
<td>6</td>
<td>1.3e+6</td>
<td>2</td>
<td>7.3e+5</td>
<td>1</td>
<td>5.6e+5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41487a1</td>
<td>18</td>
<td>3</td>
<td>4.7e+5</td>
<td>1</td>
<td>5.6e+5</td>
<td>1</td>
<td>5.6e+5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24135t001</td>
<td>18</td>
<td>2</td>
<td>1.2e+6</td>
<td>2</td>
<td>1.2e+6</td>
<td>2</td>
<td>1.2e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24135t001a</td>
<td>18</td>
<td>2</td>
<td>1.2e+6</td>
<td>3</td>
<td>1.5e+6</td>
<td>3</td>
<td>1.5e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24135t122</td>
<td>18</td>
<td>3</td>
<td>1.4e+6</td>
<td>3</td>
<td>1.5e+6</td>
<td>3</td>
<td>1.5e+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>340065233</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>7.2e+5</td>
<td>1</td>
<td>7.2e+5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9.1: Pre-packing results
them unplaced; assignment plus First-Fit is 9% better by this measure. It is also valuable to compare the area of the unplaced items. A raw comparison of total area shows that assignment plus First-Fit placed 12% more area than First-Fit. To remove any differences in scale, averaging the percentage of the area which is left unplaced shows that assignment plus First-Fit leaves only an average of 23% of the area unplaced, whereas First-Fit leaves 26% of the area unplaced; assignment plus First-Fit is 3% better by this measure. All of these comparisons show the quality of assignment plus First-Fit to be superior to First-Fit alone.

We illustrate our results by showing layouts for two examples. For the first example, Figure 9.10 shows the trim placement task, Figure 9.11 shows the layout obtained by First-Fit, Figure 9.12 shows the layout obtained by our assignment procedure, and Figure 9.13 shows the layout after assignment followed by First-Fit. Figure 9.14, Figure 9.15, Figure 9.16, and Figure 9.17 show the same progression for our second example.

Additional quality comes at the cost of higher running time. Our First-Fit strategy requires under five minutes, whereas our new strategy currently requires one to two hours for a 56 item example (on a 50 MHz SPARCstation). In apparel manufacturing, a human takes about 1/2 hour for an example of this size. One to two hours is within the realm of practicality. For our current implementations of containment and assignment, about 95% of the running time is consumed by group generation, and of this 95%, almost all the time is spent in the containment algorithm. This means that a dramatic increase in the speed of our hybrid containment algorithm will also have a dramatic impact on the speed.
Name: 37188c-fit
Width: 59.75 in
Length: 130.28 in
Pieces: 54
Efficiency: 86.78%

Figure 9.11: First-Fit for 37188c.mrk

Name: 37188c-matchk
Width: 59.75 in
Length: 130.28 in
Pieces: 54
Efficiency: 87.27%

Figure 9.12: Assignment for 37188c.mrk
Figure 9.13: Assignment plus First-Fit for 37188c.mrk

Name: 37188c-matchk2
Width: 59.75 in
Length: 130.28 in
Pieces: 54
Efficiency: 87.82%

Figure 9.14: Trim placement task for 41132b.mrk

Name: 41132-trim
Width: 59.75 in
Length: 182.13 in
Pieces: 72
Efficiency: 78.65%
Name: 41132-fit
Width: 59.75 in
Length: 182.13 in
Pieces: 72
Efficiency: 89.04%

Figure 9.15: First-Fit for 41132b.mrk

Name: 41132-matchck
Width: 59.75 in
Length: 182.13 in
Pieces: 72
Efficiency: 89.67%

Figure 9.16: Assignment for 41132b.mrk
Our best trim placement strategy for pants markers in apparel manufacturing is neither First-Fit nor pre-packing. It places 56 items in approximately ten minutes and its waste is in the production quality range. Neither First-Fit nor our current pre-packing implementation currently generate results which are consistently in the production quality range. Our best strategy employs special heuristics to place the largest items first, uses containment to construct small groups of at most three items, a naive greedy assignment strategy, and compaction-based techniques combined with First-Fit to place any remaining items.

Although it achieves excellent results for pants markers, its applicability to other domains is limited. In contrast, our pre-packing approach is completely general. It can be applied to any domain. The same is true of First-Fit.

Our best pants strategy also does not have much growth potential. That is, we do not expect that adjustments to parameters of this algorithm will significantly reduce the waste of the associated layouts. The same is true of First-Fit. However, our pre-packing approach has tremendous growth potential. Its initial results are good, and improvements in the assignment strategy or the group generation process will improve the quality of its results.

Future research will benefit from determining whether the current group generation process or the assignment strategy is the limiting factor. We have attempted to perform assignment manually using the groups we generate, and our initial results show that, surprisingly, the assignment strategy is less limiting than group generation.

There are many parts of the group generation procedure which can be altered: the random procedure for building a group, the elimination technique, the containment algorithm, and the number of groups. Our initial experiments have imposed a number of artificial limitations. Our current implementation limits the size of a subset to ten; our waste figures would probably improve with a higher limit. We also limit the number of hypotheses evaluated by the hybrid algorithm to less than 20, and we build no more than 150 groups for a container.

This returns us to the stopping condition issue raised in Section 9.4. It is impractical to generate all possible groups for each container. How do we determine if a given collection of
groups is satisfactory for the assignment phase without running the assignment algorithm? This is beyond the scope of the thesis. Investigating ways to obtain a lower bound on waste for a given collection of groups is a subject of future work. Given a lower bounding technique, one could iteratively determine a bound and selectively generate more groups until the desired waste level was achieved.

9.6 Summary and Conclusions

This chapter has addressed the following maximal item problems: 1) given a container and a collection of items, finding maximally scaled copies of items which fit in the container, 2) given a container and a collection of \( \leq k_{\text{practical}} \) items for which the sum of the item areas does not exceed the area of the container, finding a "large" sub-collection which fits in the container, 3) given a container and a collection of items for which either the sum of the item areas exceeds the area of the container or the number of items is \( > k_{\text{practical}} \), finding a "large" sub-collection of \( \leq k_{\text{practical}} \) items which fits in the container, and 4) problem (3) for a collection of containers.

For problem (1), we offer a heuristic which uses binary search combined with containment. The heuristic has most of the advantages of the greedy and the enhanced balanced strategies. It beats both of these approaches on some inputs.

The remaining three problems are all related to our pre-packing strategy. We use an elimination strategy for problem (2), which uses feedback from the hybrid containment algorithm to identify "troublesome" items. For problem (3), we randomly construct a subset of items and then solve problem (2). This process considers multiple (discrete) orientations for items. For problem (4), we find groups of items which can fit into each container, and then apply a look-ahead assignment strategy. We demonstrate what we believe is the first practical application of pre-packing. We are able to generate a sufficiently rich collection of groups for each container so that our assignment strategy, followed by a First-Fit strategy, can outperform a First-Fit strategy. This process takes one or two hours for a 56-item example, using the current implementation of our hybrid containment algorithm. There are many ways to improve our group generation and assignment processes. We believe that, in addition to demonstrating the feasibility of pre-packing, this approach has the potential to eventually match the production-quality results of our best algorithm for trim placement in apparel manufacturing.
Part III

Conclusion and Appendix
10.1 Conclusion

This thesis demonstrates that, although containment is NP-hard, it is fruitful to: 1) develop algorithms for containment, as opposed to heuristics, 2) design containment algorithms so that they say “no” almost as fast as they say “yes”, 3) use geometric techniques, not just mathematical programming techniques, and 4) maximize the number of items for which the algorithms are practical.

Our approach to containment has a number of important features. First, we introduce a restrict/evaluate/subdivide paradigm. Second, we develop new theory and practical techniques for the operations within the paradigm. The techniques are appropriate for two-dimensional containment problems in which the items and container may be irregular (i.e. nonconvex) and have multiple components, and in which the items may be translated, but not rotated. Our techniques can be combined to form a variety of two-dimensional translational containment algorithms which follow the restrict/evaluate/subdivide paradigm. The paradigm is designed so that, unlike existing iteration-based algorithms, containment algorithms based on the paradigm are adept at saying “no”, even for slightly infeasible problems. Infeasible problems occur frequently in practice.

We develop geometric restrictions which are powerful configuration space pruning techniques. They often detect infeasibility without any evaluation or subdivision. We give two algorithms which use geometric restrictions and are based on the restrict/evaluate/subdivide paradigm. We obtain the first practical running times for NP-complete two-dimensional translational containment problems for up to ten nonconvex items in a nonconvex container. Our examples are drawn primarily from the apparel industry. Each item is represented as a polygon which typically has from 4 to 100 vertices. The container is also represented as a polygon; it typically has from 100 to 300 vertices. The items and container may be nonconvex.

We demonstrate that viewing containment as a feasibility problem has many benefits for layout optimization problems. We present an effective method for finding minimal enclosures which uses containment to perform binary search on a parameter. We note that compaction can be used to accelerate the search. This work represents the first practical approach to finding minimal enclosures for multiple nonconvex items. Clustering is used.
often in layout, but current clustering techniques for more than two items are heuristic. Our minimal enclosure work should lead to more powerful clustering methods, and is an example of how strong “bottom-up” techniques from computational geometry can help the layout community.

We believe that some potentially promising layout techniques have been abandoned by the layout community because they lack strong bottom-up methods. The primary example of this is pre-packing. We challenge the view that pre-packing is impractical for multiple-container packing problems by generating a rich collection of groups for each container, using containment, and then applying an assignment strategy.

Our advances in containment will help automate layout for industrial processes that rely on two-dimensional packing of objects. Our results are particularly relevant to industries which deal with irregularly shaped items. This includes industries such as aerospace, ship building, apparel and shoe manufacturing, furniture production, and steel construction.

10.2 Future Work

Section 1.5 invites other researchers to exchange containment data with us. We hope this will result in a broad set of containment benchmarks which can be used to evaluate the effectiveness of different containment algorithms. In addition to soliciting containment problems from a variety of sources, we plan to automate the process of creating containment problems from existing layouts.

Here we offer directions for future research on containment problems and applications of containment to layout. Section 10.2.1 describes problems related to containment algorithms, and Section 10.2.2 discusses applications.

10.2.1 Containment

There are several directions for future work in containment. One direction is to increase the dimensionality of the containment problems we address. For instance, many important three-dimensional packing and layout problems could be solved using three-dimensional containment algorithms. Another way to increase the dimensionality of the containment problem is to allow rotation. Rotational containment adds an extra degree of freedom to each item by allowing it to rotate continuously within a range of angles. Many layout problems allow continuous rotation of items, particularly problems related to sheet metal work. Section 1.2.1 reviewed work in this area. There is clearly room for much work on rotational containment involving more than one item.

There are ways rotation could be dealt with in a heuristic fashion. The first way involves our minimal enclosure work. Recall that compaction can be used to update the upper bound. Li and Milenkovic have rotational as well as translational compaction algorithms. Rotational compaction could be applied to the translational solution as a final, post-processing step. In our hybrid translational containment algorithm, one could develop a rotational overlap reduction evaluator. Using this evaluator, the hybrid algorithm would try to produce a valid rotational configuration from each overlapping translational configuration found by the linear program. A second direction for future work involves improving our algorithms for translational containment so that larger numbers of items (more than ten) may be placed in a practical amount of time. One way to obtain a speed-up is to apply course-grained parallelism.
Throughout the thesis we have identified parts of our work which are natural candidates for course-grained parallelism.

Some layout applications represent objects using circular arcs in addition to polygons, and so enriching our underlying object representation is a worthwhile goal.

There is a considerable literature on rectangle packing and integer programming that remains untapped for general kNN containment. Integer programming requires a great deal of finesse to apply correctly, and we have been disappointed so far with results we have obtained using state-of-the-art mixed integer programming packages. Our work on combining computational geometry and linear programming is a large step towards the ultimate goal of usefully combining computational geometry and integer programming.

Exact covering problems, such as solving jigsaw puzzles, can be solved using containment algorithms. For the non-exact case, we hope to develop algorithms which decide if a collection of items can cover another item.

We plan to develop more valid and semi-valid restrictions for translational containment. We will continue to explore the question of which restriction operations allow s, the maximum number of vertices in a restricted polygon, to remain finite even when an infinite number of operations is applied.

The following problems are associated with our greedy evaluation method:

- Given polygons $A$ and $B$, is there a translation $t$ for which $H((A + t) \cup B)$ is convex?
- Find $t$ which maximizes the convexity of $(A + t) \cap B$.
- Solve these problems for convex $A$ and $B$.
- Given a collection of polygons $\mathcal{P}$, how fast can we find the translation $t_p$ for each $p \in \mathcal{P}$ such that the area of $\cap_{p \in \mathcal{P}}(p + t_p)$ is maximized? This is also useful for finding the maximum area breaking polygon.

The shrinking-point heuristic which maximizes the area of intersection of translated $U_{ij}$s works well unless the $U_{ij}$s consist only of isolated edges and points. This case can occur when we form subproblems using a boundary restriction. In this case, we propose using a distance-based heuristic instead of an area-based one.

These problems arise in relation to subdivision:

- Subdivide in a that encourages translations of $U_{ij}$s to intersect.
- Given a set of polygons $\mathcal{P}$, how fast can we find the line $L$ which partitions $\mathcal{P}$ into $\mathcal{P}'$ and $\mathcal{P}''$, such that the area of $(H(\mathcal{P}') \cup H(\mathcal{P}'')) \cap H(\mathcal{P})$ is minimized?

Potential improvements to the approximate algorithm include:

- Use multiple resolution techniques.
- Invoke a linear programming-based algorithm when the convex hulls of the restricted $U_{ij}$s are all inside their respective unrestricted $U_{ij}$s.
- Extend the size-based subdivision used in this algorithm so it subdivides $U_{ij}$s for which $i \neq 0$. 

182
We plan to evaluate the cost-effectiveness of using a partial characterization to produce \( k \) subproblems for \( k > 4 \). Our current experiments suggest that it is beneficial for \( k = 4 \), even when the subproblems are solved sequentially. We expect that, for non-identical item problems, a critical value of \( k \) exists beyond which the sequential advantage is lost. A detailed comparison of the parallel cost will be necessary in order to justify using the partial characterization for values of \( k \) larger than the critical one. Course-grained parallelism can be applied to our algorithm by maintaining a queue of hypotheses. Each hypothesis represents an independent containment subproblem which is created via subdivision. Whenever a processor is available, it operates on the next hypothesis in the queue. The subproblems generated by the characterization are also independent from each other, and so the hypotheses they generate are independent and can be added to the queue. We plan to construct a program which uses a queue and simulates the parallel execution of our algorithm.

10.2.2 Applications of Containment to Layout

In our work on container transformation, we currently approximate the area integral of \( \text{Prob}(\{t\}, v') \) for \( t \in v \) via sampling. We seek efficient algorithms for computing this integral exactly.

Pursuing more efficient algorithms for finding the exact MAAPR is certainly one direction for future work. We are interested in faster algorithms or lower bounds for these polygon types:

- Orthogonally convex
- Horizontally convex
- Vertically separated, horizontally convex

Another direction of practical importance is to find a fast approximate MAAPR algorithm. Such an algorithm would be very helpful in our applications.

There are many open problems associated with the concept of inking sets. The problem of minimizing the number of vertices while maximizing the minimum area of inking set pieces is open, even for convex polygons. Given a collection of polygons, the problem of finding a polygon representing the maximal common subset of translations of these polygons is useful not only for inking sets, but also for the subset substitution restriction.

Future work on our minimal enclosure applications includes implementing our idea of using compaction to accelerate binary search. We also plan to experiment with using our minimal enclosure algorithms for finding repeating patterns of irregular objects.

Future work on maximal item problems includes improving our algorithm for solving the maximal homothet problem. Our pre-packing work can be improved by obtaining lower bounds for assignment and feeding these results back into the group generation process.
This appendix describes the simple file format we propose for two-dimensional polygonal containment data. The format first describes the polygonal container and then each of the polygonal items to be placed. For each polygon, the number of vertices precedes the list of $x$ and $y$ vertex coordinates. Data for each vertex appears on a separate line. The vertices are listed in counterclockwise order, so that the inside of the polygon is to the left of each directed edge.

The small example on the next page is for a 3CC problem for which our approximate algorithm finds a valid configuration in one second on a 28 MHz SPARCStation. In this example, the container is a triangle, and each of the three items is also a triangle.
container vertices: 3
  0.0 30.0
  200.0 30.0
  100.0 200.0

items: 3
item vertices: 3
  0.0 0.0
  0.0 100.0
  -80.0 -60.0
item vertices: 3
  0.0 0.0
  80.0 -60.0
  0.0 100.0
item vertices: 3
  0.0 0.0
  -70.0 -60.0
  70.0 -60.0
Bibliography


193