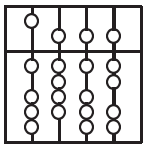
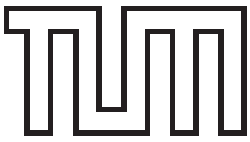


Institut für Informatik
der Technischen Universität München



Geometric Reasoning About Translational Motions

Dissertation

Fabian Schwarzer

Institut für Informatik
der Technischen Universität München

Geometric Reasoning About Translational Motions

Fabian Schwarzer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Bernd Radig

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Achim Schweikard
2. Senior Lecturer Dr. Leo Joskowicz,
   Hebrew University of Jerusalem, Israel

Die Dissertation wurde am 10.7.2000 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11.9.2000 angenommen.

Abstract

This thesis addresses the problems of planning highly coordinated collision-free translational motions of multiple objects (general polygons or polyhedra) and finding sequences of translations that allow for the removal of one or more objects. These problems are known to be PSPACE-hard in general. Our main focus is therefore on the design and analysis of adaptive algorithms that can cope with practically relevant cases such as tightly interlaced placements. Exact and complete algorithms are important, for example in the field of computer aided mechanical assembly planning: a valid plan for separating a given placement of rigid non deformable objects (parts) can be reversed to assemble the individual components. Our algorithms are exact and complete and can prove in practical cases that no separating motions exist.

We present our discussion within the uniform framework of the composite configuration space of multiple translating objects. The general dimension of this space, which is proportional to the number of parts, causes certain combinatorial problems that are treated in detail. First, we consider single simultaneous ('one-shot') translations. For the special case of a single moving subset of parts, a polynomial time algorithm was known before this work. However, it was not clear if the underlying principles could be generalized to obtain a polynomial time algorithm for more than a single moving subset. We show that the problem becomes NP-complete when two or more different directions of simultaneous translation are allowed. We analyze in detail under which conditions translational one-shot separability is NP-hard, and derive a new polynomial-time algorithm for separating placements of polygons in which no pair of parts is separated. For general instances of one-shot separability, we propose heuristic reductions on a high level of abstraction.

Translational one-shot separability is closely related to high-dimensional linear unboundedness testing. We present a new efficient algorithm for linear unboundedness testing that consists of solving a single homogeneous system of equations followed by a single linear feasibility test. It is shown that our algorithm is optimal if optimal algorithms are used for these two main steps.

Efficient unboundedness testing is also one of the keys to practical algorithms for general translational separability. In the context of the latter problem, arbitrary sequences of translations may be required to remove one or more parts. We propose opportunistic incremental algorithms that take advantage of the fact that for many practical instances, the set of feasible translations is quite constrained. The salient features of our algorithms are exact implicit convex cell decomposition of the composite configuration space, complete incremental expansion of a single connected component, and elimination

of redundancies by cell defragmentation. We show that additional effective heuristic improvements can be formulated on a high level of abstraction, which makes them relevant for a whole variety of problem instances.

The presented algorithms have been implemented and tested on several planar and spatial examples of differing complexity. Our experimental results confirm the practicality of the approach. Observed running times are dominated by the discrete number of reachable critical placements which are usually very limited in assembly planning applications. The algorithms are exact and complete. For several examples with many degrees of freedom, it was possible to compute a complete proof that no separating translations exist. Extensions of practical importance can be integrated directly into our implementation. These include linear tolerance models, parameters that permit small bounded overlap, and methods to identify features of parts that would have to be slightly modified to allow for overlap-free separation of otherwise non separable placements.

Zusammenfassung

Die vorliegende Arbeit behandelt das Problem der Planung von koordinierten, translatorischen Bewegungen mehrerer Objekte (allgemeine Polygone oder Polyeder) und das Problem, Translationsfolgen zu finden, um ein oder mehrere Objekte von den übrigen zu entfernen. Es ist bekannt, dass diese Probleme im allgemeinen PSPACE-hart sind. Wir konzentrieren uns daher auf Design und Analyse adaptiver Algorithmen, die in praktisch relevanten Fällen – wie etwa verschachtelte Anordnungen von Teilen – erfolgreich eingesetzt werden können. Exakte und vollständige Algorithmen sind zum Beispiel in der Montageplanung von besonderer Bedeutung: eine Bewegungsfolge, die eine gegebene Anordnung von Teilen zerlegt, kann umgekehrt ausgeführt werden, um die Zielanordnung zu montieren. Die in dieser Arbeit beschriebenen Algorithmen sind exakt und vollständig und können in praktischen Beispielen beweisen, dass keine Trennung durch Translation möglich ist.

Dem Ansatz liegt als allgemeines Konzept der zusammengesetzte Konfigurationsraum (engl. 'composite configuration space') für Translationen mehrerer Objekte zugrunde. Die allgemeine Dimension dieses Raums ist proportional zur Anzahl der Teile und verursacht daher gewisse kombinatorische Probleme. Diese werden im Detail behandelt. Zunächst wird Zerlegbarkeit durch eine einzelne, unbegrenzte simultane Translation (engl. 'one-shot translation') untersucht. Für den Spezialfall, dass sich alle bewegenden Teile nur mit einer einheitlichen Geschwindigkeit bewegen können, war bereits ein polynomieller Algorithmus bekannt. Es war allerdings nicht klar, ob die zugrundeliegenden Methoden verallgemeinert werden können, um einen polynomiellen Algorithmus zu erhalten, der verschiedene Geschwindigkeiten für verschiedene Teilmengen erlaubt. Hier wird gezeigt, dass die Problemstellung bereits dann NP-vollständig wird, wenn man zwei (oder mehr) Richtungen für gekoppelte Translation zulässt. Wir untersuchen im Detail, unter welchen Umständen das Problem der Zerlegbarkeit durch eine einzelne Translation NP-hart wird, und leiten einen polynomiellen Algorithmus für den Fall her, dass in der zu zerlegenden Anfangsstellung noch kein Polygonpaar getrennt ist. Für den allgemeinen Fall der Zerlegung mit einer einzelnen Translation werden Heuristiken auf hohem Abstraktionsniveau vorgeschlagen.

Zerlegbarkeit durch eine einzelne, gleichzeitige Translation steht in enger Verwandtschaft mit der Fragestellung, ob der Lösungsraum eines Systems hochdimensionaler linearer Ungleichungen unbegrenzt ist. Dafür wird ein neuer effizienter Algorithmus vorgestellt. Dieser löst das Problem in zwei Teilschritten: zunächst wird ein aus dem Ungleichungssystem unmittelbar hervorgehendes homogenes Gleichungssystem gelöst, worauf gegebenenfalls noch ein linearer Erfüllbarkeitstest auszuführen ist. Es

wird gezeigt, dass der neue Algorithms optimal ist, wenn man für die beiden Teilprobleme optimale Algorithmen einsetzt.

Der effiziente Unbegrenztheitstest ist eine der Grundvoraussetzungen für praktikable Algorithmen zur Lösung allgemeiner translatorischer Trennbarkeitsprobleme. In diesem Zusammenhang lautet die Fragestellung, ob man eine Anordnung von Teilen mit beliebigen Translationsfolgen zerlegen kann. Es werden neue opportunistische Algorithmen vorgestellt, deren Erfolg darauf basiert, dass in vielen praktischen Fällen der tatsächlich verfügbare Bewegungsfreiraum stark eingeschränkt ist. Die Hauptmerkmale der Algorithmen sind implizite Zerlegung des hochdimensionalen Konfigurationsraums in konvexe Zellen, vollständige Berechnung von einer einzelnen erreichbaren Zusammenhangskomponente, und Elimination von Redundanzen durch Zellerweiterung. Es wird gezeigt, dass zusätzliche Heuristiken auf hohem Abstraktionsniveau für ganze Problemklassen anwendbar sind.

Die vorgestellten Algorithmen wurden implementiert und anhand einiger zwei- und dreidimensionaler Beispiele verschiedener Schwierigkeit getestet. Die experimentellen Ergebnisse bestätigen die Praktikabilität des Ansatzes. Die beobachteten Laufzeiten hängen hauptsächlich von der Anzahl der erreichbaren kritischen Stellungen ab, welche in Montageplanungsanwendungen gewöhnlich stark eingeschränkt sind. Die Algorithmen sind exakt und vollständig. In einigen Beispielen mit vielen Freiheitsgraden war es möglich, vollständig zu beweisen, dass eine Trennung durch Translationen nicht möglich ist. Praktisch relevante Erweiterungen können direkt in die bestehende Implementierung eingebaut werden. Diese umfassen zum Beispiel lineare Toleranzmodelle oder Parameter, um kleine Überschneidungen zu erlauben. Daneben kann eine erweiterte Implementierung kleine Designänderungen ausfindig machen, die erforderlich sind, um ansonsten nicht zerlegbare Anordnungen auseinander nehmen zu können.

Acknowledgements

First of all, I would like to thank my advisor Prof. Achim Schweikard for giving me the possibility to write this thesis and for being encouraging and patient. Without his ideas and our discussions, this work would not have been possible.

My sincere thanks to Prof. Leo Joskowicz from the Hebrew University, Jerusalem, for our ongoing pleasant and successful collaboration, most of the time via e-mail. His tireless input, comments and suggestions were a great motivation and definitely improved many aspects of this thesis. I highly appreciate his invitations for my two official stays in Israel. He was a great host, making these sojourns delightful memories for me.

Thanks to Prof. Bernd Radig, who holds the chair *Informatics IX* at Technische Universität München, for being chairman of the thesis committee, and for employing me during the time I was doing research and writing this thesis.

Friends, colleagues and former colleagues made my stay at the chair *Informatics IX* worthwhile. The *Round Table*, as we would call the weekly meeting of all associates, and the daily lunch with some of my colleagues were important places for technical and social communication. Thanks to Matthias Hilbig, Ernst Bartels, and Adam Ferrier for proof-reading parts of the text. Thanks also to Sebastian Buck for discussions.

Thanks to my brother for being different but straightforward and, last but not least, deep thanks to my parents for their support during both easy and difficult times of my life. They have really shown me what it means to be reliable and honest.

Contents

x

List of Figures

List of Tables

Chapter 1

Introduction

Research in the field of artificial intelligence has traditionally been concerned with designing and building systems that are capable of interacting with their environment in an 'intelligent' way. Over the course of time however, it has become increasingly clear that 'general human intelligence' is hard to capture in a computer. This problem is directly related to the difficulty of giving a precise formal definition for the term 'intelligence'.

As a consequence, most of the recent research has focussed on 'intelligent' algorithms and systems for special tasks and in limited domains. Examples include game playing, vision, perception of speech, computer aided design and layout, and robotics. The goal has been to outperform the human in these and other tasks, and this has been partly achieved up to now. One of the most prominent examples is a chess machine known as *Deep Blue* that beat the chess world champion Kasparov in 1997. A second, less renowned example is that of marker making in the apparel industry. Recent commercial programs, using proper heuristics, are capable of finding layouts that consume slightly less fabric than those found by human experts. Note that optimal layout is an NP-hard problem. (That is, according to the current standards of knowledge in mathematics and computer science, it is very unlikely that all instances of the problem can be solved within deterministic polynomial time.)

Despite the growing success and spectrum of specific applications, new basic techniques are becoming increasingly important. One relevant aspect for intelligent applications is reasoning about the motions and interactions of geometric objects. Typical problems consist of planning collision-free motions or finding overlap-free placements of objects. Human capabilities in this domain span a wide range of tasks and it seems difficult to formalize this variety with a uniform concept.

A remarkable observation is that humans are not only capable of finding a solution to a given motion planning problem rapidly, but also of quickly recognizing that no feasible solution exists. It would seem that recognizing infeasibility is more difficult than just finding proper motions. Establishing that a given problem is infeasible requires a proof. For an example, consider the simple lock/bolt mechanism in Fig. 1.1. The goal is to decide whether or not the bolt is removable. To explain why the bolt cannot be removed, one would examine a series of critical intermediate configurations of the mechanism, and test for removability of the bolt in each of them. The human is able to recognize that relatively few *critical*

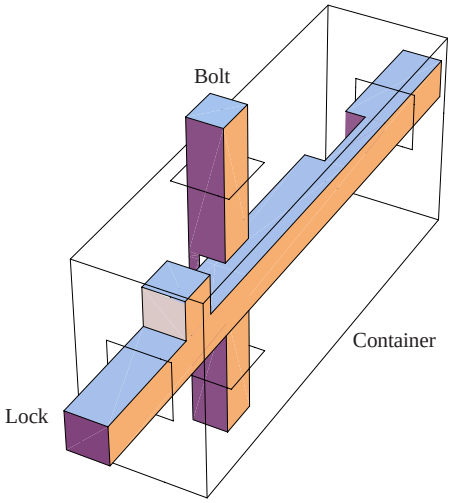


Figure 1.1: To determine whether or not the bolt can be removed, reachable placements of the lock must be examined (example from [64]).

intermediate positions of the lock need to be tested to prove that the bolt is not removable. However, the main problem with reproducing such an intuitive approach in an algorithm is the decision about which relative placements are indeed the critical ones. A naive complete enumeration of all distinct contact placements of parts quickly becomes rather costly at best and impossible even for slightly more difficult cases than the above example. We are thus interested in methods for reducing the set of critical relative placements of parts, *while retaining completeness*. To obtain a practical method, it is necessary to recognize the essential contacts between parts. Conversely, however, it should be noted that our detection of infeasibility would be in error if the set of examined contacts was too small.

In this work, we consider the problem of *exact reasoning about rigid virtual objects*. That is, we assume that an exact and complete representation of the objects and their positions is available. We will base our discussion on linear boundary descriptions (polygons and polyhedra) because they are commonly used and conceptually simple. We abstract away physical aspects such as mass and inertia and assume that we can accurately handle all objects without introducing positional uncertainties. It will become clear that such assumptions are acceptable for applications in geometric assembly planning which will be the main target of this work. The restriction to translational motions is the key to our main contribution: exact and complete incremental computation of high-dimensional configuration spaces.

In the context of virtual objects, geometric reasoning may be located within the discipline of computational geometry. O'Rourke [49] defines the latter broadly as 'the study of algorithms for solving geometric problems on a computer'. From the representation and complexity point of view, important aspects of this field are strongly connected to the mathematical discipline of discrete and combinatorial geometry [13].

1.1 Applications and Complexity of Geometric Reasoning

In the following, we give a sample of two areas of practical application in which geometric reasoning plays an important role. In particular, we consider robot motion planning and assembly planning/separability, and summarize important theoretic results.

1.1.1 Robot Motion Planning

The automated planning of collision-free motions for a robot is an important step towards task-level programming. Current industrial practice still consists of tedious teach-in which means that every single step of a robot task has to be planned and programmed by a human. One important goal is therefore to make robots smart enough to plan their motions themselves. Much of the research on robot motion planning has focussed on planning in a virtual environment. A common assumption is that a complete geometric model of the robot and its environment is available.

> **Robot motion planning problem (following [39]).** *Given a robot R at some initial position p_s and a finite collection of static impenetrable obstacles $\mathcal{O}$. The task is to plan motions that will take R from p_s to some goal position p_g without colliding with parts of itself or any of the obstacles. It is assumed that the geometry and location of the obstacles are completely known.*

Since this definition is quite general, it comprises apparently different tasks such as path planning for autonomous robots, manipulator arm planning and planning for general kinematic chains. However, one can use an abstraction that maps any robot to a point in a high-dimensional space, the *configuration space* (or parameter space) of the robot [39]. The dimension of this space equals the number of degrees of freedom of the particular robot. Both the obstacles $\mathcal{O}$ in the robot's environment and self-intersecting configurations of the robot map to certain abstract obstacles (configuration obstacles) in the configuration space. (Note that some robots, such as manipulator arms, are in fact in danger of self-intersections.) Theoretically, all conceivable variants of the above motion planning problem reduce to finding a continuous curve in a configuration space that avoids the (static) configuration obstacles.

Reif [56] first showed that in general, 'planning a free path for a robot made of an arbitrary number of polyhedral bodies connected together at some joint vertices, among a finite set of polyhedral obstacles, between any two given configurations, is a PSPACE-hard problem'. Schwartz and Sharir [60] were the first to give an exact and complete algorithm that computes an exact and complete cell decomposition of any general configuration space and can thus theoretically solve any robot motion planning problem. However, the complexity of their algorithm is twice exponential with respect to the dimension. Later, Canny [3] improved this result with an algorithm that computes a complete roadmap of the configuration space. The time complexity of his algorithm is singly-exponential with respect to the dimension. Both approaches reduce robot motion planning to a decision problem on the first order theory of the reals.

They scale extremely badly to higher dimensions. Thus, neither of them has ever been implemented for dimensions that exceed some very small constant.

Results from robot motion planning, especially the notion of configuration space, are important in other geometric reasoning and planning domains as well. For a broad survey on robot motion planning until 1991, including a comprehensive introduction to the configuration space concept, cf. Latombe's book [39].

Next, we introduce the assembly planning problem and its relation to robot motion planning which constitutes the main motivation for this thesis.

1.1.2 Geometric Assembly Planning / Separability

Modern product design is usually supported by a computer (computer aided design, CAD). That is, geometrical models are constructed to evaluate the appearance, physical properties and other aspects of the final outcome at an early design stage by means of virtual prototypes. Interweaving design and assembly planning can be extremely helpful to spot misfits, especially when different components are concurrently constructed by different manufacturers. The most fundamental geometric assembly planning problem is to determine if the individually designed CAD models can actually be assembled to the desired product.

Since the final placement of the parts is usually more constrained, it is often simpler to compute a disassembly motion, that is, a motion for separating or removing parts from a given assembly. Under the assumption of rigid (non-deformable) parts, a disassembly plan can be simply executed in reverse order to obtain a valid assembly plan. Most geometric assembly planning approaches use this 'assembly-by-disassembly' approach.

> **Geometric assembly planning (separability) problem. (following [73])** *Given an initial overlap-free placement of non-deformable rigid parts $P_1, \ldots, P_k$, compute collision-free motions that separate the parts. (The parts are considered to be separated if each of them can be translated arbitrarily far from the remaining parts without causing intersections.) If the parts are not separable, give a proof for infeasibility.*

Wolter [73] showed that assembly planning is PSPACE-hard even when only translational motions but no rotations are allowed. As a consequence, much research on geometric assembly planning has focussed on various restricted versions of the problem. However, even several restricted subproblems turned out to be NP-hard (cf. Section 1.2 and our new results in Chapter 2).

Assembly planning can be reduced to robot motion planning. The obvious formal difference between the two problems is that in the case of robot motion planning, both an initial and a goal configuration are given. In the assembly planning problem, we have an initial configuration, but the goal configuration is not specified explicitly (the parts just have to be separate from each other). Wolter [73] showed that by placing the parts in the points of an n-pointed star, one can always create a goal placement in which every part is separate from all other parts. Another, more important difference is that assembly planning

usually involves very small clearances between parts. In fact, most clearances will be zero because parts usually have to fit tightly in their final placement. This observation substantiates that exact algorithms are especially important here. In contrast to that, clearances in robot motion planning applications must be sufficiently large to keep safety distances from obstacles.

Note our distinction between *separability* (*separable* by motions) on the one hand, and *separation* (*separated* by a straight line or plane). In computational and combinatorial geometry, separability is sometimes used for our notion of separation. We use the notion of (movable) separability from [69, 49]. (Other authors use the term 'partitioning' instead of separability.)

1.2 Related Work

Robot motion planning. Owing to the complexity of the general robot motion planning problem, exact algorithms that are both complete and practical have been limited to low-dimensional configuration spaces [52, 48]. However, many approximate, heuristic and randomized algorithms for higher dimensions have been proposed over the years. The recent trend in robotics goes towards heuristic methods to solve problems in quite impressive realistic scenarios (cf. [21] for a survey). However, these methods are still not complete and cannot prove infeasibility of planning tasks.

Containment, layout and compaction. Basic concepts from robot motion planning have been applied in other domains, such as automated layout. For example, in the apparel industry, the problem of automated marker making is to place a set of polygons (patterns) within a limited area (of cloth) so that the total waste between the shapes is minimized. Several problems in this context have been shown to be NP-hard. For an overview of containment and layout problems, cf. [10]. Another layout application, compaction, is closely related to motion planning for multiple moving objects. In this context, Li [40] derives linear constraints for pairs of polygons and applies linear optimization to 'squeeze out the extra free spaces in an existing layout to produce a shorter layout and hence a tighter packing of the polygons'. Li applies some of the basic techniques that are also used in our work, but his work aims at local compaction and is therefore not complete.

Mechanism analysis. Joskowicz and Sacks [29, 58] use configuration space composition to analyze the operation of mechanical mechanisms. They consider devices that are composed of linkages (parts linked by permanent joints) and fixed-axes mechanisms (sets of parts that move along fixed spatial axes). Mechanism analysis ensures the desired functional behavior of the device as a whole. Some of the basic principles (such as composition of pairwise configuration spaces) are similar to ours. However, other aspects that are important for applications in the assembly planning domain are not relevant for mechanism analysis. These include the dynamic selection of pairwise constraints and unboundedness testing (the latter being one of the keys for identifying removability). In [29], the required pairwise constraints are identified only once, at the beginning, for example by intersecting envelopes for motions along the

fixed axes. In contrast, in an assembly planning application, interacting pairs may change during the removal motions.

Separability and assembly planning. Toussaint [69] surveys earlier methods for separating sets in two and three dimensions. For recent surveys on assembly planning, cf. [12, 22]. Geometric assembly planning is studied in [71]. An application of separability in medicine is described in [30]. Agarwal, de Berg, Halperin and Sharir [1] consider sequences of translations for separating polyhedra. There, the set of allowed motion directions is assumed to be given in advance. In [65], the concept of blocking graphs is introduced. This concept allows for deciding whether there is a *subassembly* S of a given assembly, such that S is removable by *a single translation*. The output of the corresponding algorithm consists of S and a translational direction v for removing S, if there is such a subassembly S. This computation is possible in polynomial time, even though the number of removable subassemblies is exponential in general. The algorithm computes a valid subassembly and a removal direction if there is such a subassembly, but avoids enumerating the entire set of possible subassemblies. This simple approach allows for several extensions (see e.g. [37]), but inherently requires that all *moving* parts perform the same motion. Wilson et al. [72] describe extensions of the basic techniques. Guibas and Halperin et al. [18] consider *infinitesimal* translations and rotations for partitioning three-dimensional sets. Such infinitesimal motions can indicate directions for removing parts in a single step. However, it cannot be guaranteed that the corresponding extended motion will be collision-free. Pollack, Sharir and Sifrony [52] describe a near-optimal method for separating two polygons translating in the plane. Their analysis is based on computing the boundary of a connected component in a two-dimensional arrangement. This connected component represents the set of placements that are reachable by the moving polygon. A generalization of their techniques to the case of multiple moving polygons would have to compute the boundary of a connected component in a d-arrangement. An example in [6] provides an *exponential lower bound* for the number of translations necessary to separate objects (see also [49]). This lower bound already holds for a restricted class of polygons in the plane. Interestingly, in special cases our technique allows for establishing infeasibility in polynomial time, even if the boundary of the set of reachable placements has exponential complexity. One of the few exact and complete algorithms that is theoretically capable of finding sequences of translations to separate polygons is described in [23]. However, that approach is limited to sequences of very small constant length, and its practicality is doubtful (no experimental evaluation has been given).

1.3 Contribution and Thesis Overview

Previous research on exact and complete algorithms in the fields of motion planning and assembly planning reveals a gap; on the one hand, there are general theoretical algorithms that are capable of solving any geometric motion planning problem in an exact and complete way [60, 3]. Unfortunately, the underlying mathematical techniques are impractical in higher dimensions and have not been implemented

except for very simple settings with a very small constant for the dimension of the configuration space [48]. On the other hand, exact and complete algorithms that are of practical value are limited to either low-dimensional problems or subclasses of problems. [52] considers sequences of translations to separate two simple polygons, [35] allows rotations, but restricts the moving parts to a single convex polygon. [71] considers realistic assembly planning examples but restricts motions either to infinitesimal motions or single translations of a single subset of parts. The local techniques proposed in [40, 30] for other complex planning problems are exact or perform exact computations on a first-order approximation. However, local approaches are not complete and cannot establish infeasibility.

Our work provides a first step towards algorithms for solving practically relevant instances of high-dimensional separability problems in an exact and complete way. Of course, we cannot break worst-case lower bounds. However, we believe that the incremental algorithms proposed in this thesis are an important approach, since their actual computational complexity depends mainly on the complexity of the specific problem instance and not on the worst-case complexity for the whole class. We focus on the problem of translational assembly planning of polygonal and polyhedral parts. Parts can have any topology and can only be assembled by a sequence of part translations, possibly simultaneous. The key property is that contact relations between pairs of parts can be expressed as linear constraints and can be represented as hyper-planes embedded in a higher dimensional space, called the *composite configuration space*. This configuration space partitions into an arrangement of cells which can be searched for an assembly path. We propose a general framework for translational assembly planning based on efficient solution of linear constraints. Within this framework, we present new algorithms for three problems: (1) efficient testing of linear unboundedness, (2) m-handed assembly, and (3) general translational assembly planning.

Some of the results in this thesis appeared or have been accepted as conference and journal publications. In particular, [61] contains parts of Chapters 2 and 4. The basics of Chapter 3 are described in [63, 64]. In [62], we focus on the applications of Chapter 4 described in Chapters 2 and 3.

The rest of this dissertation is organized as follows:

In Chapter 2, we discuss the problem of separating a given overlap-free placement of polygons or polyhedra into two or more subsets by collision-free simultaneous one-shot translations. It is shown that this problem is closely related to linear unboundedness testing. In this context, we present novel algorithms and new NP-completeness results.

In Chapter 3, we consider arbitrary sequences of translations that allow for removing at least a single part from the remaining parts without causing overlap between any two parts. Although general translational separability is known to be PSPACE-hard, many constrained instances of the problem are much simpler to solve. We therefore propose a framework for exact composite configuration space computation and derive complete algorithms that are of practical value. Our goal is not to derive a collection of intuitive special-case heuristics but to show that a uniform approach can be used to handle a variety

of practical problem instances with different properties. To reduce a significant part of the exponential complexity of a composite configuration space, we use an implicit representation and introduce the techniques of *configuration space expansion* and *D-node reduction*.

In Chapter 4, we address the problem of testing for linear unboundedness. Linear unboundedness testing consists of finding a direction in which the feasible region of a given set of linear constraints is unbounded. For conjunctions of constraints, this region is the intersection of d-dimensional half-spaces forming a (possibly unbounded) convex polyhedron. We describe a new efficient unboundedness testing algorithm that relies on solving linear equations and linear feasibility testing. The algorithm is shown to be optimal when optimal algorithms are used for the two main steps. It has specific practical applications in separability analysis (Chapters 2 and 3) and can also be of more general use. We also show that linear unboundedness testing becomes NP-complete for very simple AND/OR constraint sets.

In Chapter 5, we discuss details that are important for a practical implementation of our methods in the previous chapters. In particular, we consider (1) linear feasibility testing for cells in a composite configuration space, (2) efficient data structures for marking visited cells, and (3) deriving linear constraints for pairs of polygons and polyhedra.

In Chapter 6, we evaluate the practicality of our algorithms for efficient linear unboundedness testing, for m-handed separability and for general translational separability. The chapter also summarizes results and measured running times for unboundedness testing of randomly generated high-dimensional convex cells, and for separability of a variety of two- and three-dimensional part placements.

Chapter 2

One-shot Translational Separability

In this chapter, we analyze the problem of separating a given overlap-free placement of polygons or polyhedra into two or more subsets by collision-free simultaneous one-shot translations. We propose novel algorithms and heuristics, and obtain new NP-completeness results.

We first provide a classification into different subproblems and types of one-shot translations, according to the number of subsets that move with different velocities. We discuss in detail the case of m-handed separability of polygons and show that it is related to linear unboundedness testing. In an m-handed translation, parts must move simultaneously but each part may move with its own constant velocity. We identify three classes of problem instances according to the initial relative part positions. It will turn out that if all pairs of parts are separated in the initial placement, a collision-free infinite m-handed translation can be computed with a simple linear-time algorithm. In contrast, if no pair of parts is separated, the problem becomes more difficult but can still be solved efficiently. For this case, we present the first deterministic polynomial-time algorithm. It is the third class with both initially separated and non-separated pairs of parts that makes m-handed separability NP-hard. We propose powerful heuristics and an algorithm based on the branch-and-bound paradigm to tackle these problem instances. We also present the first proof for NP-completeness of m-handed separability of polygons based on a reduction from *PARTITION* for integers.

Thereafter, we consider c-handed separability. In this case, at most c subsets of parts may translate with different velocities, and c is regarded a small constant. We suggest that this problem can be reduced to a c-handed assignment problem: the unknown velocities of the unknown moving subsets can be first derived from an arrangement of hyperplanes in subassembly velocity space, a kind of configuration space of constant dimension and therefore polynomial complexity. We prove that both velocity assignment and c-handed separability are NP-complete problems for $c = 2$. This will be an important result, since deterministic polynomial-time algorithms are known for the special case $c = 1$.

Although we will limit our discussion to polygons and polyhedra, the principles in this chapter also apply to curved part shapes, as long as separating directions for pairs of parts can be computed.

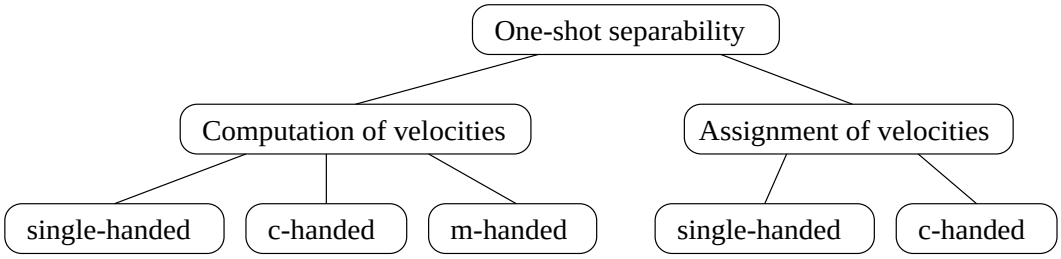


Figure 2.1: One-shot translational separability

2.1 Overview and Classification

We consider simultaneous infinite ('one-shot') translations that allow for separating a given initial placement of 2- or 3-dimensional parts into two or more subsets. Moving subsets may translate with different velocities, as long as these translations are performed simultaneously. The part motions have to be collision-free, that is, they must allow for arbitrary extension without causing interferences between parts at any point of time. We assume parts to be open sets, thus permitting touching or overlapping boundaries.

Definition 1 (Simultaneous one-shot separating translation) *Given an initial overlap-free placement of 2- or 3-dimensional parts, a one-shot separating translation is an infinite simultaneous translation of one or more parts. Each part may move with its own constant velocity (or rest), but parts must not interfere (intersect in their interiors) and at least one pair of parts must move with distinct velocities. We call such a one-shot translation m-handed if it involves at most m different non-zero velocities.*

Note that it is not required that the velocities be constant during the motion. However, their directions must be fixed. For each pair of velocities, the ratio of their absolute values must not vary over time. Thus, in practice, there are infinitely many ways to execute a specific one-shot translation. Disregarding time, all equivalent ways to execute a one-shot translation are represented by a single ray in the composite configuration space of all parts. We will therefore implicitly assume constant velocities. In this context, a part velocity can always be uniquely specified by two placements of the part at two fixed points in time. Since we assume that an initial placement is given, we may equivalently represent velocities as tuples consisting of a placement and a certain point in time.

Following the assembly planning literature, we call a motion m-handed if it requires m hands to execute. This should not be taken too figuratively, however, since we ignore connectedness or stability issues of parts moving with the same velocity. In our context, the term m-handed rather states that at most m different non-zero velocities are involved.

Note that a different interpretation concerning the number of hands can be found in the literature: in [65, 72], the resting parts are assumed to require an additional hand. There, our single-handed separability problem is called two-handed.

In its most general form, the one-shot separability problem consists of finding an infinite separating translation, if one exists. We define two subclasses depending on whether a set of velocities has to be computed from scratch or is given in advance. (When candidate velocities are given, the remaining problem is to assign them to the different parts.) These subclasses will be divided further in turn, depending on the number of hands. Fig. 2.1 gives an overview:

1. **Computation of velocities.** For each part, a velocity vector (possibly zero) must be computed.

   (a) **Single-handed separability**. A single removable subset together with a direction vector (normalized velocity vector) has to be computed. Schweikard and Wilson [65] present a polynomial-time algorithm for this case. As a consequence, infeasibility for a single hand can be proved in deterministic polynomial time, too. The algorithm is remarkable, since a naive approach that successively tests all subsets of parts for removability would take exponential time in the worst case.

   (b) **M-handed separability**. In this case, each part is allowed to move with its own velocity, or remain stationary. Since the worst case complexity of all known motion planning problems is exponential in their degrees of freedom, one may expect this class to be NP-hard. We will prove that this is true by a reduction from the well-known *PARTITION* problem for integers. Nevertheless, most practical instances are much simpler and we will present general and powerful heuristics to tackle them.

   (c) **C-handed separability**. Limiting the maximum number of independently moving subsets to a constant c, we obtain the c-handed separability problem. In fact, the above single-handed separability problem is contained in this class. The reason for yet featuring the single-handed problem as a class of its own is that its computational complexity is unique. As we will see, already two-handed separability is an NP-complete problem.

2. **Assignment of velocities.** Given a set of velocity vectors, the task is to assign them to the parts so that the resulting one-shot translation is collision-free. An efficient algorithm for the single-handed assignment problem has been presented in [1]. As it is the case for c-handed velocity computation, we will show that the assignment problem becomes NP-hard for $c = 2$.

2.2 M-handed Separability of Polygons

The m-handed separability problem is stated as follows:

Definition 2 (M-handed separability) *Given an initial, overlap-free placement of parts, find a single simultaneous one-shot translation that allows for removing at least one part without collisions. Each part may translate in a different direction with its own constant velocity, as long as these motions are performed simultaneously.*

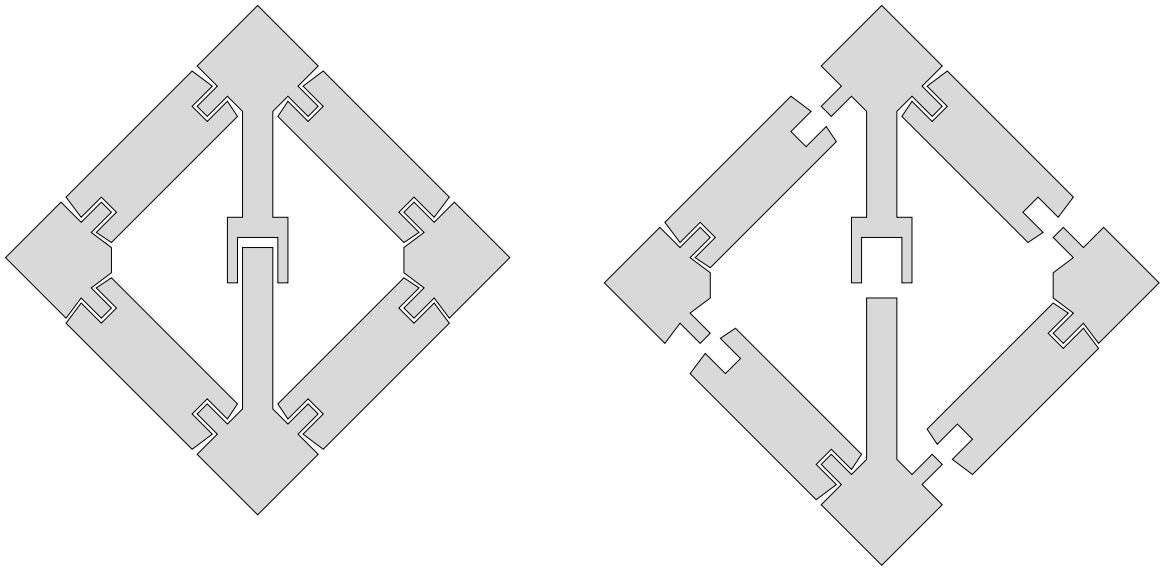


Figure 2.2: Initial placement of polygons and snapshot during one possible three-handed separating translation.

In the following, we consider only the polygonal case. Differences and necessary extensions for polyhedra will be discussed at the end of this section. Fig. 2.2 gives an illustrative example for the planar case. It shows a given initial placement of polygonal parts on the left, and a snapshot during a valid three-handed separating translation with one stationary subset and three subsets moving in different directions. Note that this is only one possible m-handed translation.

To approach the m-handed separability problem, we first consider isolated pairs of parts. We then combine the concepts derived from pairs to tackle the problem as a whole. It will finally turn out that in general, m-handed separability is NP-complete, but there are two classes of problem instances that can be identified easily and solved in linear and deterministic polynomial time, respectively. For the other cases, we propose simple yet conceptually general and powerful heuristics, showing that many general instances can be solved efficiently in practice.

2.2.1 Constraints for Pairs of Parts

Let $P_1, \ldots, P_k$ be polygons in a given initial placement which is, by definition, overlap-free. Since we consider only translational motions, the position of P_i is given by a vector $\mathbf{p}_i = (x_i, y_i)$. We assume $\mathbf{p}_i = (0, 0)$ for the initial placement, so $\mathbf{p}_i \neq (0, 0)$ describes the location of polygon P_i after a translational displacement.

No pair of polygons shall intersect during a valid separating motion. Initially, we consider isolated pairs of polygons, because this case is well understood and provides a good basis for the following concepts. The configuration space obstacle $CO(P_i, P_j)$ of two polygons describes the set of relative placements of P_i with respect to P_j for which P_i and P_j will overlap [39]. We regard all parts to be

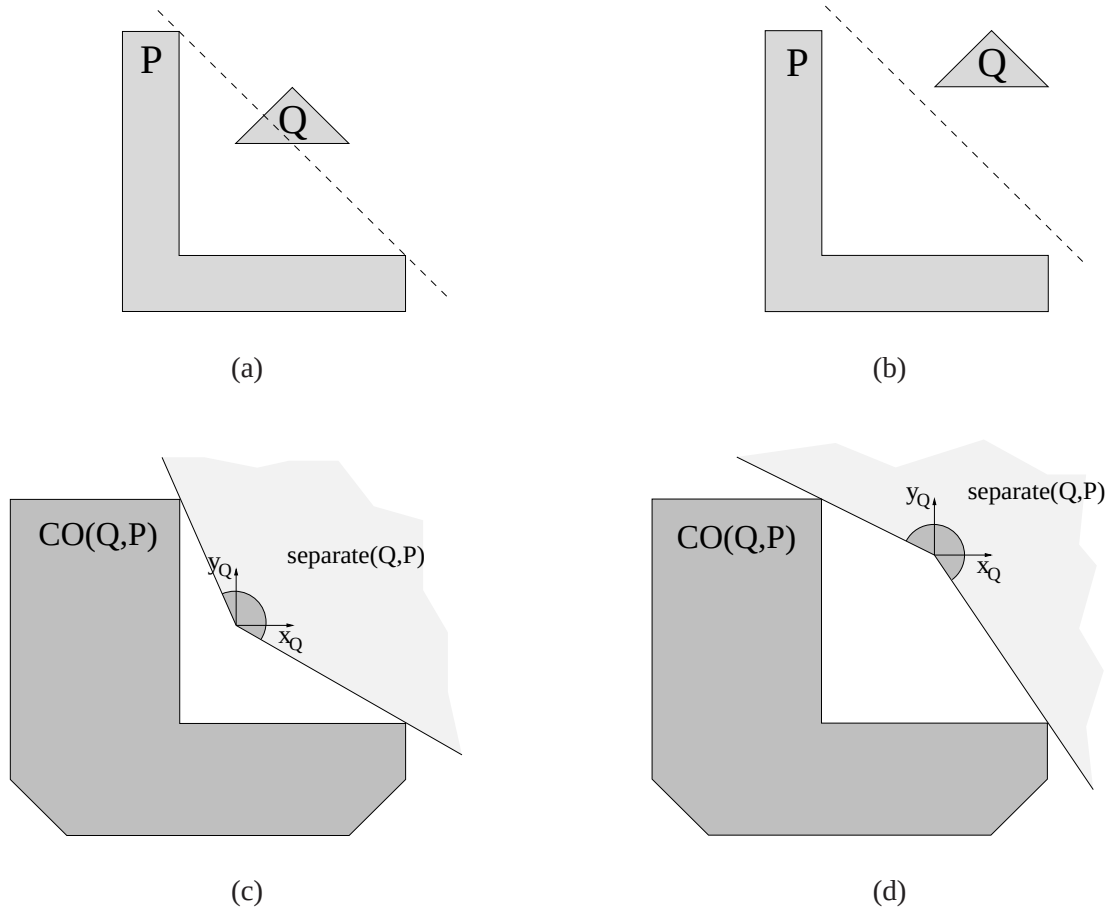


Figure 2.3: (a) Non-separated and (b) separated parts and their respective configuration spaces and feasible directions (c,d)

open sets. Consequently, the complement of $interior(CO(P_i, P_j))$ represents relative placements of polygons P_i and P_j in which their interiors do not intersect.

For a pair of polygons (P_i, P_j) in their initial placement, where $1 \leq i < j \leq k$, we now consider the set $separate(P_i, P_j)$ of one-shot translations separating P_i from P_j without collisions when P_j is fixed (motions of P_i relative to P_j). This set of translations is represented by a cone of rays emanating from the origin (in the plane).

We distinguish whether the polygons are separated or not in their initial placement. In general, two sets are said to be *separated* if there exists a hyperplane $\mathbf{ax} = b$ such that one set is entirely contained in the half-space $\mathbf{ax} \geq b$ while the other set is entirely contained in the (open) half-space $\mathbf{ax} < b$. For example, the polygons Fig. 2.3.a are not separated, while those in Fig. 2.3.b are.

If P_i and P_j are initially *non-separated*, the interior angle of $separate(P_i, P_j)$ is at most π. This is illustrated in Fig. 2.3.c which shows the configuration obstacle for the polygons in Fig. 2.3.a and the cone of separating directions. For a non-separated pair of polygons, the cone is thus convex and

can be represented by an intersection of two half-planes or, equivalently, as a conjunction of two linear constraints

$$(\mathbf{a}_1^{(ij)}\mathbf{x} \geq 0) \wedge (\mathbf{a}_2^{(ij)}\mathbf{x} \geq 0)$$

where the two-dimensional vector $\mathbf{x}$ contains the coordinates of P_i relative to P_j.

In contrast, if P_i and P_j are initially *separated*, the interior angle of this cone of free directions is greater than π (Fig. 2.3.d) and we need a disjunction of two linear constraints to represent this set:

$$(\mathbf{a}_1^{(ij)}\mathbf{x} \geq 0) \vee (\mathbf{a}_2^{(ij)}\mathbf{x} \geq 0)$$

These combinations of inequalities form a succinct representation of possible relative placements for the two polygons during a valid separating translation. To obtain the valid placements for all polygons, we have to consider the constraints for all pairs of parts *simultaneously*.

The problems of computing Minkowski sums and configuration spaces for pairs of polygons have been extensively studied in the literature [42, 33, 55]. Here, we used this concept mainly for illustration. In practice, the constraints for a pair of polygons can be derived directly without the need of explicitly computing the configuration space obstacle [70].

2.2.2 Embedding Pairwise Constraints

A *simultaneous placement* of all polygons $P_1, \ldots, P_k$ is represented by a vector $(x_1, y_1, \ldots, x_k, y_k)$ in E^{2k}. The origin in this space describes the initial placement. For example, the vector $(1, 0, \ldots, 0)$ is the placement, possibly overlapping, obtained by translating the first polygon along the positive x-axis by one length unit. E^{2k} will be called the *composite* or *assembly configuration space*, since it contains simultaneous placements of all polygons.

A ray in E^{2k}, emanating from the origin, represents simultaneous infinite translations of all polygons. To obtain only the collision-free motions, we combine the constraints derived from all pairs of parts. Therefore, it becomes necessary to transform the coordinates underlying the pairwise constraints to a global reference frame.

Up to now, we assumed that for each pair of polygons one part was the fixed reference part while the other part was allowed to translate. This amounts to representing the coordinates of the movable polygon in a coordinate frame attached to the reference polygon. We now need a coordinate transform from local (part-relative) to global (absolute) coordinates. In the case of translations, this is particularly simple and consists of substituting $\mathbf{x} = \mathbf{p}_i - \mathbf{p}_j$ in the two-dimensional constraints derived from the sets $separate(P_i, P_j)$.

Performing this substitution, we obtain four-dimensional linear constraints on the absolute polygon positions $\mathbf{p}_i$ and $\mathbf{p}_j$

$$\begin{aligned} C_1^{(ij)} : \mathbf{a}_1^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) &\geq 0 \\ C_2^{(ij)} : \mathbf{a}_2^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) &\geq 0 \end{aligned}$$

which are joined by a conjunction or disjunction as before. Each ray in the set described by these constraints is a direction of simultaneous translation of both P_i and P_j, during which P_i and P_j will not collide.

We embed these constraints in the high-dimensional assembly configuration space of all polygons by simply regarding the inequalities as $2k$-dimensional inequalities with at most four non-zero coefficients. Then, the constraints for all pairs of parts can be combined into a single expression of linear constraints

$$ONESHOT(P_1, \ldots, P_k) \equiv \bigwedge_{(i,j)\in\mathcal{N}} (C_1^{(ij)} \wedge C_2^{(ij)}) \quad \wedge \quad \bigwedge_{(k,l)\in\mathcal{S}} (C_1^{(kl)} \vee C_2^{(kl)}) \tag{2.1}$$

where $\mathcal{N}$ is the set of initially non-separated pairs and $\mathcal{S}$ is the set of initially separated pairs of polygons. To re-emphasize, for the initially non-separated pairs, the set of separating directions is convex and is represented by a conjunction of two half-space constraints $C_1^{(ij)} \wedge C_2^{(ij)}$. For an initially separated pair, the corresponding set is nonconvex and is split into a disjunction of two half-space constraints $C_1^{(kl)} \vee C_2^{(kl)}$.

$ONESHOT(P_1, \ldots, P_k)$ filters those rays that represent feasible simultaneous one-shot translations starting at the given initial placement. This still includes the subspaces $\{(v_x, v_y, v_x, v_y, \ldots, v_x, v_y) \mid (v_x, v_y) \in E^2\}$, since it is always possible to move all parts with the same velocity without introducing collisions. To exclude them, we add simple constraints

$$FIX(P_j) \equiv (x_j = 0 \wedge y_j = 0)$$

to fix the placement of a single arbitrary part P_j. Now, a ray that emanates from the origin and satisfies all constraints represents a valid one-shot translation. The question is therefore whether we can efficiently compute such a ray.

The linear constraint expression $FIX(P_j) \wedge ONESHOT(P_1, \ldots, P_k)$ describes a union of convex cones in high-dimensional space. This can be seen by transforming it into disjunctive normal form (DNF). In the normal form, each clause is the intersection of a finite number of half-spaces whose bounding hyperplanes contain the origin. A ray in this set indicates an unbounded direction. Since efficient linear unboundedness testing and computation of a feasible ray is possible for convex sets (Chapter 4), we could first transform the constraints into DNF and then process the clauses one after another. The normal form expression may have an exponential number of clauses, each of them representing a convex cell. However, many of the cells could be bounded due to the special structure of the constraints, and it is therefore a natural question to ask for the minimum number of unbounded convex sets needed to describe the set of separating directions of all polygons. The following lemma shows an exponential lower bound for the worst case.

Lemma 1 *The number of convex cones required to represent simultaneous one-shot translations of k polygons is at least $k!$ in the worst case.*

Proof: Consider the given initial placement of k squares in Fig. 2.4.a. Adjacent squares are separated by at least the side-length length of a single square. There are $k!$ distinct critical placements that correspond

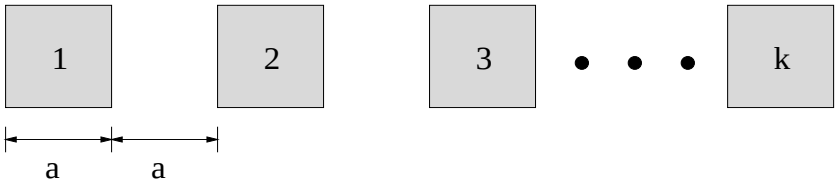


Figure 2.4: Initial placement of squares.

to all permutations of the squares when stacked vertically with their vertical edges aligned. In each such critical placement we assume that the vertical clearance between two adjacent squares equals at least the side-length of one square. Each pair of critical placements corresponds to a pair of points in configuration space that cannot directly see each other. This is because connecting any two critical placements by a linear segment in configuration space would require at least one pair of squares to change positions directly. We have to show that each critical placement is reachable from the initial placement on a simultaneous one-shot translation without collisions. First note that for all pairs (i, j), the relative motion of square i with respect to square j is clearly free of intersection: in the initial placement, the parts have sufficient horizontal clearance and then, in the vertical arrangement, they have sufficient vertical clearance. Now, since this is true for all pairs, linear interpolation between the initial and the critical placement is possible without causing any interference.

To summarize, we have shown that there are $k!$ unbounded directions that must be contained in distinct convex cells. □

Lemma 1 shows that in general, it is impractical to compute the complete convex cone decomposition of free configuration space if all parts are allowed to translate with different velocities. In contrast, when all moving parts are required to move with the same velocity (single-handed separability), a complete convex cone decomposition can be computed in polynomial time [65]. The obvious difference is that the dimension of the underlying parameter space for single-handed separability is constant, whereas in the m-handed case, the dimension depends on the number of parts. (It is a well-known fact that the complexity of linear arrangements grows exponentially with the dimension of the space. In Section 2.3 we will show that this is not the only fact that makes one-shot translational separability a hard problem.)

Note that although the example in the above proof exhibits a combinatorial configuration space, in cases like this it is not necessary to compute a complete convex cone decomposition to find valid m-handed one-shot translations. This will become evident in the next subsection, where we derive general conditions under which m-handed separability can be solved efficiently, outline algorithms that exploit such special properties, and mark out the class that contains the really hard problem instances.

2.2.3 Subdivision and Efficient Algorithms

A) All pairs of parts are initially separated

For this case, there is no need to consider the constraints derived above. As shown in [11], any collection of k star-shaped objects with disjoint interiors can be separated with $k-1$ hands using simultaneous one-shot translations. It follows that if all pairs of parts are separated in the initial placement, there is always a valid m-handed separating translation. Note that this is true for sets in any dimension. Parts need not even be star-shaped, as long as all pairs of parts are initially separated. The same arguments apply to assemblies consisting only of convex parts.

B) No pair of parts is initially separated

For planar assemblies in which no pair of parts is separated in the initial placement, a motion can be computed in polynomial time. Fig. 2.5 shows an example that can be separated by a four-handed one-shot translation. We briefly outline an efficient algorithm for this subclass: given only non-separated pairs, the constraint expression $ONESHOT(P_1, \ldots, P_k)$ is a single conjunction of constraints and therefore the set it describes is already convex. That is, the complete convex cone decomposition of the configuration space consists of a single cell. Finding a ray in a convex cell or showing that no such ray exists can be done efficiently (cf. Chapter 4). If no pair of parts is initially separated, the m-handed separability problem therefore reduces to testing a single convex cell for unboundedness. Tab. 2.1 summarizes the algorithm.

C) General case: initially both separated and non-separated pairs of parts

In the general case with both separated and non-separated pairs of polygons, the constraints in the expression $ONESHOT(P_1, \ldots, P_k)$ are joined by several disjunctions, one per initially separated pair of polygons. We would have to transform the constraints into disjunctive normal form to obtain convex cells which could then be tested for unboundedness. However, this approach does not yield an efficient algorithm, since there may be an exponential number of resulting clauses to test. In general, the problem of m-handed separability of polygons is NP-complete. We will prove this further below. However, NP-completeness states the worst-case complexity. In fact, for many NP-complete problems it is the case that a large number of practical instances can be solved quite easily [8, 51]. This motivates heuristic concepts, such as the ones to be described next; they allow for effective reduction of the computational complexity of the naive approach.

2.2.4 Reduction Heuristics for the General Case

In our case, the combinatorial explosion of the constraint set when converting the constraint expression $ONESHOT(P_1, \ldots, P_k)$ into DNF is caused by the disjunctions introduced by initially separated pairs of polygons. So far, we derived the separating directions for each pair of polygons while ignoring all

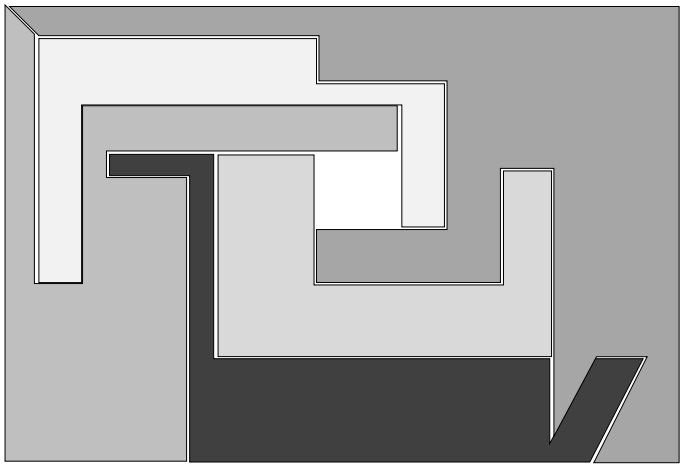


Figure 2.5: Initial placement with no separated pair of polygons.

Input: Overlap-free placement of pairwise non-separated polygons $P_1, \ldots, P_k$.

Output: An m-handed separating translation $\mathbf{u} = (x_1, y_1, \ldots, x_k, y_k)$ (unbounded direction) if the polygons can be separated with m hands or *failure* otherwise.

1. Verify that all pairs of polygons are non-separated. Exit on failure.
2. For $1 \leq i < j \leq k$ compute the (convex) sets $separate(P_i, P_j)$ and linear constraints to describe them
$$(\mathbf{a}_1^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0) \ \wedge \ (\mathbf{a}_2^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0)$$
3. Combine the constraints from all pairs into a single conjunction of constraints
$$ONESHOT(P_1, \ldots, P_k) \equiv \bigwedge_{1 \leq i < j \leq k} \Big((\mathbf{a}_1^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0) \ \wedge \ (\mathbf{a}_2^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0) \Big)$$
and run the unboundedness testing algorithm in Chapter 4 on the cell defined by
$$x_1 = 0 \ \wedge \ y_1 = 0 \ \wedge \ ONESHOT(P_1, \ldots, P_k)$$
Return the result of the unboundedness test.

Table 2.1: Efficient algorithm for computing an m-handed separating motion of initially non-separated polygons.

other parts. This yields correct but often unnecessarily lax constraints. In many cases, the actually possible relative directions for a pair of polygons can be reduced due to the presence of other parts.

This observation motivates an additional preprocessing step whose aim is to reduce the number of pairs which require disjunctions of constraints and thus the number of necessary unboundedness tests for finding an m-handed separating translation. In the following we describe a conceptually simple but very general and effective method based on projecting the embedded constraints from initially non-separated pairs of polygons into all pairwise configuration spaces. The following examples illustrate this concept:

- A pair of initially separated polygons might not be in danger to interact at all due to the presence of other parts. Fig. 2.6.a illustrates this situation: part C restricts the separating directions of A and B so that A and B cannot interact during a one-shot translation. In the embedded space, the nonconvex pairwise constraints from (A, B) are fulfilled by all placements that fulfill the more restrictive convex constraints from pairs (A, C) and (B, C). Therefore, the constraints derived from (A, B) are redundant and could even be ignored altogether.

- A pair of initially separated polygons might be able to interact, but other parts restrict the possible relative directions for the considered pair to a convex set. Fig. 2.6.b illustrates this situation: although separated parts A and B can collide, their relative motion directions are restricted by C to a single convex set. This set reflects the fact that A must move out before B.

More generally, we can analyze the influence of the constraints from all initially non-separated pairs of polygons. We combine these constraints to describe a single convex cell $\mathbf{A}\mathbf{x} \geq 0$. This cell which we call the *base cell* can be projected into all pairwise configuration spaces. Each such projection is either of the following: the entire plane, a single 2-dimensional convex cone, or simply the origin. In each pairwise configuration space we use the projection to determine how the allowed motion directions for the considered pair are further restricted by other parts. Specifically, separating directions for initially separated pairs may be restricted to a convex set.

In Figs. 2.6.a–c, the base cells consists of the embedded constraints from (A, C) and (B, C). The projections $proj$ of these cells in the configuration space of (A, B) are shown in Fig. 2.6.a'–c', respectively. The figures also show the separating directions $separate(A, B)$ derived directly from the part geometries of A and B. We consider the intersection of the projected base cell $proj$ and the set $separate(A, B)$. In the first two cases (Figs. 2.6.a',b'), this intersection is a single convex set of directions and can be described by two linear constraints. We may thus replace the initially derived disjunction of constraints for $separate(A, B)$ by a conjunction of constraints. In the third case (Fig. 2.6.c'), however, the intersection decomposes into two convex sets. There, a disjunction of constraints will still be necessary.

The projections of the base cell can be obtained using a simple algorithm that is inspired by the more general *Convex Hull Method* proposed in [28]. Two observations are important here. First, the base cell is a single convex cone in the composite configuration space. Thus its projection in a pairwise

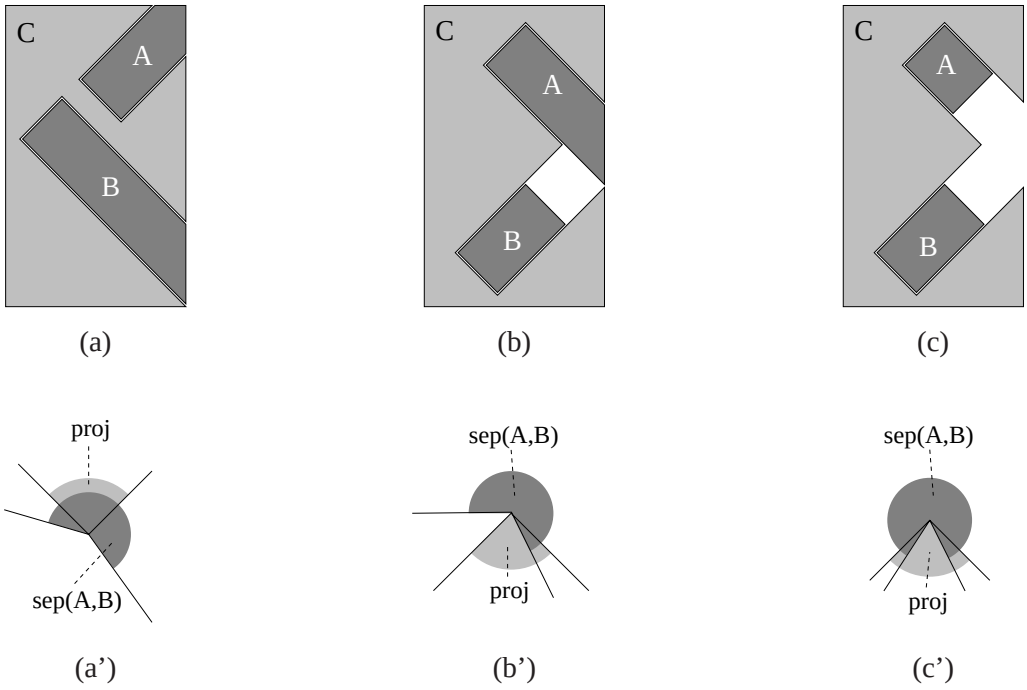


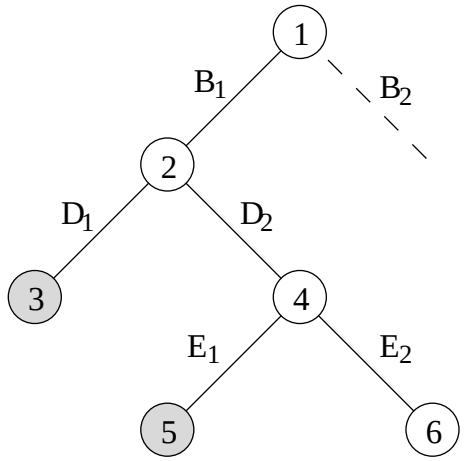
Figure 2.6: Three situations in which the motions of two separated parts A and B are restricted by a third part C: (a) A and B can only move in one direction each and cannot collide in a single translation, so their pairwise constraints are not needed; (b) A and B can collide, but their relative motion directions are restricted by C to a convex set: A must move out before B; (c) C restricts the set of relative directions of A and B but the set of allowed directions is nonconvex: the parts can be disassembled by simultaneously moving A quickly and B slowly, or vice versa. (a'),(b'),(c') show the projections $proj$ of constraints from $\{separate(A, C), separate(B, C)\}$ (base cell) into the configuration space of parts A, B (B fixed).

configuration space will be either the full plane, a single 2-dimensional convex cone, or consist only of the origin. Second, the dimension of the projection space is two, a small constant. We can therefore explicitly compute the vertices of the projection by a series of linear optimizations in different directions in the plane, without the danger of being overstrained by an exponential number of them. A minor problem is that in general the base cell is unbounded. We therefore intersect it with a unit box around the origin and compute the projection of this intersection. Further details and an example are given in Appendix A.2.

2.2.5 Heuristic Algorithm for the General Case

The heuristic preprocessing in the previous subsection aims at reducing the number of disjunctions in the expression $ONESHOT(P_1, \ldots, P_k)$. In general, we will still be faced with disjunctions after this preprocessing, and the constraints may still expand to an exponential number of convex cells.

A branch-and-bound approach may be used to cut down the number of necessary unboundedness tests. The general branch-and-bound paradigm has also successfully been applied to other linear and mixed integer programming applications [68]; we therefore only illustrate the principles in our applica-



Node	Constraints	Result
1	$\{A_1, A_2, C_1, C_2\}$	(unbounded)
2	$\{A_1, A_2, C_1, C_2, B_1\}$	(unbounded)
3	$\{A_1, A_2, C_1, C_2, B_1, D_1\}$	(bounded)
4	$\{A_1, A_2, C_1, C_2, B_1, D_2\}$	(unbounded)
5	$\{A_1, A_2, C_1, C_2, B_1, D_2, E_1\}$	(bounded)
6	$\{A_1, A_2, C_1, C_2, B_1, D_2, E_2\}$	(unbounded)

Figure 2.7: Top: evaluation tree for an AND/OR constraint expression. Bounded nodes are shaded. Bottom: cells tested for unboundedness at the nodes in the above tree.

tion, using a simple example. Suppose that we are given the expression

$$(A_1 \wedge A_2) \wedge (B_1 \vee B_2) \wedge (C_1 \wedge C_2) \wedge (D_1 \vee D_2) \wedge (E_1 \vee E_2)$$

where each atom is a single homogeneous linear half-space constraint, that is, e.g. $A_1 \equiv \mathbf{a}_1\mathbf{x} \geq 0$. We test this expression for unboundedness by traversing a binary tree as shown on the top of Fig. 2.7. Nodes in this tree represent conjunctions of constraints and can therefore be tested efficiently. Each child node contains the constraints of its father plus a single additional constraint which is one alternative of a specific disjunction. In the figure, each edge is labelled with the constraint added to the constraints of its father node.

We traverse the tree in depth-first order, backtracking at bounded nodes, until we find an unbounded leaf node. The table at the bottom of Fig. 2.7 exhibits the constraint sets corresponding to the nodes in the tree, in the order they are visited, and the result of unboundedness testing for each node. The root node of the tree contains the set of all constraints that are joined by conjunctions exclusively, $\{A_1, A_2, C_1, C_2\}$. We assume that, in the example, this set yields an unbounded cell. We therefore add the first alternative from the disjunction $(B_1 \vee B_2)$ to the constraints in the root node, thus forming node 2. If testing this node yields the result unbounded, we add another constraint from a different disjunction. In this example

we assume this is so and choose the constraint D_1. Let the resulting node 3 be bounded, so we backtrack to node 2 and consider the second alternative there, which is D_2. Having found the resulting node 4 to be unbounded, we must analyze disjunction $(E_1 \vee E_2)$ and select the constraint E_1 which leads to the (bounded) node 5. Backtracking over node 4, we finally arrive at node 6 which we assume to be unbounded. Furthermore, node 6 is a leaf node, since it contains all conjunctive constraints plus one constraint from each disjunctive clause.

Based on this evaluation paradigm, and using the heuristic reductions described above in a preprocessing step, we can now outline a heuristic algorithm for m-handed one-shot separability of general placements with both separated and non-separated pairs of polygons: first, we derive the constraints for each pair of polygons, then classify the pairs as separated and non-separated, and construct the base cell from the constraints of all non-separated pairs. We then successively project the base cell into all pairwise configuration spaces. We compute new, possibly more restrictive constraints for each pair from the 2-dimensional intersection of the projected base cell with the original separating directions of the considered pair. After each step, constraints of separating directions which have become convex may be included in the set of constraints forming the base cell. After this preprocessing, we finally apply the branch-and-bound paradigm, using efficient unboundedness testing for convex constraint sets, to evaluate the resulting constraint expression. Tab. 2.2 summarizes this heuristic algorithm for the general case.

The performance of the branch-and-bound approach depends on the order in which the different disjunctive clauses are tested. Here, several additional heuristics may be applied to speed up practical cases. For example, one may sort the pairs of parts which require a disjunction of constraints according to their distances. Closer pairs may be expected to introduce more restrictive constraints and therefore bound the cell earlier than pairs of parts which are further apart.

Input: Overlap-free placement of polygons $P_1, \ldots, P_k$.

Output: Unbounded direction (m-handed separating translation) $\mathbf{u}$ if the polygons can be separated with m hands or *failure* otherwise.

1. For $1 \leq i < j \leq k$ compute the sets $separate(P_i, P_j)$ and derive linear constraints

$$(\mathbf{a}_1^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0) \quad \circ \quad (\mathbf{a}_2^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0)$$

   where $\circ = \vee$ if the parts are initially separated and $\circ = \wedge$ if the parts are initially non-separated.

2. Combine the constraints from the set of all non-separated pairs (denoted $\mathcal{N}$) into a single expression

$$BASECELL(P_1, \ldots, P_k) \equiv \bigwedge_{(i,j)\in\mathcal{N}} \Big((\mathbf{a}_1^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0) \quad \wedge \quad (\mathbf{a}_2^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0)\Big)$$

3. Project $BASECELL(P_1, \ldots, P_k)$ into the configuration spaces of all pairs (i, j) and intersect the projections with the current pairwise feasible directions $separate(P_i, P_j)$. If the intersection is a convex subset of the current pairwise feasible directions, replace $separate(P_i, P_j)$ with this intersection and add constraints describing this intersection to $BASECELL(P_1, \ldots, P_k)$. Repeat this step until no further reductions are possible.

4. Derive constraints from the reduced pairwise separating directions to form an expression

$$ONESHOT_r(P_1, \ldots, P_k) \equiv \bigwedge_{1\leq i<j\leq k} \Big((\mathbf{a}_{r1}^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0) \quad \circ \quad (\mathbf{a}_{r2}^{(ij)}(\mathbf{p}_i - \mathbf{p}_j) \geq 0)\Big)$$

   where $\circ = \wedge$ for a convex set of reduced directions and $\circ = \vee$ for a concave set of reduced directions.

5. Evaluate the expression

$$x_j = 0 \quad \wedge \quad y_j = 0 \quad \wedge \quad ONESHOT_r(P_1, \ldots, P_k)$$

   using a branch-and-bound tree and testing the convex sets in the nodes of tree with the efficient unboundedness test in Chapter 4. At the first unbounded leaf node found, stop and return the computed unbounded direction.
   If all leaf nodes are bounded, return *failure*.

Table 2.2: Heuristic algorithm for computing an m-handed separating motion of a general polygonal assembly.

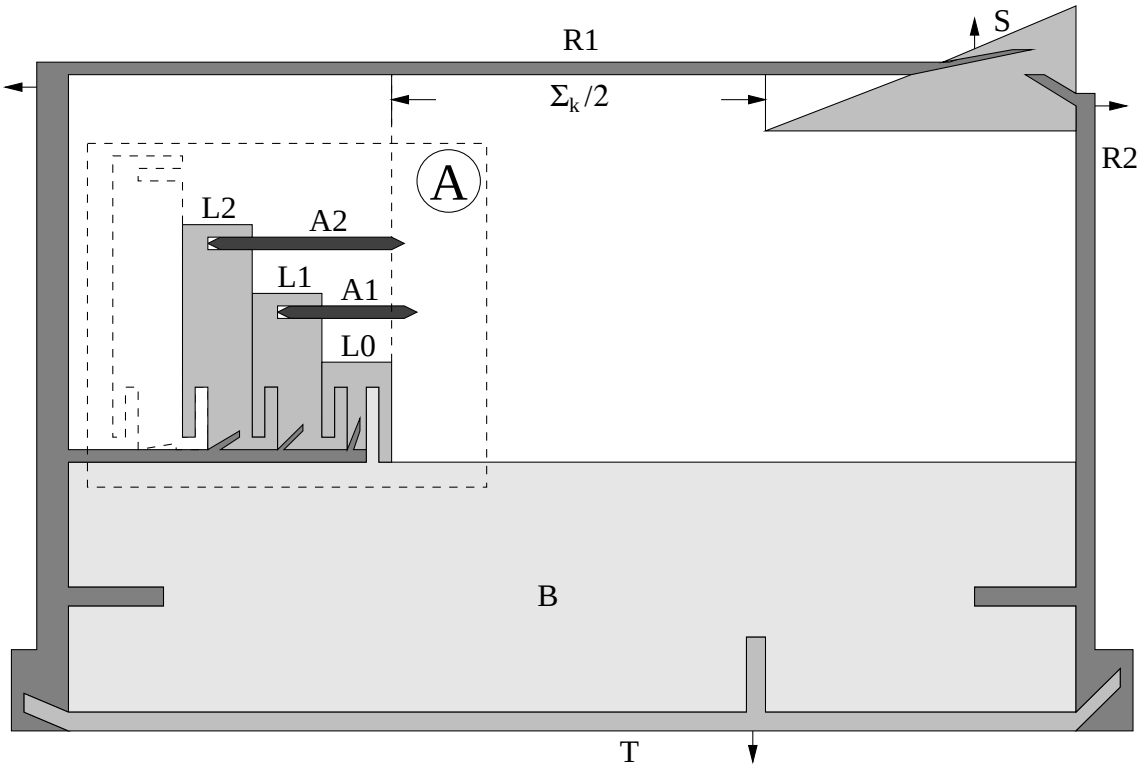


Figure 2.8: Construction principle for reduction from *PARTITION*.

2.2.6 M-handed Separability is NP-complete

We show that the general problem of finding an m-handed separating translation is NP-complete for polygonal parts.

Lemma 2 *M-handed translational separability of polygons is NP-hard.*

Proof: By reduction from *PARTITION* for integers, a well-known NP-hard problem [16]. For a given set of integers $A = \{a_1, \ldots, a_k\}$, we construct an assembly of polygons which can be separated by an m-handed translation if and only if there is a subset $I \subset A$, so that

$$\sum_{i \in I} i = \frac{1}{2} \sum_{a \in A} a$$

Fig. 2.8 shows the construction principle. Note that this figure exhibits only a rough sketch and that parts are not dimensioned properly. There are two main functional units each consisting of several parts: a frame and an internal device A. We first describe the frame. It consists of the parts B, $R1, R2, T$ and S. We assume that the basis B is fixed. $R1$ and $R2$ can only move to the left respectively right and act similarly to a sliding roof. They are coupled by a transmission T which ensures that either both $R1$ and $R2$ move or none. Moreover, we can calibrate the speed ratio for $R1$ and $R2$ by properly changing the interconnection between $R1$ and T on the one side and $R2$ and T on the other side. The sliding roof has

to open in order to remove any of the parts. However, there is an additional part inserted in the frame: the separator S. This part is connected to $R1$ and $R2$ in a way that causes it to move exactly in the vertical direction upwards when $R1$ and $R2$ move with their unique possible speed ratio. S thus partitions the opening gap in the roof.

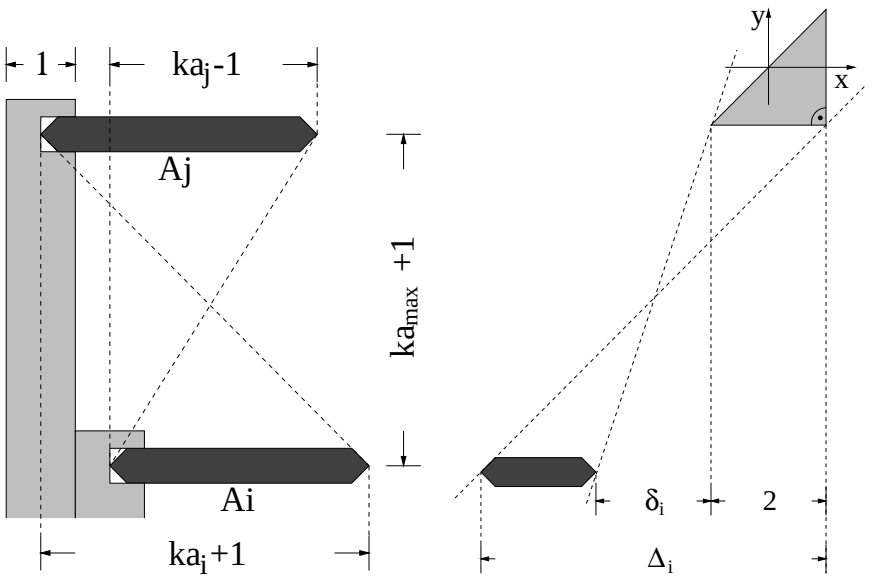
We are now to describe the heart of the construction, namely device A. For each integer number a_j in the input set there is a stick Aj with length proportional to a_j. Each of them is inserted into a lift Lj so that its relative motions with respect to the lift are constrained to the right horizontal direction. Additionally, there is an empty lift $L0$. The lifts are triggered by hooks of different slope in $R1$ and can move only vertically upwards with determined speeds $v_0 > v_1 > v_2 > v_3 > \ldots > v_k$. Thus all sticks must move out of their associated lifts or else they would collide with at least one of the other lifts.

We calibrate the lifts so that the tips of the sticks simultaneously arrive at the level of the gaps in the roof. This level is determined by the upper edge of the peg that joins $R2$ with S. A short time before, $R1$ must have moved far enough to let the leftmost lift pass. The separator will move up with a calibrated vertical velocity so that its bottom simultaneously arrives at one level together with the axes of all sticks. At this point of time, the gap between the separator S and $R2$ on the right side will have width equal to half the sum of the lengths of all sticks. The gap between the rightmost lift and $R1$ will have the same width a priori. Now all parts can escape without collisions if and only if the sticks are properly placed into the two gaps, thus solving the *PARTITION* problem. If the given *PARTITION* problem has a solution, this can be achieved by proper selection of the horizontal velocity components for the sticks.

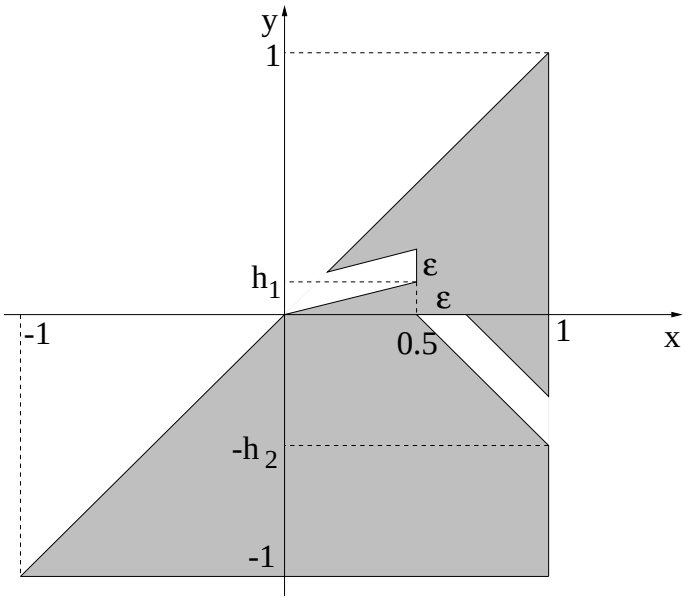
We now prove that the parts can in fact be dimensioned properly to achieve the described behavior. A further important issue will be that the input size is not increased more than by a polynomial. The reader, if not interested in the details, may want to skip the rest of the proof since it is rather technical. To keep the discussion short, we give explicit values for the major coordinates. Thus, we meet the requirements by a unique set of construction rules. Of course there are infinitely many other possibilities for dimensioning the parts.

We begin with the sticks. A stick Ai, representing an integer a_i, has length ka_i and width $\frac{1}{2}$. The ends are slanted by angles of 45° as shown in Fig. 2.9.a. The lifts have unit width each, and so, for each stick, its right tip is guaranteed to be right of the left tips of all other sticks. To escape, a stick must be able to overtake all other sticks above it at any of their ends (shown by slanted and dashed critical lines in Fig. 2.9.a). We place the axes of two adjacent sticks Ai and Aj with a vertical distance of $ka_{max} + 1$, where a_{max} is the maximum number in the input. This guarantees that no pair of sticks will collide when simultaneously translating into any collision-free vertical alignment where all axes are at the same level. We place the top edge of $L0$ with the same vertical spacing below the axis of $A1$. Now it can be shown that no stick will collide with $L0$ when it is overtaken in vertical direction by $L0$. Choosing these distances, the initial vertical distance between the axis of Lk and the top edge of $L0$ will be $k(ka_{max}+1)$.

The next critical element is the separator S. We model it as a small right-angled triangle having two sides of length 2, respectively, and place it in the global coordinate frame as shown in Fig. 2.9.c. The



(a) (b)



(c)

Figure 2.9: Details of the construction: (a) pair of adjacent sticks and lifts, (b) stick and separator, (c) holes in separator for pegs.

horizontal clearance between the rightmost lift $L0$ and the lower left vertex of S will be chosen to be exactly $\sum_k /2$, where $\sum_k = k \sum_{i=1}^{k} a_i$ will in the following denote the sum of the lengths of all sticks. Thus, the horizontal distance between the left end of stick i and the lower right vertex of S will be, as shown in Fig. 2.9.b, equal to $\Delta_i = i + \frac{1}{2} + \sum_k /2 + 2$. (The second and forth addends are the insertion depth of a stick in its lift and the width of S, respectively.) The horizontal distance δ_i between the right end of the stick and the left lower triangle vertex is always smaller but positive and equals $\Delta_i - ka_i - 2$. (We may assume that no stick is longer than $\sum_k /2$.) Thus, giving the axis of stick i a vertical distance of at least Δ_i from the bottom side of the separator triangle S will allow stick i to overtake the separator at any side. Note that it suffices to consider the topmost stick, since the vertical spacing between the axes of adjacent sticks is greater than the horizontal distance of their left tips (which is 1).

Now we consider the peg connecting $R2$ and S. (Fig. 2.9.c shows the corresponding cavity in S.) The horizontal distance of the critical vertex of this peg from the right tip of stick i is bounded by Δ_i. (In fact, it is equal to $\Delta_i - ka_i - \frac{1}{2}$.) Since the critical vertex of the peg is above the bottom edge of S, placing the sticks so that they can pass the separator asserts that they can also pass $R2$. Furthermore, since $R2$ moves horizontally, this critical vertex determines the level ($y = 0$) at which the axes of all sticks and the bottom edge of the separator must arrive simultaneously. For the pegs in the separator, we set

$$h_1 = \frac{1}{2k + \sum_k +7}, \quad h_2 = \frac{1}{\sum_k +1}$$

Since $h_1 < \frac{1}{10}$, we can select $\epsilon = \frac{2}{5}$ for the pegs and the cavities. From now on let us assume that $R1$ and $R2$ move with the velocities (other velocities would be possible but the ratio of their absolute values must be the same)

$$v_{R1} = -\begin{pmatrix} k + (\sum_k +7)/2 \\ 0 \end{pmatrix}, \quad v_{R2} = \begin{pmatrix} (\sum_k +1)/2 \\ 0 \end{pmatrix}$$

It can be shown that then v_S is uniquely determined to be

$$v_S = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

When the bottom of S has arrived at $y = 0$, $R2$ will have moved $\sum_k /2 + \frac{1}{2}$ units to the right, thus creating the desired gap. $R1$ will have moved much further to the left (the left gap is between $L0$ and S), and we have to show that v_{R1} suffices to get $R1$ out of the way of the leftmost lift. We therefore first need to specify the vertical positions of the lifts and sticks.

We place the left tip of stick i at

$$\begin{aligned} x_i &= -(i + \frac{\sum_k +3}{2}) \\ y_i &= -(k + \frac{\sum_k +7}{2}) - (k - i)(ka_{max} + 1) \end{aligned}$$

and the top edge of $L0$ at $y_0 = y_i \mid_{i=0}$ and $x_0 \in [-2 - \sum_k /2, -1 - \sum_k /2)]$. This satisfies the vertical spacing constraints on (1) adjacent sticks and (2) the sticks and S.

Since the axes of all sticks and the top edge of $L0$ must arrive at $y = 0$ simultaneously with the bottom edge of S, the velocity of lift i will be uniquely determined to be

$$v_i = \begin{pmatrix} 0 \\ -y_i \end{pmatrix}$$

Now $R1$ gets out of the way of the leftmost lift Lk in time (and thus lets all lifts pass). We omit further details to show this.

Based on the above parameters, it is not difficult to dimension the remaining elements and show that indeed no pair of parts will collide during a valid partitioning motion. For constructing the pegs in $R1$ which trigger the lifts, one may specify some small constant width and use the fact that the relative velocity of lift i with respect to $R1$ must be equal to $v_i - v_{R1}$. Similar observations apply to the pegs connecting $R1$, $R2$ and T. We omit further details but show that the whole construction can be represented without increasing the input size by more than a polynomial function.

For an instance $A = \{a_1, \ldots, a_k\}$, the input size is $\sum_{i=1}^{k} b_i$ where b_i is the number of bits required to represent a_i. Thus, the input size is of the order $\Omega(k + b_{max})$. After appropriate scaling, we need $O(k)$ integer numbers to represent the vertex coordinates of all parts, and each of them will be in $O(k^c a_{max})$ where c is a constant. We can represent these numbers in total space $O(k \log k + k b_{max})$. Thus, the total increase in space is bounded by a polynomial function. □

Theorem 1 *M-handed translational separability of polygons is NP-complete.*

Proof: It remains to show that the problem is in NP. The proof follows from the discussion in subsections 2.2.1 and 2.2.2. A non-deterministic polynomial-time algorithm first guesses a one-shot separating translation and then verifies it by projecting it into the pairwise configuration spaces. All operations for the latter step can be performed in deterministic polynomial time. □

A closely related problem is finding *finite* simultaneous translations that are collision-free just until the polygons can be partitioned into two subsets by a straight line. Note that this definition of separability is equivalent to our original definition using infinite translations *only* in the single-handed case: for more than a single moving subset, there may be a finite collision-free translation that leads to a separated placement but cannot be extended arbitrarily without collisions. To prove NP-hardness of separability using finite translations we just have to modify a detail in the construction of our above proof: the pegs that connect T with $R1$ respectively $R2$ must be made long enough to guarantee that these parts remain non-separated at least until the sticks arrive at the level of the gaps in the roof.

The hardness results still hold when all parts are required to have a constant number of vertices. In our reduction, part $R1$ requires $\Theta(k)$ vertices to model the hooks that push up the lifts. We can replace $R1$ with k parts having a constant number of vertices each. Fig. 2.10 shows the corresponding section of the assembly. All other parts in the proof already have a constant number of vertices.

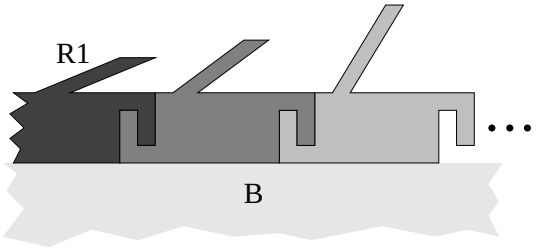


Figure 2.10: Modeling a chain of hooks with parts having constant number of vertices.

Note that parts having a constant number of vertices are not necessarily parts of constant complexity. In our view, the part complexity depends on the number of bits needed for a minimal representation of the part. In the case of polygons, a minimal representation is a list of vertices. In the construction in the proof, the space requirement for each vertex depends on the input size of a given instance of *PARTITION*. Thus the reduction does not generalize to parts of constant complexity.

We summarize the above arguments in the following

Theorem 2 *The problem of finding a (possibly) finite collision-free m-handed translation that separates a given placement of polygons, each having a constant number of vertices, is NP-complete.*

Proof: The proof of NP-hardness follows from the above arguments. The problem is in NP since we can guess a final placement and a straight line and then verify in deterministic polynomial-time that (1) the line partitions the placement and (2) the simultaneous translations required to reach the placement are collision-free. □

2.2.7 Polyhedral Parts

The linear constraint framework in Sections 2.2.1–2.2.5 can be extended directly to the spatial case by introducing a z-coordinate for each part. However, the actual construction of the sets of directions separating each pair of parts, $separate(P_i, P_j)$ is more difficult than in the polygonal case. In [65], pairwise separating directions are computed using a central projection from the origin onto the unit sphere or two parallel planes ($z = -1$, $z = 1$). Our constraints can then be derived from a planar arrangement since all bounding planes of the cones of directions must contain the origin.

In the polygonal case, we were able to develop efficient algorithms for assemblies with no initially separated pairs of parts because the pairwise constraints for non-separated pairs were convex. However, this does not hold for spatial pairs: the set of separating directions can partition into several convex sets even for non-separated polyhedra (e.g. a part with several connected cavities that contain another part, see Fig. 2.11). In principle, the heuristic preprocessing and the branch-and-bound algorithm can be generalized to the polyhedral case. The evaluation tree of the constraint expression will become n-ary, since the separating directions for a pair of parts may require more than two convex sets to represent them.

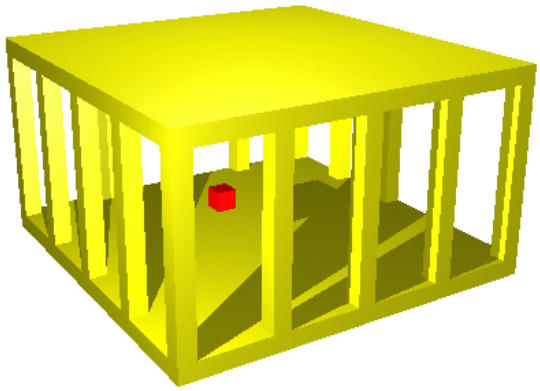


Figure 2.11: A box in a cage: the parts are not separated but the set of separating directions is not convex.

An interesting general difference between polygonal and polyhedral assemblies is that, unlike assemblies of convex polygons, assemblies of convex polyhedra cannot always be assembled with a single hand [66]. Therefore for polyhedral assemblies it is not sufficient that all pairs of parts be separated in order to allow for single-handed separability. In contrast, as shown above, m-handed separability is always possible for pairwise separated parts in any dimension.

Showing that m-handed separability of polyhedra is in NP is not difficult after the above discussion and works similarly to the polygonal case. To show NP-hardness, we perform the reduction from *PARTITION* to m-handed separability using the part shapes from the proof for the polygonal case. We extrude the parts to obtain polyhedra and fix additional large top and bottom cover plates to one of the parts. These covering plates will restrict all motions to a single plane, thus allowing the same reduction as above. We therefore have

Theorem 3 *M-handed translational separability of polyhedra is NP-complete.*

2.3 C-handed Separability

While m-handed separability is NP-complete (Sections 2.2.6 and 2.2.7), single-handed separability is in P [65]. In the m-handed case, computing a separating translation amounts to analyzing an arrangement of half-spaces in a space of general dimension. The complexity of arrangement computation is known to depend exponentially on the dimension of the underlying space [13]. Now the natural question arises whether one-shot separability is in P if one allows more than one but only a constant number of different subset velocities, a problem in between the above extremes. We therefore define the c-handed separability problem:

Definition 3 (C-handed separability) *Given an overlap-free placement of parts, find a one-shot translation that allows for removing at least one part without collisions. At most c subsets of parts may*

translate in different directions with different but constant velocities. Within a moving subset, parts must not move relatively to each other.

Assuming a constant number of moving subassemblies, the dimension of the underlying parameter space becomes constant, and an arrangement which partitions this space into a polynomial number of critical cells can be computed in polynomial time.

2.3.1 Partitioning Subassembly Velocity Space

For polygonal parts, the space E^{2c}, where c is a constant, will be called the *subassembly velocity space*. A vector $(w_{1x}, w_{1y}, \ldots, w_{cx}, w_{cy})$ in E^{2c} states that the parts in the i-th subassembly move with velocity (w_{ix}, w_{iy}). (For polyhedral parts, these concepts can be extended directly.)

Note that the subassembly velocity vector does not contain any information about which part belongs to which subassembly. Although we can partition any subassembly velocity space of constant dimension into a polynomial number of critical cells by a direct extension of the concepts in [65], a hard problem remains: we must assign the subassembly velocities within a given cell to the parts, so that collision-free simultaneous translations result.

Formally, the velocity assignment problem is defined as follows:

Definition 4 (Velocity assignment) *Given an overlap-free placement of parts and a set of velocity vectors, assign the velocities to the parts so that during a one-shot translation with the respective velocities, no pair of parts collides and at least two parts move relatively to each other.*

Note that in general, velocity assignment subsumes the problem of direction assignment. When velocity vectors can be assigned to parts, their directions are clear, but not vice versa. In the single-handed case ($c = 1$), direction and velocity assignment are equivalent problems and can be solved efficiently: assigning parts to the moving, respectively stationary subset of parts can be reduced to searching strong components in a directed graph. This graph describes the invariable blocking relations between all parts for all directions within a critical cell [1, 65]. We will show next that this approach cannot be generalized to more than one moving subset.

2.3.2 Two-handed Separability is NP-complete

We prove that the problem of finding a two-handed separating translation is NP-hard. The result holds even if the set of allowed translational velocities consists only of two given orthogonal velocities with equal magnitude. As a direct consequence, we will obtain hardness results for the two-handed direction and velocity assignment problems. In the case of polygonal parts, our proof requires fixing some of the parts. This requirement will not be necessary for polyhedral parts, thus extending the general hardness result from our discussion of m-handed separability to the case of two-handed separability.

Lemma 3 *Two-handed separability of polygons with fixture constraints is NP-hard.*

Proof: By reduction from 3-SAT [16]. For a logic expression E in 3-CNF, we construct an assembly consisting of polygons and several fixels (polygons constrained to remain stationary). Some parts will be surrounded by fixels so that they can translate in a single direction, horizontally to the right or vertically downwards. Others will be less constrained but can translate either to the right or downwards. It will be possible to remove one or more parts with a single simultaneous one-shot translation if and only if there is a satisfying truth assignment for the variables in E. Furthermore, the construction will require that if one or more parts can be removed, their speeds have to be equal.

We now describe the details. Fig. 2.12.a shows the parts required to represent a variable and its negation. Fixels are shown in black. Either X or $\neg X$ can move to the right but not both. If X moves out, the additional part moves down with the same rate and blocks $\neg X$. The shaded arrow heads in the figure indicate the strips swept by the parts when moving out. Shaded arrow tails on the left indicate where incoming parts can arrive if the corresponding variable part clears their way.

Next, we consider an OR gate with three inputs (Fig. 2.12.b). The logical state of the gate is regarded true exactly if part O can move downwards to infinity. Therefore, it is necessary that at least one of the three input parts moves downwards to infinity. In the case of stationary $I1$, the gate must expel some auxiliary parts to the right while O moves out. Shaded arrows indicate strips swept by incoming or possibly outgoing parts.

To connect the variables to the OR gates we need additional redirection mechanisms which allow translation down a specific strip if and only if translation to the right along another horizontal trip is possible. Fig. 2.12.c shows such a construction. O can translate down if and only if I moves to the right, and O cannot move faster than I does. The arrow tail at the left side indicates that multiple such elements can be connected to a single variable in case the variable is contained in several disjunctive clauses. To connect the construction to an OR gate, we place it below the desired input part of the OR gate. Note that output and input parts need to be aligned horizontally, but there can be a vertical clearance between them: the input part of the OR gate will be able to move out if and only if the output part of the transmission moves down. Moreover, the OR input part will not be able to move out faster than the transmission output part since we consider infinite translations.

Three minor technical details: (1) the horizontal spacing between the inputs of the OR gate, as shown in the figure, is not sufficient for placing the redirection mechanism below it. Either this spacing must be properly enlarged or the input part of the transmission must be shortened. (2) Placing the transmission below the left input of the OR gate requires slight modification of the transmission as shown in Fig. 2.12.d. (3) Inputs of an OR gate that are not to be connected to a transmission have to be blocked by an additional fixel.

Finally, we need an AND gate for representing the conjunction of the outputs from all OR gates. This is shown in Fig. 2.12.e. The AND bar can move to the right if and only if all of the input parts move downwards. Furthermore, its rate cannot exceed the minimum rate of the input parts. The figure suggests that there is a latch at the right end of the AND bar. This role of this latch will become clear

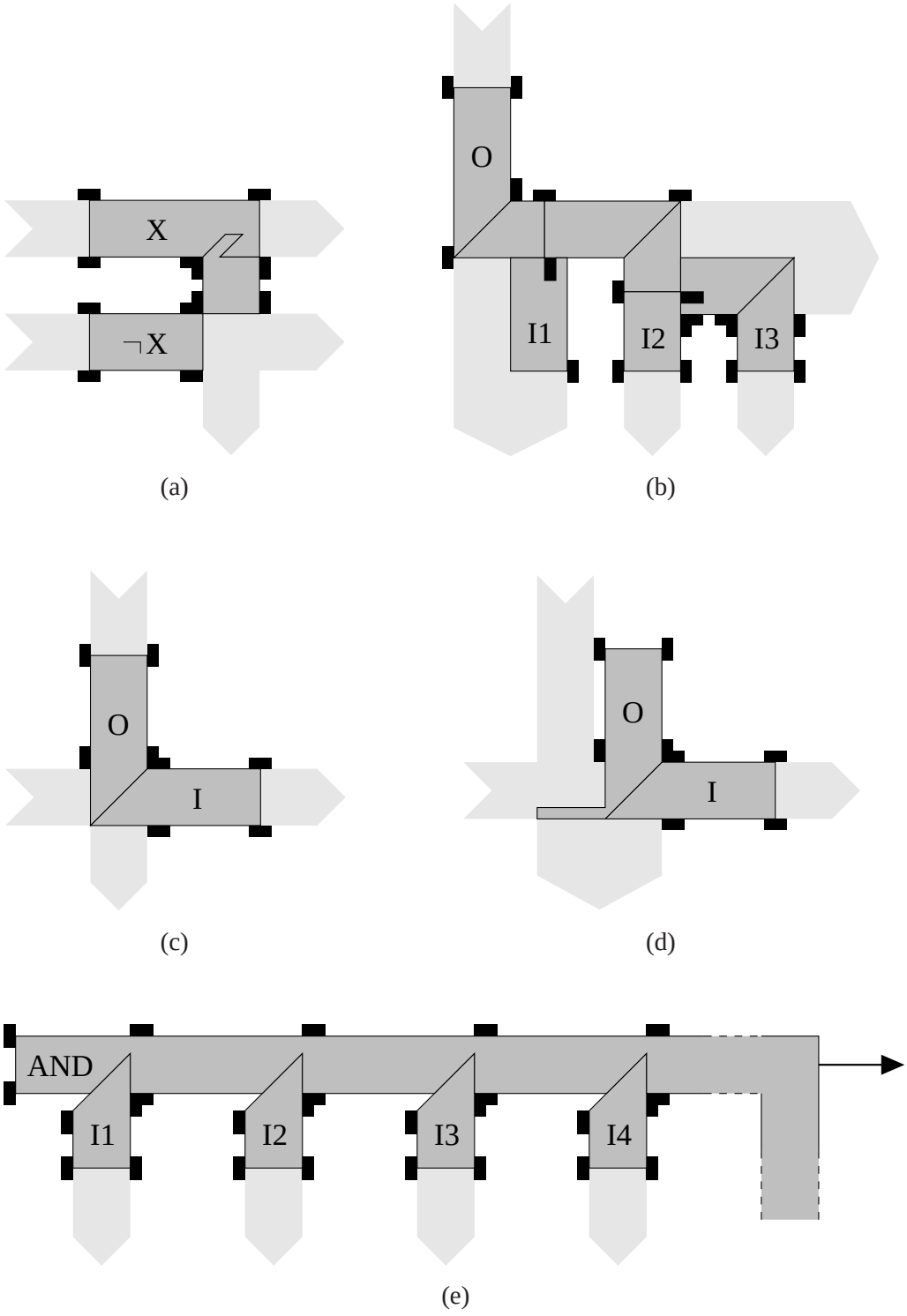


Figure 2.12: Basic elements (fixels shown black): (a) variable mechanism, (b) OR gate, (c) vertical to horizontal redirection (L), (d) special vertical to horizontal redirection (L), (e) AND bar.

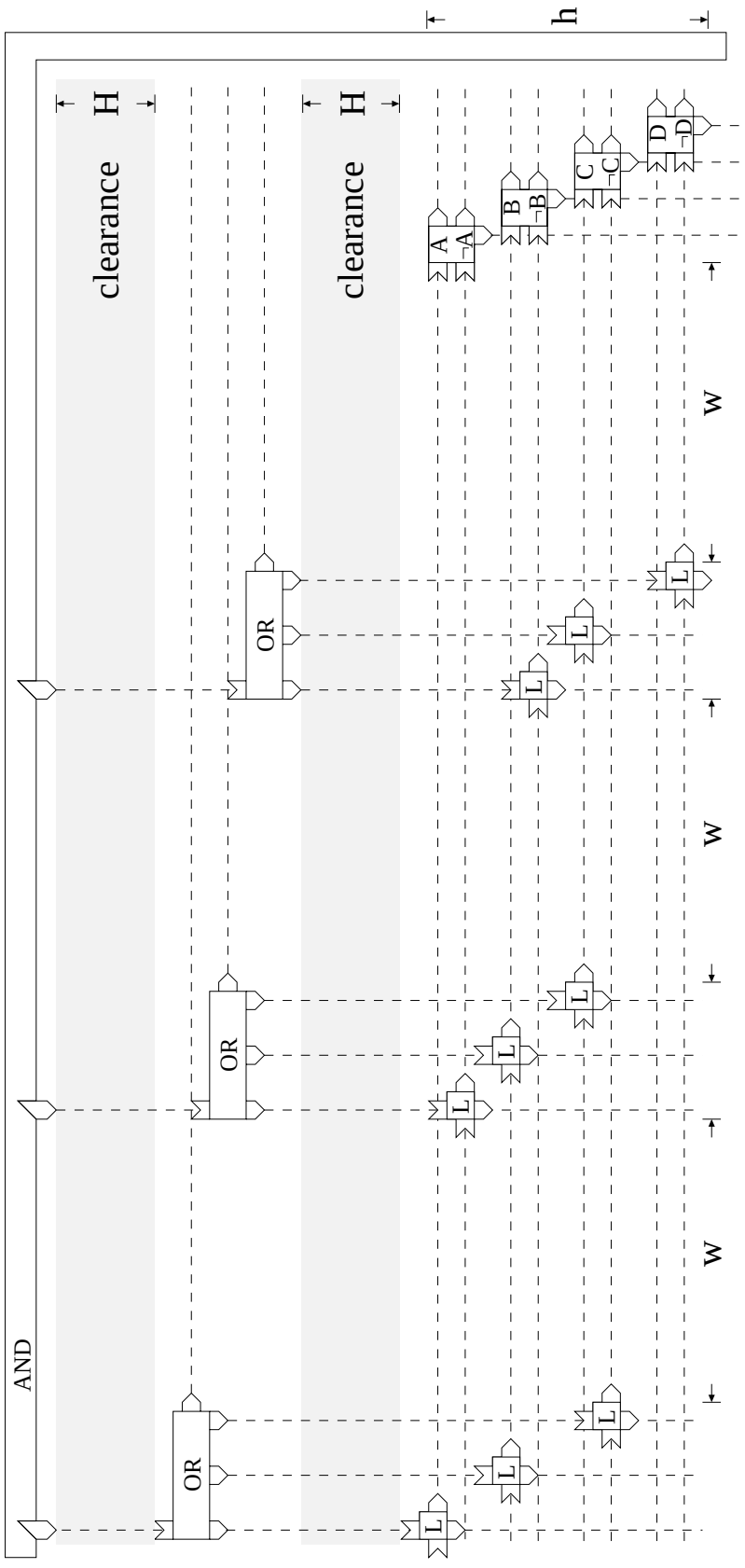


Figure 2.13: Layout for $(A \vee B \vee \neg C) \wedge (\neg A \vee B \vee C) \wedge (\neg B \vee C \vee \neg D)$.

from the global context of the construction which we describe now.

For illustration purposes, we use the example expression $(A \vee B \vee \neg C) \wedge (\neg A \vee B \vee C) \wedge (\neg B \vee C \vee \neg D)$. Fig. 2.13 shows the construction. There the mechanisms from Fig. 2.12 are illustrated only as black box devices with arrows indicating their interfaces. Note that local details of spacing and scaling as discussed above are neglected in the following. Instead, we focus on the major issues: if the expression has a satisfying truth assignment, then there is an infinite two-handed removal translation during which no pair of parts will collide. Conversely, if the expression cannot be made true by any variable assignment, there is no collision-free one-shot removal translation (not even an m-handed one).

We first show that there is a cyclic dependency on the speeds which requires all moving parts to move with the same rate: variable parts cannot move faster than the AND latch, which cannot move faster than its input parts, which in turn cannot move faster than the output parts of the OR gates. This relation on the speeds propagates through the OR input parts, the transmission output parts and the transmission input parts back to the variable parts. Consequently, all moving parts must move with the same rate. Moreover, no parts can move at all unless the AND bar moves, since it blocks all variables, which in turn block all other parts. On the other hand, the AND bar can be moved if a satisfying truth assignment is used to select the moving variable parts. In this case, the corresponding transmissions open, allowing for the OR gates to unlock their output parts. The latter, when moving, clear the way for the AND inputs which unlock the AND bar.

It remains to show that the parts can be placed such that during such a simultaneous translation, no undesired collisions occur. Specifically, the crossings of horizontal and vertical paths must be observed.

We begin with the variable section, and place it at the lower right corner with small constant vertical clearances between the boxes. Small constant horizontal displacements of adjacent variable boxes allow for the auxiliary parts in the variable boxes to escape downwards, if necessary. Dashed horizontal rays on the right indicate where the variable parts move out while dashed rays on the left indicate slim strips blocked by the corresponding variables. We assume that in general the input expression consists of n disjunctions and k variables. Thus $k = O(n)$ and the vertical dimension of the variable section, denoted h in the figure, is in $O(n)$.

For each disjunction, we establish a set of three transmissions on the rays emanating from the involved variables or negated variables. The output parts of these triplets have to be aligned properly for placing OR gates above them. Without loss of generality we may assume that within a triplet, each transmission is to the right of the transmissions above it, thus avoiding collisions among parts from the transmissions of a single triplet. To avoid collisions of horizontally moving parts with downward moving parts from different transmissions, we set the distance between adjacent triplets to $w > h$ with $w = O(n)$. This also applies to the horizontal spacing between the rightmost triplet and the variable section, for the same reason.

We can now dimension the vertical clearance H, as shown in the figure, between the OR section and the assignment grid. We choose H larger than the total width of the whole construction. Note that still

H can be chosen such that it is in $O(n^2)$. The same clearance will be inserted between the AND bar and the topmost OR gate. To be able to expel their auxiliary parts, adjacent OR gates need a constant vertical displacement just like the variable boxes, and so the total height of the OR section is in $O(n)$. Put together, the overall height of the construction will be in $O(n^2)$. Since all moving parts must move with the same rate, the horizontally moving auxiliary parts from the OR section will have passed before the downwards coming input parts of the AND gate have arrived at their level. The same argument applies, at the next lower level, to the horizontally moving input parts of the transmissions and the vertically downward moving output parts from the OR gates.

To summarize, we have shown that no undesired collisions can occur, and parts can be removed to infinity by a two-handed one-shot translation if and only if the given expression allows for a satisfying truth assignment. The construction requires a total of $O(n)$ parts and fixels. All vertex coordinates are in $O(n^2)$ which require $O(\log n)$ storage ($n > 1$). The transformation therefore increases in the input size only by a polynomial function, and can be computed in deterministic polynomial time. □

Since it is a priori guaranteed that only two unique velocities are possible (up to scaling), we obtain the following

Corollary 1 *Two-handed direction/velocity assignment for polygons with fixture constraints is NP-hard.*

The construction in the above proof bears a little resemblance to the one used in [50] showing that translational mobility of polygons is NP-hard. However, there the allowed motions were restricted to infinitesimal translations which makes the construction more predictable. Instead, we consider infinite translations here, and therefore we need additional fixels. It is an open question, whether the same results can be obtained for polygons without fixture constraints. An important difference from our construction is that in [50], more than two directions of motion are required.

Lemma 4 *C-handed translational separability of polygons (polyhedra) with or without fixture constraints is in NP.*

Proof: A non-deterministic algorithm guesses a removal velocity vector $u \in E^{2k}$ ($u \in E^{3k}$) and verifies that (a) the number of moving subassemblies does not exceed c, (b) the fixture constraints (if present) are fulfilled and (c) the motion is collision-free. The latter can be done using the concepts discussed in Section 2.2. □

We therefore have the following result:

Theorem 4 *Two-handed translational separability of polygons with fixture constraints is NP-complete.*

The proof of NP-hardness can be generalized to polyhedral parts similarly to the m-handed case in Section 2.2.7. We excise all parts from a thin plate and add large bottom and top covering plates to the

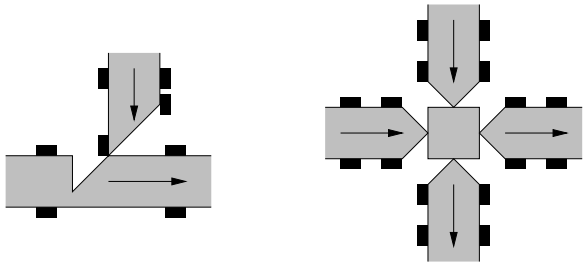


Figure 2.14: Connection of variable to OR gate (left) and independent crossing (right) for small finite translations.

assembly which restrict all possible motions to the plane. We connect the covering plates to all the fixels. Indeed, there is no need of explicit fixture constraints here, since the fixels can be modeled as features of the bottom or top plate. Thus, in the case of polyhedral parts, the proved statement becomes stronger.

Similar to the m-handed case, a variation of the problem is to find *finite* simultaneous translations that can be executed at least until a straight line (a plane) partitions the set of parts into two subsets. The problem remains NP-hard. To prove this for polygons, three directions of motion ($c = 3$) are necessary, while for polyhedra, the result can be shown to hold for $c = 2$.

Following is a sketch of the proofs. The basic idea is to prolongate the AND bar so that no straight line (plane) can be put between it and any other part in the initial placement. We then add a tiny part that can only move downwards out of the AND bar's bounding box while the AND bar moves a very small distance to the right. As soon as the tiny part has moved out, we have two separated subsets and can stop moving. The problem is now to reduce 3-SAT to testing whether there is a small translational motion of the AND bar to the right. This has been addressed in [50], but there, more than two directions of motion were required.

We therefore use the construction from our above reduction and close the gaps between parts that have to influence each other. That is, the AND input parts will now touch their associated OR output parts, the OR input parts will be connected to the transmissions, and so forth. We replace the transmissions by copies of the mechanism shown on the left of Fig. 2.14. The parts must be extended at the ends left open in the figure to reach the parts they are logically connected with. We additionally need crossings of vertical and horizontal lines that are independent for small translations (similar to those in [50]; cf. right picture in Fig. 2.14). These must be placed where no transmissions (L's) were necessary in the proof for infinite translations. Note that they require an additional direction of motion to the lower right for the central square. For polyhedral parts, we do not need this latter mechanism because we can model the independent crossing by exploiting the third dimension. Now, in the final construction for a given 3-CNF expression, not even an infinite translation will be collision-free unless the satisfying variable parts are selected to move.

We finally summarize the above discussion in the following

Theorem 5 *Two-handed separability of polyhedra is NP-complete for both finite and infinite translations.*

Theorem 5 should be considered the most important result in this section. In particular, it shows that it is very unlikely that the algorithm in [65] can be extended to a deterministic polynomial-time algorithm if more than a single moving subassembly is allowed.

Chapter 3

General Translational Separability

We discuss the problem of general translational separability for given overlap-free placements of polygonal or polyhedral parts. The class of considered motions comprises arbitrary sequences of translations that allow for removing at least a single part from the remaining parts without causing overlap between any parts. It is a known fact that even subclasses of this problem are PSPACE-hard. However, such characterizations concern the worst case, and many instances of the problem are much simpler to solve.

We therefore propose a framework that is as general as possible but allows for specifying algorithms that are of practical value. Our primary goal is to show that a uniform approach can be used to handle a variety of problem instances with different properties and not to derive intuitive special-case heuristics. Separability of tightly packed placements with many parts is of special practical importance. Such placements typically arise in the assembly planning domain, but also in other applications. Although in recent years, random planning schemes have been successfully applied to various other motion planning problems, there is generally small likelihood for success of a randomized approach when feasible motions are very constrained. This is common in assembly applications where clearances between parts are small or virtually non-existent. Furthermore, random planners can neither prove infeasibility nor determine that the problem could be solved with small bounded overlap, and identify the inevitable collisions.

The algorithms presented in this chapter are capable of finding separating translations for several tens of parts with or without clearances between them. They are exact and complete. That is, if no part can be removed, infeasibility can be proved (theoretically always and in practice in many relevant cases). It is possible to extend our concepts to find nearly overlap-free motions and identify the critical contacts involved in the inevitable collisions, which may be of great importance in applications.

In general, the running times of our algorithms are dominated only by the complexity of the set of translations that are *feasible* for the particular problem instance. In other words, instances that permit only straight-forward motions will be solved quickly while more complicated ones will take longer. The algorithms have been implemented and tested with a variety of 2- and 3-dimensional examples; observed running times are acceptable if the problems are not inherently combinatorial. Experimental results are summarized in Chapter 6.

This chapter is organized as follows. In Section 3.1, we define the general translational separability

problem and summarize known complexity results. In Section 3.2, we formalize the problem by extending the well-known configuration space concept to an arbitrary number of translating parts. We show that we can combine the configuration spaces derived from pairs of parts into a 'composite' configuration space (c-space) for all parts. Valid and forbidden simultaneous placements of all parts correspond to points in this space. We discuss various combinatorial problems resulting from the general dimension of the composite configuration space. In Section 3.3, we summarize our basic algorithm from [64] which is a first step towards deciding general translational separability in simple practical cases. To determine whether parts are removable, it suffices to search a single connected component, namely the component containing the initial placement. We derive an implicit convex cell decomposition as a discrete representation of this space. Then, a sequence of valid cells connecting the initial cell to an unbounded cell directly yields a feasible plan for removing one or more parts. We call this technique of incremental search of an implicitly given representation *configuration space expansion.* In Section 3.4, we introduce *D-node reduction* to significantly improve the decomposition obtained with our basic algorithm. We show that the concept of reduced D-nodes can yield exponential speedup and discuss how it can be implemented and integrated into our algorithm. In Section 3.5, we discuss remaining problems and outline further straight-forward improvements. We show that practically important extensions such as allowing for small bounded overlap between all or certain pairs of parts or identification of inevitable collisions can be integrated directly.

Note that the concepts in this chapter are presented in a very general and basic framework to avoid elementary assumptions that are application-specific rather than problem-specific. The main algorithmic foundations are well-known techniques from computational geometry and linear optimization.

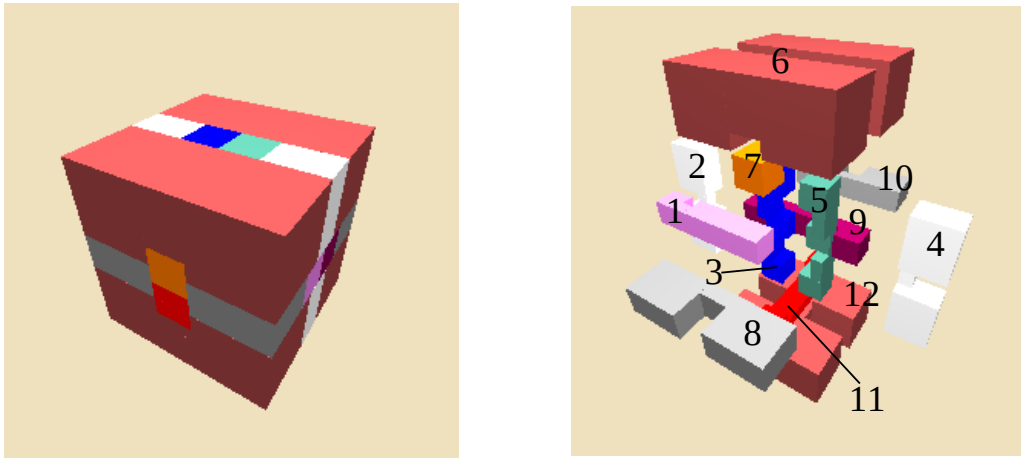
3.1 Problem Definition and Computational Complexity

The general translational separability problem is stated as follows:

Definition 5 (General translational separability) *Given an overlap-free placement of polygonal or polyhedral parts, decide whether one or more parts can be removed without collisions from the remaining set of parts by an arbitrary sequence of translations.*

Note that we do not require an optimal solution, which could be a shortest sequence with respect to some proper cost function. That is, we allow for executing any number of translations in a sequence, for changing the subset of moving parts, and for parts simultaneously moving in different directions with different velocities. General translational separability thus includes non-monotone sequences of motions, in which one or more subsets of parts have to be brought in intermediate placements. We refer to the above definition as the *general problem*.

For an example, consider the wooden cube puzzle in Fig. 3.1.a that is composed of twelve pieces. The exploded view in Fig. 3.1.b reveals the individual parts. Note that due to the tight packing, feasible



(a) (b)

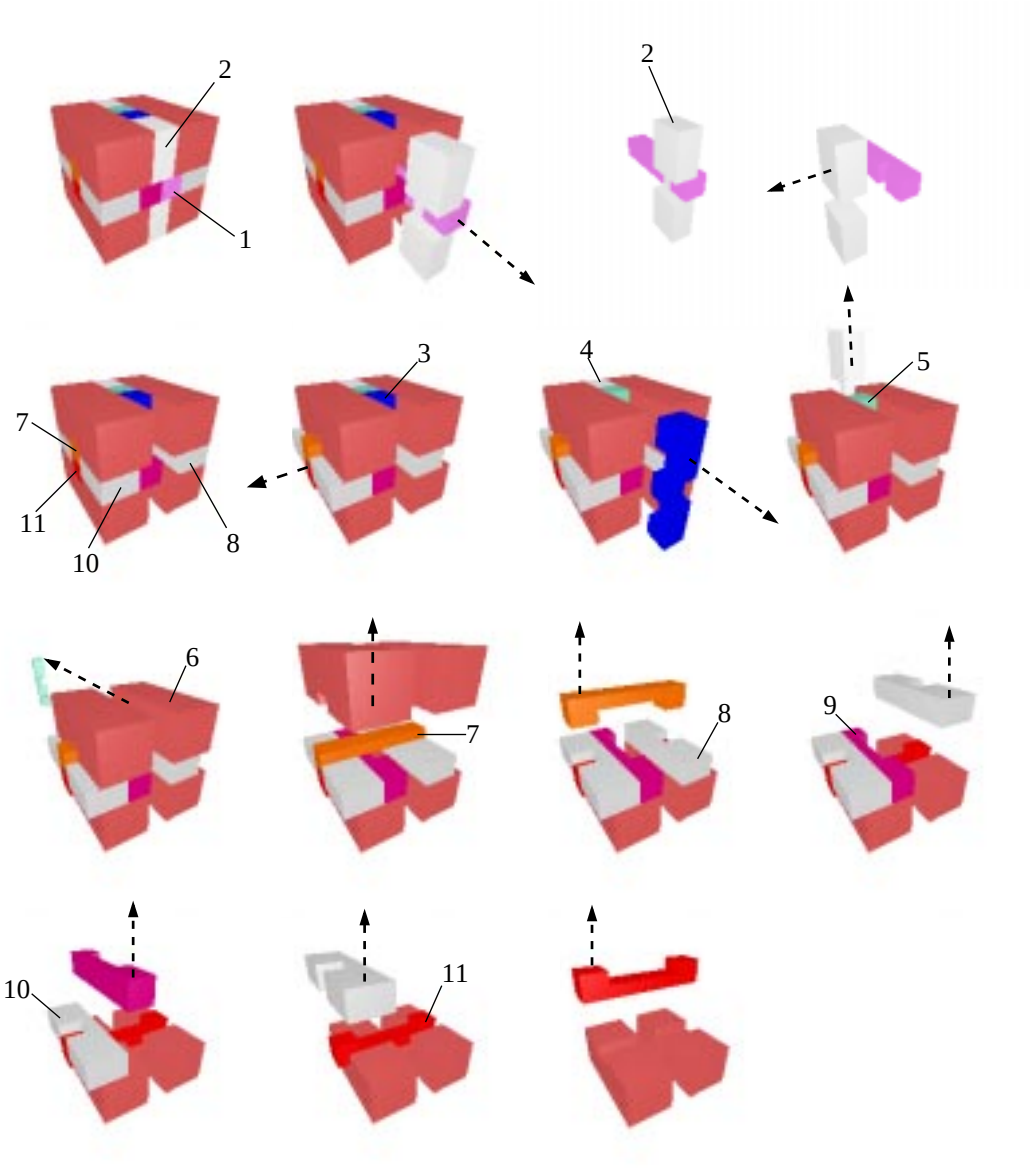


Figure 3.1: Top: (a) Wooden cube puzzle with twelve parts requiring a non-monotone assembly sequence. (b) Exploded view showing the individual parts. Bottom: complete disassembly sequence (parts that move between two pictures in the sequence are referenced).

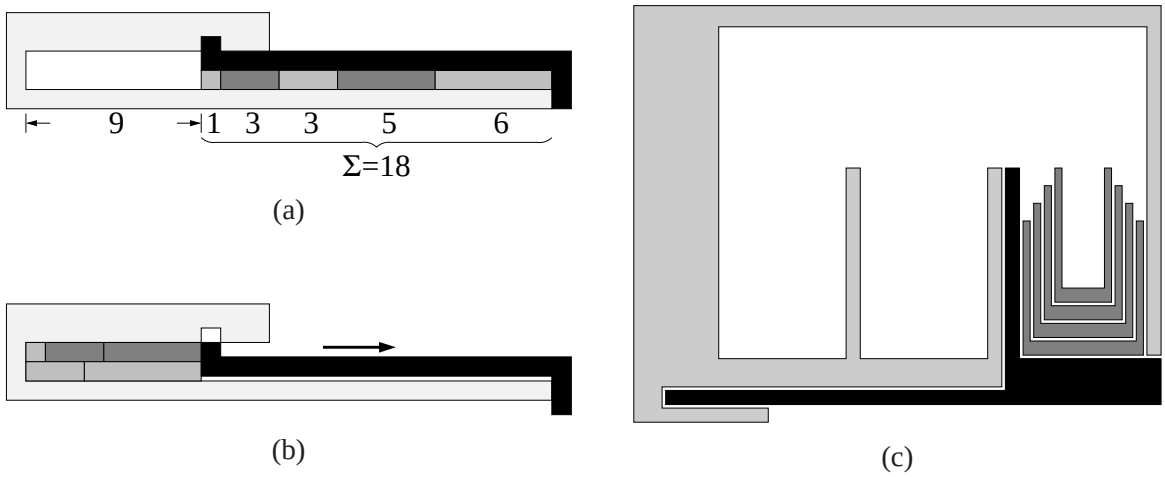


Figure 3.2: Constructions in [49] for reducing (a),(b) *PARTITION* and (c) *Towers of Hanoi* to general translational separability.

motions are very constrained. Fig. 3.1 also shows a complete non-monotone disassembly sequence. The sequence is non-monotone because some parts must be brought into an intermediate position to allow for the removal of other parts.

Computational Complexity

The above formulation of general translational separability is very permissive. Hence, it is not surprising that this problem has been shown to be PSPACE-hard [73, 47]. Earlier complexity results are based on particularly simple and intuitive constructions: NP-hardness can be easily proved by a reduction from *PARTITION* for integers [6]. Let $\sum$ be the sum of the input numbers. Fig. 3.2.a shows the construction for the numbers $1, 3, 3, 5, 6$ that are represented by blocks of corresponding lengths (we use equivalent figures from [49]). The black bolt can removed iff the blocks are tightly packed into the space on the left, thus occupying two rows of length $\sum/2$ each. Fig. 3.2.b shows such a key placement. Now consider using blocks for the numbers $2, 2, 2, 6, 6$ instead. The sum of their lengths is still the same as before, but it will no longer be possible to remove the bolt because the new set cannot be partitioned. Note that in these constructions, it is sufficient that a single part moves at each step. An exponential lower bound on the number of moves in the solution has been derived using a construction that simulates the well-known *Towers of Hanoi* problem [6]. Fig. 3.2.c shows the construction principle as depicted in [49]. It is possible to violate the rules twice per column, but in general, the number of moves required to remove the black bolt is exponential in the number of parts. These hardness results hold for very simple parts; in both reductions, each polygon has a constant number of vertices, and all segments are either vertical or horizontal.

Several simplifications of the general problem have been analyzed with respect to their worst-case computational complexity. Limiting both the number of translations in the sequence and the number

of moving subsets to a single one (single-handed one-shot separability) permits a polynomial-time algorithm [65]. In Chapter 2, we showed that allowing more than a single moving subset will make the problem NP-hard even in the case of one-shot translations. When an arbitrary number of translations is allowed, it is NP-hard even to identify a subset of parts that can be removed by a sequence of single-handed translations [34]. The result holds regardless of whether the subset of moving parts is required to be connected (touching parts) or not. In [23], it is theoretically shown that limiting the number of translations in the sequence to a small constant allows for a polynomial-time single-handed algorithm. However, this algorithm is rather complex, has not been implemented and its practicality has not been demonstrated.

Although our above general problem is hard in the worst case, many practical instances of it are much simpler because their initial configuration is either very constrained or very unconstrained. In the following, we will exploit this fact to derive exact yet practically applicable algorithms. For instance, in the initial placement in Fig. 3.1.a, the parts are tightly packed, which limits the set of possible translations considerably.

3.2 Composite Configuration Space

We extend the configuration space concept from pairs of parts to an arbitrary number of independently translating parts. A first formal description is derived to explain inherent combinatorial problems and motivate our approach in the next section.

Let $P_1, \ldots, P_k$ be an overlap-free placement of polygons. (The discussion in the entire chapter equally applies to polyhedra, the necessary extensions being quite obvious or mentioned otherwise.) We allow for all parts to translate independently, thus the position of each polygon is given by two parameters $\mathbf{p}_i = (x_i, y_i)$. We assume $\mathbf{p}_i = (0, 0)$ for the initial placement, so $\mathbf{p}_i \neq (0, 0)$ describes the location of polygon P_i after a translational displacement. In the motion planning literature, $\mathbf{p}_i$ is commonly called the *reference point*. Note that its relative position with respect to P_i is completely arbitrary as long as it does not change. A space E^d of dimension $d = 2k$ describes simultaneous placements of all parts. E^d will be called the *composite (assembly) configuration space*. A point $(x_1, y_1, \ldots, x_k, y_k)$ in this space is forbidden if two or more parts intersect in their interiors in the corresponding placement. In this work, we generally assume that parts are open sets and allow for overlapping part boundaries. The origin in E^d represents the initial placement of all parts and is not forbidden.

To obtain a formal representation of the composite configuration space, we use a similar approach as in the case of one-shot translations (cf. Chapter 2): first derive valid relative placements for independent pairs of parts, and then embed and combine them in the composite configuration space. Thus, an exact and complete *convex cell decomposition* of the composite configuration space is made up of the constraints from all pairwise configuration spaces. We will explain this representation, inherent problems and ways to address them in detail in the following.

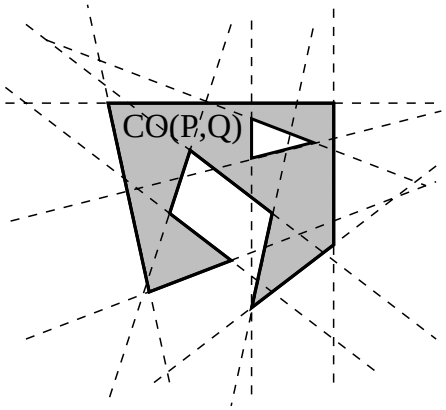


Figure 3.3: Decomposition of a two-dimensional configuration space by an arrangement of straight lines.

In robotics and computational geometry it is well-known that the set of forbidden relative placements of a polygon P_i translating with respect to a fixed polygon P_j, is also a polygon (possibly with holes): the so-called *configuration obstacle* $CO(P_i, P_j)$ [39]. (Several algorithms have been proposed in the literature to compute it, cf. Chapter 5.) In our case, only the interior of $CO(P_i, P_j)$ is actually forbidden, since parts are allowed to touch at their boundaries. Formally, $CO(P_i, P_j)$ equals the Minkowski difference of P_j and P_i, that is

$$CO(P_i, P_j) = P_j \ominus P_i = \{\mathbf{p}_j - \mathbf{p}_i \mid \mathbf{p}_i \in P_i, \mathbf{p}_j \in P_j\}$$

It is important to note that this set is 2-dimensional, because it describes forbidden positions of P_i's reference point in a coordinate frame centered at P_j's reference point. We call the set of relative placements of P_i with respect to P_j a *pairwise configuration space*. Since the free space around $CO(P_i, P_j)$ may have a complex shape, we decompose it into simpler convex pieces: the boundary edges of $CO(P_i, P_j)$ define an arrangement of straight lines $A(P_i, P_j)$ in the plane (Fig. 3.3). $A(P_i, P_j)$ decomposes the set of allowed placements of P_i with respect to P_j (the complement of the interior of $CO(P_i, P_j)$) into closed convex cells. The cells can be assumed being closed, since only the interior of the configuration obstacle is forbidden.

To resolve the asymmetry in the above definition of $CO(P_i, P_j)$, we next derive a characterization of feasible placements of P_i and P_j in global coordinates of their reference points. We can embed the straight lines from the arrangement $A(P_i, P_j)$ in the composite configuration space E^d where they become hyperplanes: a straight line in $A(P_i, P_j)$ is described by a linear equation $ax_{ij} + by_{ij} = c$, where a, b and c are constants and (x_{ij}, y_{ij}) represents the position of P_i's reference point with respect to a frame rigidly attached to P_j and centered at P_j's reference point. Since both reference points (x_i, y_i) and (x_j, y_j) are described using global coordinates in E^d, we have $(x_{ij}, y_{ij}) = (x_i - x_j, y_i - y_j)$. (cf. Chapter 2, Section 2.2.2 for a similar substitution.) The straight line equation thus becomes a hyperplane $a(x_i - x_j) + b(y_i - y_j) = c$ in the four-dimensional space of all placements of P_i and P_j.

As a consequence, we can embed this hyperplane in E^d by simply regarding the above four-dimensional equation as a d-dimensional equation with at most four non-zero coefficients.

3.2.1 Global Complexity Problem

To obtain a complete description of the composite configuration space, we have to perform the above embedding for all straight lines derived from all pairwise configuration obstacles. The resulting arrangement A^d partitions E^d into closed convex cells of dimensions ranging from 0 to d. These cells are regular, that is, all points within a single cell are either allowed or forbidden. (This follows directly from the regularity of the embedded 2-dimensional cells.) We can thus label a cell in A^d as allowed or forbidden by examining a single point inside. Finding the label for a cell is straight-forward: choose an arbitrary point $\mathbf{p}$ within the cell, project it into all pairwise configuration spaces and test whether the projections are outside (or on the boundary of) the configuration space obstacles for each pair. The projection of $\mathbf{p} = (x_1, y_1, \ldots, x_k, y_k)$ into the configuration space of a part P_i with respect to another part P_j is the point $(x_i - x_j, y_i - y_j)$. The cell in composite space is allowed if and only if no projection is inside the interior of the corresponding pairwise configuration obstacle, which means that no pair of parts intersect in their interiors in the placement $\mathbf{p}$.

In principle, a graph representation of A^d could be used to make the adjacency relations amongst the cells explicit [13]. Such a graph would represent the connectivity of the composite configuration space in an exact and complete way. An exact and complete algorithm for general translational would search this graph for a sequence of adjacent free cells that connects the initial cell to an unbounded cell. Finally, a ray in an unbounded cell proves that one or more parts can be removed. However, a complete graph representation of A^d has an exponential number of nodes, and it is not possible to precompute such a graph even for a small number of simple parts.

Note that motion planning based on *exact cell decomposition* of configuration space has been proposed in the literature for a variety of other configuration space and cell types (cf. [39] for a survey). The most general approach [59] is based on a cylindrical algebraic decomposition known as the *Collins Decomposition* and allows – at least in theory – for considering more general types of part motions, including rotations. However, all these approaches suffer from an exponential dependence on the configuration space dimension and are practical only if the number of independently moving objects is small.

3.2.2 Local Complexity Problems

Besides the exponential number of cells in a convex decomposition of composite configuration space, the general dimension causes other combinatorial obstacles.

It is well-known that the number of vertices of a single cell in an arrangement of hyperplanes can be exponential in the dimension of the space (for instance, consider a unit hyper-box). This problem also arises for cells in our special type of arrangements that are defined by embedded pairwise constraints, as illustrated by the following example.

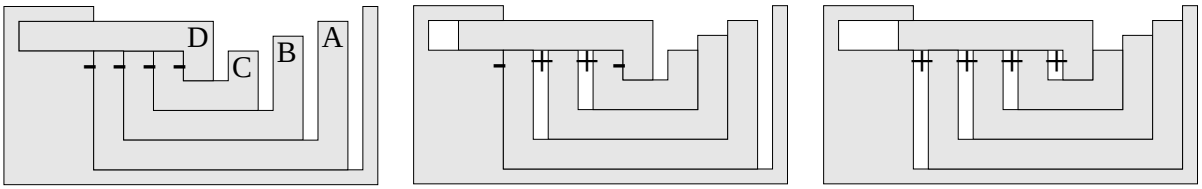


Figure 3.4: Example for single convex configuration space cell with an exponential number of vertices. Sample vertices (a) $(-,-,-,-)$, (b) $(-,+,+,-)$, (c) $(+,+,+,+)$.

The polygons in Fig. 3.4.a cannot be separated, but the latch and the U-shaped parts can move independently to the left or right. A single convex cell defined by the linear constraints $\{0 \leq x_A \leq 1, 0 \leq x_B - x_A \leq 1, 0 \leq x_C - x_B \leq 1, 0 \leq x_D - x_C \leq 1, y_A = 0, y_B = 0, y_C = 0, y_D = 0\}$ represents the subset of free composite configuration space that is reachable via translations. Note that the constraints are bounded by hyperplanes derived from pairwise configuration spaces (The unnamed container is fixed and we therefore eliminated its coordinates from the constraints.) In general, we can place k U-shaped parts in the container (let the container be part 0 and the latch be part $k+1$). Then the vertices of the configuration space cell correspond to certain critical placements, and their number is exponential in k: a maximization of $\sum_{i=1}^{k+1} c_i(x_i - x_{i-1})$ using a specific combination $(c_1, \ldots, c_{k+1})$, $c_i \in \{-1, +1\}$, yields a unique vertex. Three corresponding critical placements are shown in Figs. 3.4.a–c, labelled according to the signs of the c_i in the objective function of the maximization (signs indicate relative placements of adjacent nested parts). This example shows that, in the worst case, the number of vertices of a cell in a composite configuration space can be exponential in the number of parts. Note that the result holds even though the sets of feasible relative placements for each pair of parts have only dimension one.

Inherent Local Configuration Space Complexity

Computing vertices can be avoided by working with a linear constraint representation that always has polynomial size. In the above example, we need only $2(k+1)$ linear equalities and inequalities. However, another local combinatorial problem can obviously not be solved: a configuration may be contained in an exponential number of different cells, and this is independent of a particular convex decomposition.

Lemma 5 *The optimal convex decomposition of an ϵ-ball around a given configuration in a composite translational configuration space of k parts consists of $\Omega(2^k)$ cells in the worst case.*

Proof: We show that the configuration of k boxes in Fig. 3.5.a must be contained in the intersection of at least 2^{k-1} convex configuration space cells. To prove this, we define 2^{k-1} critical placements obtained by independently sliding each pair of adjacent boxes an infinitesimal distance along their horizontal or vertical edge pairs in contact. Figs. 3.5.b,c show two such placements. We can label the critical place-

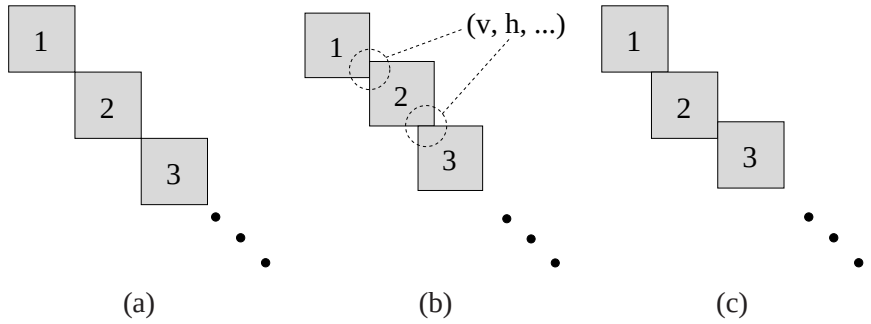


Figure 3.5: (a) Configuration contained in an exponential number of convex configuration space cells. (b),(c) Critical configurations that cannot be in the same convex cells.

ments with tuples of length $k-1$: the i-th entry in a label will be h (respectively v) if boxes i and $i+1$ have been shifted along their horizontal (vertical) edge pair. Thus, the labels for the critical placements in Figs. 3.5.b,c are $(v, h, \ldots)$ and $(h, v, \ldots)$, respectively. In general, two different critical placements cannot be connected in composite configuration space by a straight line segment, because at least one part would have to move around the vertex of another part; so they must be contained in different convex configuration space cells. Since we shift boxes only an infinitesimal distance, all these cells must contain the initial placement in Fig. 3.5.a which is therefore contained in the intersection of an exponential number of cells. □

For three-dimensional parts, the result can be extended. We use polyhedra, each of them being composed of two cones that have been agglutinated together with their bases. The bases are regular n-gons. In the initial placement, the parts will be stacked on top of each other with touching tips of adjacent parts. We therefore have

Corollary 2 *The optimal convex decomposition of an ϵ-ball around a given configuration in a composite translational configuration space of k three-dimensional parts, each having $O(n)$ vertices, consists of $\Omega(n^k)$ cells in the worst case.*

Thus, an optimal local decomposition of the composite configuration space cannot be computed within reasonable time in the general case. (Note that the problem of finding optimal convex decompositions is already NP-hard in two dimensions under certain circumstances [41].) Furthermore, it does not seem to be possible in general to efficiently find a single non-trivial convex cell around a given configuration. In other words, the only thing we can compute efficiently in the general case is a cell that is guaranteed to contain the current placement.

Lemma 6 *The problem of finding a convex cell of dimension greater than zero that contains a given configuration in free composite translational configuration space is NP-hard.*

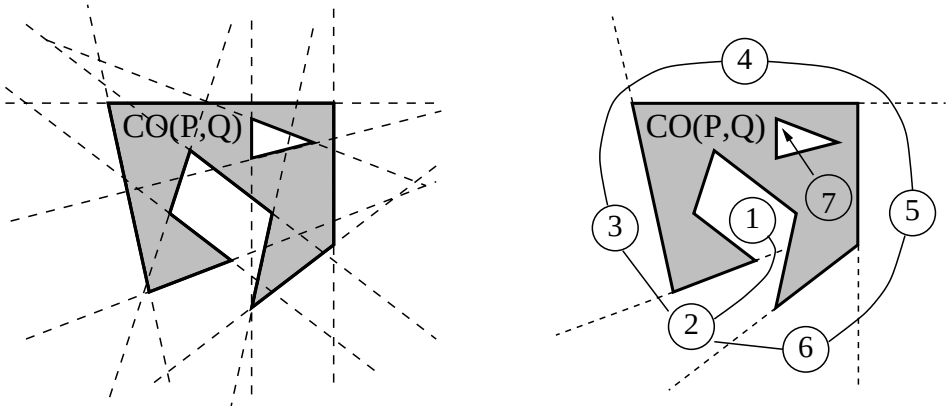
Proof: Palmer [50] proves that testing for arbitrary infinitesimal translational motions, a closely related problem, is NP-hard (by a reduction from 3-SAT). The construction principle is similar to our proof of Lemma 3 in Chapter 2 (p. 31) but works without fixture constraints at the cost of requiring more than two directions of motion. For a given 3-CNF expression E, Palmer's proof specifies a placement of polygonal parts (without overlapping interiors) for which infinitesimal translation is possible if and only if there is a satisfying truth assignment for E. Finding a convex configuration space cell that contains further placements in addition to the given one would allow for an efficient test if translational motion is possible by maximizations along an arbitrary base of the composite configuration space. This proves our claim. □

3.2.3 Towards a Practical Algorithm: C-space Expansion

We showed that the composite configuration space may locally decompose into an exponential number of convex cells (Lemma 5). This inherent worst-case complexity cannot be circumvented by any convex decomposition. The problem of locally finding a non-trivial cell turned out to be NP-hard (Lemma 6). In the light of these results and several related hardness results for convex decompositions in the plane [41, 9], one must consider the problem of finding an optimal convex decomposition of composite configuration space to be intractable. However, our experiments showed that in many relevant cases, we can in fact obtain good decompositions.

The exponential overhead arising from direct global arrangement computation, as discussed in Section 3.2.1, can be drastically reduced in most practical cases with a generic approach. In particular, arrangement pre-computation can be avoided altogether by *implicit representation* and *incremental expansion.*

The next three sections describe our methods to realize these two concepts. First, in Section 3.3 we describe basic data structures to 'unclutter' the decomposition induced by an arrangement A^d of hyperplanes and to avoid computation of complete cell boundaries and non-reachable regions in E^d. We showed above that computing an explicit vertex-face representation is prohibitive even for single cells in composite configuration space, because the number of vertices can be exponential in the number of parts. An implicit linear constraint representation is therefore used instead. This representation allows for efficient incremental expansion of a single connected component containing the initial placement. Section 3.4 will then show how the number of computed cells can be reduced even further by dynamically eliminating redundancies. This additional 'defragmentation' replaces up to an exponential number of cells by their convex union without the need to handle them explicitly. Finally, in section Section 3.5, we present further heuristic strategies to improve on the methods in the preceding sections, point out practically relevant extensions and state open problems for future research.



(a) (b)

Figure 3.6: (a) Decomposition of free two-dimensional configuration space by arrangement of straight lines, (b) decomposition into overlapping convex cells bounded by line segments and rays and superimposed floorgraph.

3.3 C-space Expansion: Floorgraphs and D-graph

Direct and global arrangement computation is not applicable if the dimension is not a small constant. First, an arrangement of straight lines (and, as a consequence, the arrangement of embedded hyperplanes) suffers from excessive cell fragmentation. Second, it is not necessary to compute the entire cell decomposition of E^d. Free space may partition into several connected components, and when testing for removability it suffices to compute only the component that contains the initial placement.

We therefore introduce two new data structures, *floorgraphs* and *D-graphs*. These will allow for an implicit representation and efficient storage of free space and provide the basis for incremental computation of high-dimensional configuration spaces for translational motions.

3.3.1 Floorgraphs

Partitioning pairwise configuration spaces into cells using straight lines introduces excessive cell fragmentation. Since the underlying space has small constant dimension (2 in the case of polygons and 3 for polyhedra), better approaches are possible. Convex decompositions in two and three dimensions have been extensively studied in the literature (cf. [41, 5, 4, 7] for example). While finding optimal decompositions is NP-hard (e.g. for polygons with holes, polyhedra), near-optimal solutions can often be obtained with simple and robust algorithms.

Fig. 3.6 shows an example of a configuration space decomposition for two parts P and Q by straight lines and a better decomposition of the same space using line segments and rays. Note that the second approach yields substantially fewer cells than the simple decomposition with straight lines. For the moment, we disregard whether the decomposition is good with respect to the given placement and

configuration space of all parts. We assume that the pairwise decompositions can be obtained in an independent preprocessing step. (Our experiments show that this approach works well in many cases.)

For a pair of parts, we define a *floorgraph* as a compact representation of a convex decomposition of free configuration space and its connectivity:

Definition 6 (Floorgraph) *A floorgraph for a pair of parts (P, Q) is a graph whose nodes are closed convex cells that cover all placements of the two parts in which their interiors do not intersect. Cells are described by linear constraints on the coordinates of the reference points assigned to the parts. Two nodes are connected by an undirected edge if their associated cells intersect.*

A floorgraph for a pair of polygons (P, Q) can be obtained by decomposing the complement of the set $interior(CO(P, Q))$ into overlapping closed convex 2-dimensional cells bounded by line segments and rays. Note that in Fig. 3.6, cells overlap only at their boundaries. This is what is commonly called a partitioning. In our case, we may allow a more general covering where cells overlap in their interiors as well. (However, covering seems to have greater complexity than partitioning.) Floorgraph node cells are represented by embedded linear equalities and inequalities in the absolute part positions (x_P, y_P) and (x_Q, y_Q), that is, for two-dimensional parts, they have the general form $a_x(x_P - x_Q) + a_y(y_P - y_Q) \geq b$. The floorgraph for the decomposition in Fig. 3.6.b is superimposed in the picture. Here, the constraints associated with node 5 are $\{x_P - x_Q \geq c, y_P - y_Q \leq d\}$, where c and d are constants. Note that because in general, $CO(P, Q) = -CO(Q, P)$, the orientation of the configuration obstacle does depend on the part order (since it is computed in relative coordinates of the first part with respect to the second part) but the floorgraph constraints do *not* (they are embedded, that is, given in absolute part positions).

The advantage of the floorgraph representation is that cells in the composite space of all parts are bounded by surface patches rather than extended hyperplanes. (Although cells themselves are defined as the intersection of half-spaces, each cell has its own set of them and is thus not unnecessarily partitioned by hyperplanes of other cells.) As a consequence, free space is decomposed into considerably fewer cells in comparison to a direct approach using extended hyperplanes.

For three-dimensional parts, the configuration obstacle of two polyhedra is a polyhedron [39] and its complement can be described in a similar fashion as a union of convex cells in 3-space. In practice, computing configuration obstacles and free cells in three dimensions can be a challenging task, especially for nonconvex polyhedra with many vertices. We will discuss this issue in Section 5.1.

3.3.2 D-graph and Basic Incremental Algorithm

Floorgraphs are introduced to manage constraints from pairs of parts. To obtain a description of valid placements in the composite configuration space of all parts, we combine the floorgraphs from all pairs of parts into a new data structure, called *D-graph*. A D-graph is an exact and compact representation of cells and their adjacency relations in the d-dimensional composite configuration space. It is defined implicitly via *D-nodes* and *D-edges*.

Definition 7 (D-node) *Let $G_1, \ldots, G_D$ be floorgraphs for all pairs of parts (i, j), where $1 \leq i < j \leq k$ and $D = k(k-1)/2$. A D-node is a tuple $(n_1, \ldots, n_D)$ when n_l is a node in G_l for $1 \leq l \leq D$. Associated with each D-node is the set of all embedded pairwise linear constraints stemming from the contained floorgraph nodes.*

A D-node represents a convex cell of valid placements of all parts obtained by embedding and intersecting floorgraph node cells from all pairs of parts. Note that most D-nodes represent empty cells, as most combinations of nodes stemming from the pairwise floorgraphs yield infeasible constraint sets. However, for a given valid placement of parts we can compute a D-node that contains this configuration: we project the configuration to each pairwise configuration space and search the corresponding floorgraphs for the cell (node) containing the projection.

The example in Fig. 3.7 illustrates D-node construction from floorgraphs for all pairs of parts. It shows a configuration of three boxes (a), the generic floorgraph for a pair of boxes (b) and the principle for constructing the D-node containing the given configuration (c). For clarity, the floorgraphs and pairwise configuration spaces corresponding to the D-node entries are referenced by arrows in (c). For example, the first entry **e** stems from $Floorgraph(A, B)$ and states that the relative position of A with respect to B is contained in cell **e**. The set of embedded constraints for the complete D-node (**e**, **s**, **w**) is therefore ($a, \ldots, f$ are constants):

$$\{\ \underbrace{x_A - x_B \geq a,\ y_A - y_B \geq b}_{\mathbf{e}}\ ,\ \underbrace{x_A - x_C \geq c,\ y_A - y_C \leq d}_{\mathbf{s}}\ ,\ \underbrace{x_B - x_C \leq e,\ y_B - y_C \leq f}_{\mathbf{w}}\ \}$$

We call two D-nodes *neighbors* if and only if their associated cells have non-empty intersection. To find the neighbors of a D-node, we define a *successor relation* on D-nodes:

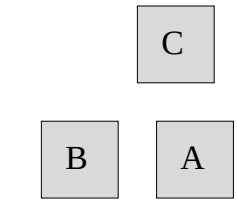
Definition 8 (Successor D-node, door cell, D-edge) *A D-node $S' = (n'_1, \ldots, n'_D)$ is called successor of a non-empty D-node $S = (n_1, \ldots, n_D)$ if and only if*

1. *n_i and n'_i are neighbors in floorgraph i,*
2. *$n_j = n'_j$ for all $j \neq i$, and*
3. *the combined constraints from (S, n'_i) define a non-empty cell, called door cell.*

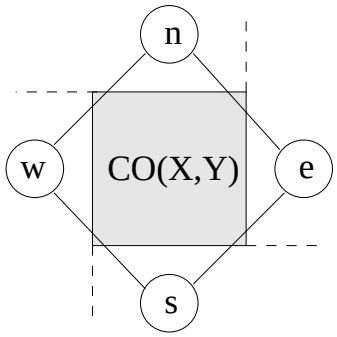
We say that a D-node S and its successor S' are connected by a D-edge.

Basic Incremental Algorithm for General Translational Separability

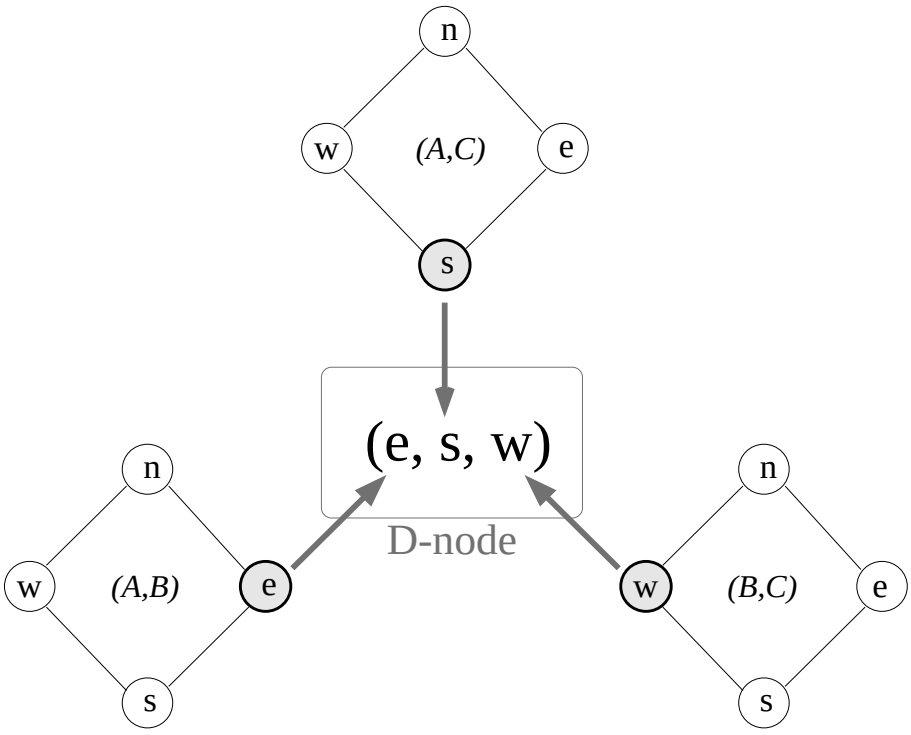
A valid sequence of translations that allows for removing one or more parts can be found by searching the implicitly given D-graph. We begin with constructing a D-node that contains the initial configuration (part placement) and recursively explore successor D-nodes using a standard search algorithm, such as depth-first search. For a given valid D-node, we generate all successor D-nodes by successively



(a) Placement of boxes



(b) Generic configuration obstacle and floorgraph for a pair of boxes (X, Y)



(c) D-node construction

Figure 3.7: Example for D-node construction from pairwise floorgraphs.

exchanging all its entries, one at each step, and verifying condition (3) in Definition 8 with a standard linear feasibility test. Note that these steps avoid computing the complete boundaries of cells in high-dimensional space: instead they only test feasibility of *door cells*, that is, if the convex patches (S, n_i') contribute to the boundary of the D-node cell. In [64] we have proved formally that all neighbors of a feasible D-node can be reached via successor D-nodes, i.e. by switching floorgraph nodes one at a time.

During the search, visited D-nodes are stored to avoid visiting them more than once, and for retrieval of a solution path after the search has terminated successfully. For the latter purpose, a handle to access the predecessor D-node has to be stored together with each visited D-node. Efficient access to visited D-nodes is important and discussed in Chapter 5.

Every new D-node cell is tested for unboundedness immediately after generation. Once an unbounded cell is found, the search stops and the algorithm can output a sequence of neighboring free D-nodes that connects the initial cell to the unbounded cell. For this, we follow the pointers to the predecessor D-nodes back to the initial D-node. Since neighboring cells have non-empty intersection, a removal path for one or more parts (a continuous curve in E^d) can then be 'threaded' through them. In the simplest case, we may choose a sequence of straight line segments supported by points anywhere in the intersections of adjacent cells.

This basic algorithm is summarized in Tab. 3.1. It incorporates two principles: bounding cells by surface patches and incremental computation. In most practical cases, only a very small portion of the entire D-graph will actually have to be constructed. A fast and complete test for unboundedness is important, since many cells have to be tested during the search. Note that all cells except the last will be bounded. It is therefore important that the test can establish boundedness efficiently without missing possible unbounded directions. We present such a test in Chapter 4 that is fast both for bounded and unbounded cells. Problem specific heuristics may be included to guide and speed up the search (cf. Section 3.5). Nevertheless, combining floorgraph nodes from all pairs of parts induces an unnecessary fine cell decomposition. We will focus on this problem and approaches to defragment the decomposition in Section 3.4.

3.3.3 Analysis

We first consider a global arrangement A^d that is derived from extended hyperplanes, as described in Section 3.2.1. The number of cells in this arrangement is a first, although very loose, upper bound on the number of D-nodes visited during the search. For the moment, let us assume polyhedral parts. The faces of each part can be decomposed into triangles. Let n be the maximum number of triangles in each of the k parts. Then n is a bound for the number of vertices of each part. To compute a configuration obstacle (Minkowski-difference) of a pair of parts, it suffices to compute the Minkowski-differences for pairs of triangles on faces. We thus obtain $O(n^2)$ inequalities for each pair of parts. The number of such pairs is $O(k^2)$, so that the total number of half-spaces in A^d is in $O(k^2n^2)$. It is well-known than an arrangement of u hyperplanes in d dimensions has at most $O(u^d)$ cells, including cells of lower

Input: Overlap-free placement of polygons or polyhedra $P_1, \ldots, P_k$.
Output: Sequence of adjacent free cells containing a path that separates the parts.

1. For $1 \leq i < j \leq k$
   - ◇ Compute $Floorgraph(P_i, P_j)$.
   - ◇ Find a floorgraph node n^0_{ij} that contains the initial placement of P_i and P_j.
2. $S^0 \leftarrow (n^0_{1,2}, \ldots, n^0_{k-1,k})$ **co:** construct a D-node for the initial configuration.
3. $L \leftarrow [S^0]$ **co:** initialize a list of D-nodes.
   $V \leftarrow \{S^0\}$ **co:** initialize a set of visited D-nodes.
4. While not $empty(L)$
   - ◇ $S \leftarrow removefirst(L)$
   - ◇ $L' \leftarrow [\ ]$ **co:** initialize an empty temporary list of successors.
   - ◇ For each node n_{ij} in S
     - ◇ For all neighbors n'_{ij} of n_{ij} in $Floorgraph(P_i, P_j)$
       - ◇ $S' \leftarrow S$. Replace n_{ij} in S' by n'_{ij}.
       - ◇ If $S' \notin V$ and $feasible(S, n'_{ij})$ then
         - ◇ Supplement S' with a handle to access S in V.
         - ◇ $V \leftarrow S'$ **co:** mark S' as visited.
         - ◇ If $unbounded(S')$ then output the path from S^0 to S' and stop.
         - ◇ $L' \leftarrow append(L', S')$ **co:** append the new D-node to L'.
   - ◇ $L \leftarrow conc(L', L)$ **co:** attach the new successors at the front of L.
5. Output *infeasible*.

Table 3.1: Basic D-graph search algorithm using a depth-first strategy.

dimension (see e.g. [13]). Thus A^d has at most $O((kn)^{2d})$ cells. Here, $d = 3k$ (polyhedral case).

Due to the PSPACE-hardness of our basic problem, it is clear that every worst-case upper bound on the number of convex cells has to be exponential. However, using floorgraphs, the above bound can be made independent of n, the maximum number of triangles per part. In [64], we derive a bound that depends on the maximum number m of nodes in a single floorgraph. Note that usually, m is much smaller than n.

Lemma 7 ([64]) *The number of node expansions in the algorithm in Tab. 3.1 is bounded by $O(k^{2d}m^d)$, where m is the maximum number of nodes in a connected component of a floorgraph and k is the number of parts.*

Since k and d are related, the above bound on the number of node expansions can be reduced further (in the case of three-dimensional parts):

Lemma 8 ([64]) *There is a constant d_0 such that the number of node expansions in the algorithm in Tab. 3.1 is bounded by $O(k^{f(d)}m^{g(d)})$, where f and g are functions with $f(d) < d$ and $g(d) < d$ for any d larger than d_0 .*

3.4 C-space Expansion: Further Cell Defragmentation

The basic incremental configuration space expansion algorithm in the previous section improves over global arrangement computation in two ways: first, by combining floorgraphs rather than composing an arrangement full hyperplanes, it significantly unclutters the decomposition. Second, by implicit graph representation and incremental computation instead of global graph computation, it expands only reachable configuration space regions and avoids computing complete cell boundaries.

However, combining constraints derived from all pairs of parts still generates an unnecessarily fine decomposition. In this section, we will first illustrate by means of a simple example that unnecessary constraints can cause exponential overhead. In particular, convex regions in free configuration space may be artificially partitioned into an exponential number of fragment cells (individual D-nodes). We then discuss ways to reduce this cell fragmentation by eliminating unnecessary constraints from the D-nodes.

3.4.1 Reduced D-nodes: Improved Decomposition

Consider the example in Fig. 3.8.a. It consists of a container X with four separate slots, numbered 1 to 4, and four boxes, named A, B, C and D. The floorgraphs for any of the four (box, container) pairs are topologically equivalent to the container and consist of five cells each: four cells, labelled 1 to 4, correspond to the slots, and a fifth cell connects them. We may assume that the floorgraphs for each pair of boxes is equivalent to the floorgraph in Fig. 3.8.b and consists of a single component of four connected cells.

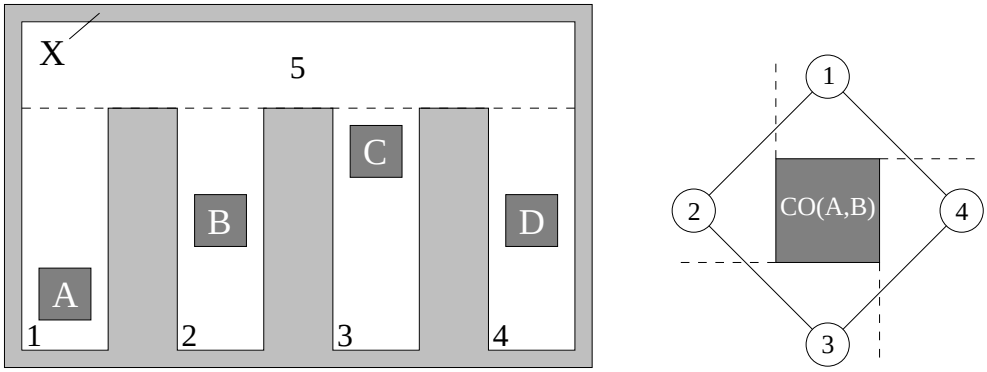


Figure 3.8: Example for D-node reduction: (a) placement of parts, (b) representative floorgraph for two boxes (A, B).

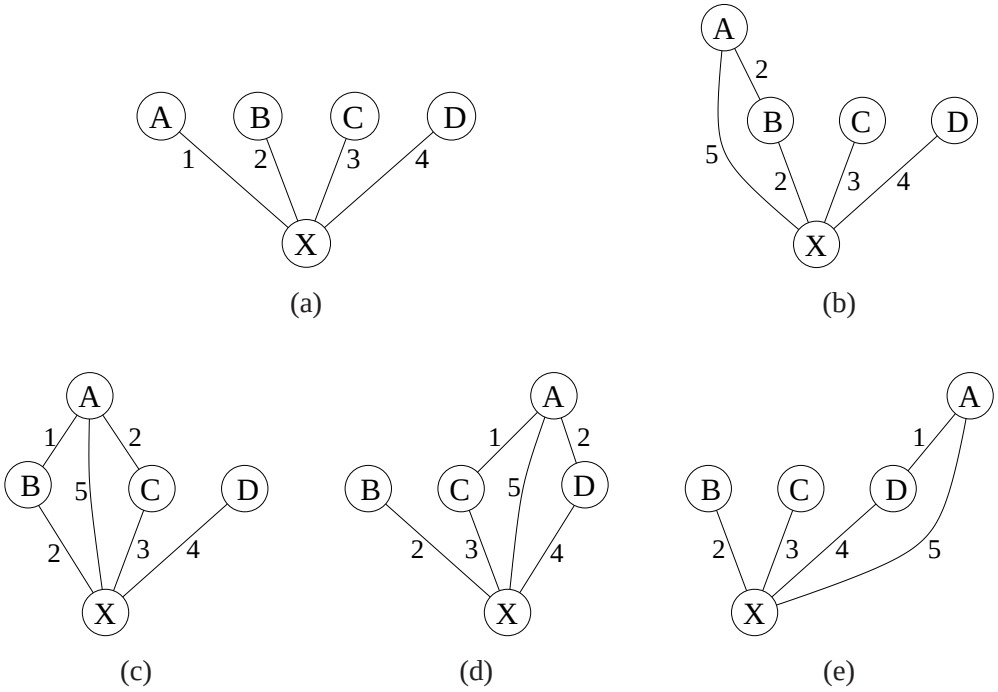


Figure 3.9: (a) Graph representation of reduced D-node for the configuration in Fig. 3.8.a. (b)–(e) Graph representations of reduced D-nodes when A moves from left to right in cell 5 of X.

We form the D-node for the configuration in Fig. 3.8.a by identifying for each pair of parts the floorgraph node that contains their relative placement:

$$(1_{AX}, 2_{BX}, 3_{CX}, 4_{DX}, 3_{AB}, 3_{AC}, 3_{AD}, 3_{BC}, 2_{BD}, 2_{CD})$$

where, e.g. 1_{AX} denotes that the relative position of part A with respect to X is contained in cell 1 in $Floorgraph(A, X)$.

The four boxes cannot interact during a single simultaneous translation, as long as they stay within their current slots. However, even when all boxes are required to stay within different cells in the container, the basic incremental algorithm in the previous section will generate and test an exponential number of D-nodes. (All vertical orderings of the boxes are contained in different D-nodes.) More precisely, in the particular example, the basic algorithm expands D-nodes of the form

$$(1_{AX}, 2_{BX}, 3_{CX}, 4_{DX}, \square_{AB}, \square_{AC}, \square_{AD}, \square_{BC}, \square_{BD}, \square_{CD})$$

where $\square$ corresponds to either node 2 or 3 in the corresponding floorgraphs. This kind of combinatorics is an artifact of the algorithm and is unnecessary since the union of all these D-node cells is a single convex cell. Floorgraphs for all box pairs based on different decompositions, for example using vertical rays, would solve the combinatorial problem for this example D-node. However, inferring suitable decompositions of pairwise configuration spaces in advance is not possible without knowledge of the part placement as a whole. Thus, we proceed differently and show that the partitioning *in the composite configuration space* can be adapted dynamically during the search by discarding redundant constraints.

Since the boxes are well-separated by the container, the current configuration space cell can be described by the reduced tuple

$$(1_{AX}, 2_{BX}, 3_{CX}, 4_{DX})$$

and this is independent of the pairwise box decompositions. Note that floorgraph nodes for all box pairs have been eliminated. The corresponding set of constraints is sufficient to prevent all parts from intersecting, because the boxes are well separated within different cells of the container. On the constraint level, one may verify this by projecting the reduced set of constraints to all pairwise configuration spaces of the eliminated pairs. The projections will not intersect the corresponding pairwise configuration space obstacles. In our case, this reduction replaces an exponential number of cells by a single cell representing their union. (Consider an extension of the above example to k boxes in k distinct slots of a similar container.)

It can also be observed in most other examples that only a subset of all pairs of parts must be separated by explicit constraints on their positions. However, the set of required constraints may change during the search since a sequence of cells in configuration space is traversed and different pairs of parts can come into proximity. These observations motivate the following

Definition 9 (Reduced D-node) *A reduced D-node is a D-node in which one or more constraints have been eliminated safely. We say that a constraint can be eliminated safely from a (reduced) D-node if and only if the set of remaining constraints allows only for placements in which no two parts intersect in their interiors.*

Safe elimination of a single constraint hyperplane extends the D-node cell into one of its neighboring cells. An entire floorgraph node can be removed from the D-node if all its constraints have been eliminated safely. As a consequence, the D-node is augmented to one or more of its successors.

A good concept for representation and visualization of required floorgraph nodes is to interpret D-nodes themselves as graphs:

Definition 10 (Graph representation of a (reduced) D-node) *In the graph representation of a (reduced) D-node, nodes correspond to parts in the assembly, and an edge between two parts is labeled with the current floorgraph node (if required) of the two adjacent parts.*

According to this definition, all D-nodes in the previous section were complete graphs (cliques). Eliminating floorgraph nodes from a D-node tuple corresponds to deleting edges in the graph representation. Note that edges in the graph representation of a D-node are undirected, since they represent floorgraph constraints. The latter do not depend on the order of the parts because they are embedded in the space of absolute part positions.

Fig. 3.9.a shows the graph of the reduced D-node that contains the configuration in Fig. 3.8.a. The edges are labeled with the required floorgraph nodes. This D-node represents all possible configurations in which all boxes stay within their current cells in the container. Edges between pairs of boxes are not required, because the corresponding constraints could be eliminated safely.

Now assume that box A has moved from cell 1 into cell 5 of the container (that is, on the intersection of cells 1 and 5). The reduced D-node for this situation is shown in Fig. 3.9.b. In this context, constraints for the pair AB are required, since B may rise up above the walls in X and interfere with A which is now free to move to the right. (Recall that the cells in the figure only symbolize the real configuration space cells; in particular, cell 2 in X resembles the cell in $CO(B, X)$ for a reference point on B's bottom edge.) Four reduced D-nodes are sufficient to cover all placements in which A is in cell 5 of the container while all other boxes remain in their initial slots (Figs. 3.9.b–e). The figures show the reduced D-nodes that cover an imaginary motion of A from left to right within cell 5 of the container. Note how edges are dynamically added and deleted as A passes the respective vertical constraint between nodes 2 and 1 in $Floorgraph(A, B)$, then $Floorgraph(A, C)$ and finally $Floorgraph(A, D)$. In a generalization of this example with k boxes, the number of reduced D-nodes that are required to cover all placements in which A is in cell 5 of the container is $O(k)$, whereas the number of non-reduced D-nodes would be $\Omega(2^k)$. The combinatorial overhead is caused by considering vertical orderings of boxes in adjacent container slots.

Graph type	*Represents*	*Nodes*	*Edges*
Floorgraph	Convex cell decomposition of pairwise relative placements	Convex cells	Convex cells
D-graph	Implicit convex cell decomposition of composite configuration space	D-nodes	D-edges
D-node	Required floorgraph nodes (constraints on relative part positions for pairs)	Parts	Floorgraph nodes

Table 3.2: Types of graphs for incremental c-space computation.

Note that the constraints that can be eliminated safely from a D-node are *not* necessarily redundant in the classical sense of linear programming. There, constraints are considered redundant *only* if they do not affect the set of feasible solutions. That is, after insertion or deletion of a constraint that is redundant in the classical sense, the cell remains unchanged. In our case, constraints that do partition the original cell may be eliminated additionally. As seen in the above example, the concept of D-node reduction allows to replace up to an exponential number of cells by their union and thus significantly unclutters the decomposition.

In Tab. 3.2, we summarize the different types of graphs defined above.

3.4.2 Improved Incremental Algorithm

In general, by using reduced instead of complete D-nodes, the incremental search algorithm gains in efficiency because

- fewer successor D-nodes have to be generated and tested,
- fewer constraints contribute to each cell in d-dimensional space,
- the cell decomposition becomes coarser, thereby reducing the size of the D-graph, and
- artificially bounded cells become unbounded and allow for earlier termination of the search.

Tab. 3.3 shows the improved incremental search algorithm which is an extension of our basic incremental algorithm from Section 3.3.2 to reduced D-nodes (new parts are highlighted). The new algorithm first constructs a complete D-node for the initial placement and then calls the function $reduce()$. This subroutine removes unnecessary constraints and floorgraph nodes, and returns a reduced D-node. After that, the search proceeds as in the basic version, but only non-eliminated floorgraph nodes are switched, thus generating significantly fewer successor candidates.

Extra work is necessary for maintaining the reduced D-nodes during the search. When moving to a successor D-node, a validation of the reduced D-node must be performed. As a consequence, edges may have to be added to or can be deleted from the graph representation of the D-node. This is done by the

Input: Overlap-free placement of polygons or polyhedra $P_1, \ldots, P_k$.
Output: Sequence of adjacent free cells containing a path that separates the parts.

1. For $1 \leq i < j \leq k$
   - ◇ Compute $Floorgraph(P_i, P_j)$.
   - ◇ Find a floorgraph node n_{ij}^0 that contains the initial placement of P_i and P_j.
2. $S^0 \leftarrow (n_{1,2}^0, \ldots, n_{k-1,k}^0)$ **co:** construct a D-node for the initial configuration.
3. $S_R \leftarrow reduce(S^0)$ **co:** eliminate unnecessary floorgraph nodes
4. $L \leftarrow [S_R]$ **co:** initialize a list of D-nodes.
   $V \leftarrow \{S_R\}$ **co:** initialize a set of visited D-nodes.
5. While not $empty(L)$
   - ◇ $S \leftarrow removefirst(L)$
   - ◇ $L' \leftarrow [\ ]$ **co:** initialize an empty temporary list of successors.
   - ◇ For each node n_{ij} in S **co:** consider only non-eliminated entries
     - ◇ For all neighbours n'_{ij} of n_{ij} in $Floorgraph(P_i, P_j)$
       - ◇ $S' \leftarrow S$. Replace n_{ij} in S' by n'_{ij}.
       - ◇ $S' \leftarrow verify(S')$ **co:** add and/or delete constraints.
       - ◇ If $S' \notin V$ and $feasible(S, n'_{ij})$ then
         - ◇ Supplement S' with a handle to access S in V.
         - ◇ $V \leftarrow S'$ **co:** mark S' as visited.
         - ◇ If $unbounded(S')$ then output the path from S^0 to S' and stop.
         - ◇ $L' \leftarrow append(L', S')$ **co:** append the new D-node to L'.
   - ◇ $L \leftarrow conc(L', L)$ **co:** attach the new successors at the front of L.
6. Output *infeasible*.

Table 3.3: Improved D-graph search algorithm derived from the basic algorithm in Tab. 3.1. Differences are highlighted.

subroutine $verify()$ which takes as argument a (reduced) D-node and returns a copy thereof after having performed the necessary adjustments.

Note that the tasks to be solved by $reduce()$ and $verify()$ are all but trivial, and will be discussed in detail in the next subsection.

The correctness of the improved algorithm follows from the correctness of the basic incremental algorithm and the requirement that removing constraints must be safe, that is, must not introduce collisions. The completeness follows from the completeness of the basic algorithm and the fact that the improved algorithm computes a cover of all D-node cells visited by the basic incremental algorithm.

3.4.3 Problems of D-node Reduction

According to Definition 9 in Section 3.4.1, a constraint can be eliminated safely from a (reduced) D-node, if and only if the set of remaining constraints describes only allowed placements of all parts. In practice, however, this definition is not immediately useful and raises the following questions:

- Which subset of constraints should be eliminated, and how can we identify such a subset?
- Having chosen constraints for elimination, how do we prove that eliminating them is safe, that is, no collisions are possible amongst the parts after elimination?

We discuss these problems next and possible solutions thereafter.

3.4.3.1 Optimal Defragmentation is Intractable

We show that it is a hard problem to determine which constraints should be eliminated from a D-node to yield optimal defragmentation.

In general, the maximum size subset of constraints that can be removed safely from a D-node is not unique. Fig. 3.10 illustrates this problem. The D-node cell is shaded with light gray, and the bounding planes of the embedded linear floorgraph constraints are indicated by thin solid and dashed lines. Bold segments indicate surface patches stemming from the pairwise configuration obstacles. Since we allow touching part boundaries, the patches themselves are not forbidden, but an ϵ-ball centered at a point on such a patch will contain forbidden placements for any $\epsilon > 0$. Thin segments on the cell boundary represent feasible door cells (passages to neighboring cells). Note that in this figure, we do not distinguish constraints stemming from different floorgraphs. As a consequence of our floorgraph decompositions, parts of the obstacle boundary do not contribute hyperplanes to the current cell (for example, the cusps below H). The dashed lines represent hyperplanes that do not bound the current cell, therefore being redundant in the classical sense of linear programming. In this example, discarding the constraint hyperplanes $\{E, F, G\}$ extends the current cell to cover four additional cells (labelled with o's in the figure),

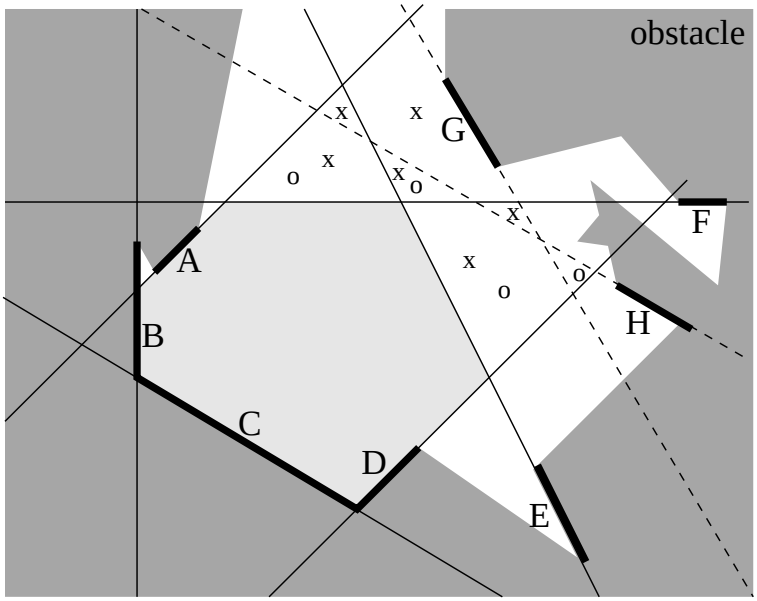


Figure 3.10: Sketch of D-node cell in E^d. Defining constraints (solid lines), forbidden surface patches (thick segments), current non-boundary constraints (dashed lines).

while elimination of $\{E, F, H\}$ covers six additional non-trivial cells (labelled with x's) and therefore yields a better defragmentation than the first subset. The resulting expanded cells are shown in Fig. 3.11.

Optimal cell defragmentation would require finding a subset of constraints that bounds a valid cell of maximum size. To show that this problem is NP-hard, we do not have to give an exact definition of cell size. We merely need the fact that a cell containing at least two different configurations (points) is obviously larger than a cell that contains only a single configuration. Then, NP-hardness is a direct consequence of Lemma 6 in Section 3.2.2 (NP-hardness of finding a cell of dimension greater than zero around a given configuration). In other words, if we could find the best subset of constraints for elimination, we could solve the infinitesimal translational movability problem for polygons which has been shown to be NP-hard [50]. Note that a superset of the constraints required to describe a cell that contains an infinitesimal translational motion can always be derived in polynomial time.

3.4.3.2 Verification of Reduced D-nodes is Costly

For the moment, assume that we could guess (nondeterministically) an optimal subset of constraints that can be removed safely. We must prove that the reduced D-node does not contain forbidden placements. This may be computationally costly although it does not involve combinatorial problems. In particular, we have to prove that no intersections are possible for any pair of parts for which one or more constraints have been eliminated from the D-node. To test whether the reduced D-node cell contains forbidden relative placements of a specific pair of parts (P_i, P_j), the following two different approaches are possible:

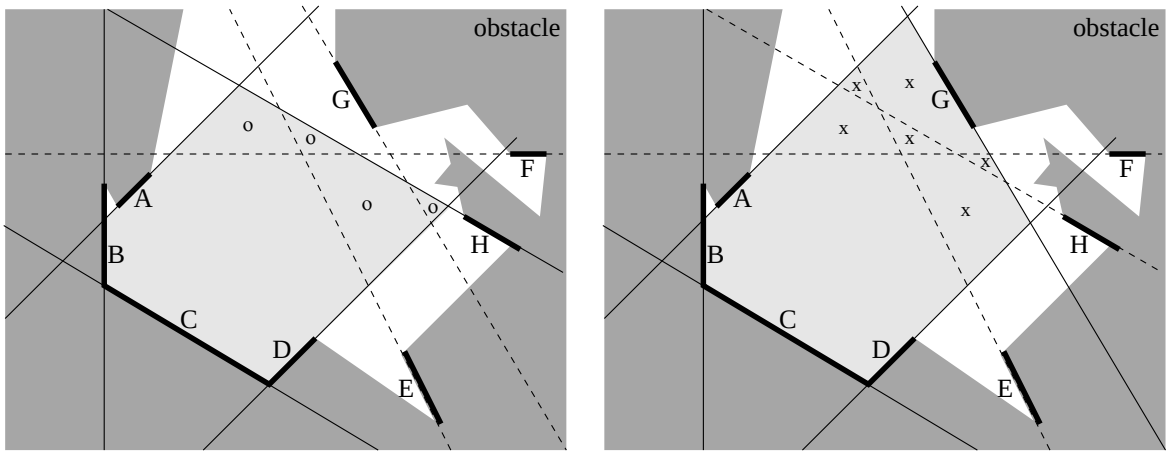


Figure 3.11: Expansion of the cell in Fig. 3.10. (a) Considering elimination of hyperplanes $\{E, F, G\}$. (b) Considering elimination of hyperplanes $\{E, F, H\}$.

- **Embed and intersect in high-dimensional space.** If the configuration obstacle $CO(P_i, P_j)$ is convex or can be decomposed into few convex pieces, we can embed it or its convex pieces and test for intersections with the D-node cell directly by high-dimensional feasibility testing. If $CO(P_i, P_j)$ cannot be decomposed into a reasonably small number of convex pieces, one should consider surrounding it by a convex bounding volume hierarchy (for example, using axis-aligned or oriented bounding boxes), and intersect only the necessary embedded convex bounding volumes with the D-node cell. Such hierarchies are state-of-the art for interference detection of massive polyhedral models and broadly discussed in the literature, e.g. [38].

- **Project and intersect in low-dimensional space.** Alternatively, we can project the D-node cell into the space of relative placements of part P_i with respect to part P_j to test for intersections of the projection with $CO(P_i, P_j)$ in a space of low constant dimension. Interference detection can be accelerated by simultaneously growing and maintaining an inner and an outer approximation of the projection (Appendix A.1). As soon as the inner approximation intersects or the outer approximation no longer intersects $CO(P_i, P_j)$, we can stop. Additionally, bounding volume hierarchies can be used for $CO(P_i, P_j)$. Note that this approach allows for simultaneously projecting the cell into multiple pairwise configuration spaces of disjoint pairs of parts.

Although these methods for detecting interferences for a pair of parts are conceptually straightforward, they may cause a significant computational burden. Each of them may involve multiple operations on the high-dimensional D-node cell, and $\Theta(k^2)$ of them may be required to prove the entire reduced D-node. In practice, however, it is also hard to find a *coherent* (not necessarily optimal) set of removal candidates without intermediate pairwise verifications. (A set of candidates can be called coherent if all of them can actually be removed safely.) We would therefore have to eliminate constraints

one-by-one and perform, after each such elimination step, a verification for all pairs for which we have already eliminated one or more constraints. This would yield a total of $\Omega(k^4)$ such tests to reduce a complete D-node, which is impractical for large k.

3.4.4 Practical Approaches for Reduction and Verification

A smallest subset that allows for identifying redundant pairwise constraints consists of three parts. Note in this context, that a triple of parts involves three floorgraph nodes. We describe a simple and practical elimination scheme based on triples of floorgraph nodes. It ignores the problem of finding an optimal set of removal candidates (the D-node is processed from left to right, and constraints from entire floorgraph nodes are eliminated one-by-one) and reduces the effort for verifying each elimination step by not considering all remaining constraints for the interference tests. The approach does not remove all possible redundancies but has been observed to work with great success in all of our experimentally evaluated examples (cf. Chapter 6). Furthermore, it allows for simple and very efficient implementations of the functions $reduce()$ and $verify()$ that are required by our improved incremental algorithm in the previous subsection.

Thereafter, in Section 3.4.4.2, we discuss an alternative approach that allows for advanced cell defragmentation.

3.4.4.1 Simple and Fast D-node Reduction Based on Triples

To safely eliminate a floorgraph node n_{ij} corresponding to a pair of parts (P_i, P_j) from a D-node, we identify a third part P_g, called a *guard*, that prevents intersections of P_i and P_j. A part qualifies as a guard if suitable floorgraph nodes n_{ig} and n_{jg} (we call them *handcuffs*) are present in the D-node. The constraints from the handcuffs must suffice to prevent intersections of P_i and P_j. For instance, in Fig. 3.8.a, the container X is a guard for all pairs of boxes, and in particular, 1_{AX} and 2_{BX} are handcuffs that allow for elimination of 3_{AB}. Once a node n_{ij} is eliminated, the required handcuffs must be marked and can no longer be eliminated from the D-node.

There are several advantages of this approach: First, interference detection by embedding or projection becomes faster and independent of the number of parts (for a triple, we have a configuration space of constant dimension). Second, since the dependencies are explicit, we can maintain a symbolic database to store and quickly look-up previous eliminations. A pair of handcuffs for a specific elimination candidate has to be verified only once, when first encountered. Referencing a table is usually much faster than explicit methods for interference detection such as linear feasibility testing, cell projection and geometrically based computations, even in low dimensions. Finally, for a single pair of parts, the total number of candidate handcuff pairs in a D-node equals $k - 2$ (the number guard candidates). Therefore, the triple method requires only $O(k^3)$ low cost low-dimensional interference tests (or just table references), while a one-by-one approach using constraints from the complete D-node cell may require $\Omega(k^4)$ high cost high-dimensional operations in the worst case.

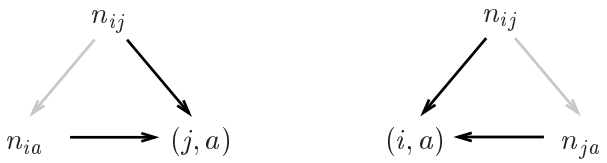


Figure 3.12: Dependencies of elimination: Arrows point from handcuffs to an eliminated D-node entry. Left: n_{ij} and n_{ia} are handcuffs for eliminating (j,a) and part i is the guard. Consequently, n_{ij} is not required as handcuff for n_{ia}, because the latter is itself a handcuff. Right: analogously, part j being the guard.

A further improvement is possible. In addition to fast table look-up, much of the verification process (in the function $verify()$) can be performed dynamically. The effort of considering all triples is actually required only for reducing the first D-node. The key observation is that when we step from the current reduced D-node to one of its successors, only a single floorgraph node is being exchanged. We therefore introduce pointers from eliminated D-node entries (pairs) to their handcuffs and vice versa, to dynamically update the dependencies and test only those pairs that are actually affected by the switch-over.

This dynamic variant is indeed a significant improvement. Assume that a non-eliminated floorgraph node n_{ij} in the current D-node is being replaced by n'_{ij}, which is one of its neighbours in $Floorgraph(i,j)$. We have to find and check all eliminated D-node entries that required n_{ij} as a handcuff. (Using n_{ij} as a handcuff, either P_i or P_j must have been a guard .) To this end, we follow the pointers from n_{ij} to pairs of the form (i,a) and (j,a). If (j,a) had been eliminated, then P_i must have been the guard, and if (i,a) had been eliminated, P_j must have been the guard. Fig. 3.12 shows that at most one of these two pairs can actually have been eliminated and therefore required n_{ij} as a handcuff. Thus, after switching from n_{ij} to n'_{ij}, at most $k-2$ eliminated D-node entries have to be verified. Now assume that the new handcuff candidate n'_{ij} is not sufficient for elimination of $r \leq k-2$ of these previously eliminated D-node entries. These r positions have to be filled with new floorgraph nodes, since the floorgraph nodes that were eliminated somewhere in the past may have been left meanwhile. Finding these new nodes is straight-forward, since we have obtained a valid placement of all parts when moving from n_{ij} to n'_{ij}. The new floorgraph nodes may release at most r handcuffs that pointed to their D-node positions, if not required for another elimination (as a detail, n'_{ij} may be released additionally if it disqualifies as handcuff for all pairs that required n_{ij}). Thus, a set of size bounded by $2r+1+s$ has to undergo a new elimination from scratch where s is the number of floorgraph nodes that have been neither eliminated nor required as a handcuff in the previous D-node. (Recall that we have r fresh floorgraph nodes plus r released handcuffs.) Eliminating this subset from scratch requires $O(k(r+s))$ interference tests (or table references). Assuming that s is small, we have a quadratic worst case bound for the dynamic variant as opposed to the $\Theta(k^3)$ complexity of a complete elimination from scratch.

Note that a D-node reduction scheme based on triples may in general not eliminate all redundancies, regardless of the chosen elimination order. For instance, consider the horizontal placement of boxes

A B C D. Although constraints for the pairs AC, AD and BD are not necessary, one of the corresponding three floorgraph nodes must remain: eliminating the constraints from AD using the handcuffs (AB, BD) prohibits further elimination of BD, and using handcuffs from (AC, CD) prohibits elimination of AC.

3.4.4.2 Advanced Elimination by D-node Construction

To eliminate more redundancies than with the triple-based approach, we have to consider constraints from all pairs of parts simultaneously. We already discussed the resulting problems which are finding a good set of elimination candidates and keeping the number of costly pairwise interference tests as small as possible (cf. Section 3.4.3).

The solution is to perform D-node reduction the other way round: instead of starting with a complete D-node and eliminating constraints, *we begin with an empty D-node and successively add constraints (or floorgraph nodes) until all pairs of parts are well-separated.* There are indeed various good reasons for such a reverse approach.

When we construct the cell from scratch, the number of constraints is initially small and the constraint set approaches its final size from below rather than from above. In contrast to this, when eliminating from a complete D-node, we would initially have to process a large constraint set with high redundancy. As a first consequence, tests for interferences of pairs become much faster.

Our experiments show that the number of required constraints is much smaller than the size of a complete D-node, which depends quadratically on the number of parts. In fact, we observed an almost linear dependence even for very unconstrained placements in which all pairs of parts are potential candidates for collisions (cf. Section 6.3.4). Thus, in addition to the savings for each verifications step, the total number of construction steps is also much smaller than the number of required elimination steps.

At each step of the construction, we can either add a single constraint hyperplane from any pair of parts or a complete floorgraph node. Choosing individual constraints may yield smaller D-nodes descriptions, since at each step, the most promising separating hyperplane may stem from a different pair of parts. However, adding complete floorgraph nodes at each step has the great advantage that each pair of parts needs to be considered exactly once: after a floorgraph node from a specific pair has been added, the pair is guaranteed to be safe, and does not have to be considered again. Of course, we add the floorgraph node only if the partially constructed cell does not separate the parts. Thus, by adding *complete* floorgraph nodes at each step, the total number of pairwise interference tests is reduced to $O(k^2)$.

It remains to consider the order of constraint insertions (each of them pruning the cell) to yield a D-node with few redundant constraints. Although the problem of finding an optimal order is NP-hard (Section 3.4.3.1), there are in fact simple and general heuristics that yield good results. For instance, we may initially compute a valid placement of parts within the current complete D-node. Then, we sort the pairs of parts according to their distances, and add floorgraph nodes in this order. A refinement of this

strategy would be to consider individual faces from the pairwise configuration obstacles and sort them with increasing distance from the current placement. We then add the constraint hyperplane supported by the currently closest face if the current D-node cell does not suffice to separate the corresponding pair of parts. We call the latter *closest face* heuristics (CF). Alternatively, one could use the distance of the current placement from the constraint hyperplanes themselves instead of their supporting faces. This will be called the *closest hyperplane* heuristics (CH).

To assess the effectiveness of such simple heuristics, we compared CF and CH with random order (RH). The results of our experiments suggest that we can reduce the number of constraints from quadratic to an almost linear number (cf. Section 6.3.4).

3.5 C-space Expansion: Extensions and Open Problems

Our above discussion of implicit representation, incremental expansion and elimination of redundancies focuses on the general problem as defined in Definition 5 on page 40. We turn a geometric problem into a symbolic search problem and prune the search space by elimination of redundancies.

This section discusses practically relevant extensions and remaining problems. We first point out improvements and extensions that can be directly integrated into the above framework. Techniques such as heuristic ordering of node expansions and parallelization can be used to speed up the search algorithm.

Then, we discuss extensions and open problems that should be investigated in the future. Possible extensions to other advanced tasks beyond pure separability queries will be sketched. An important open problem is that the general approach of embedding and intersecting pairwise configuration spaces is inefficient in certain cases, despite our reductions in the previous sections. In particular, embedding non-convex subspaces that generate large disjoint regions still causes a combinatorial explosion. We will illustrate the problem by means of an example, and discuss possible solutions.

3.5.1 Acceleration of Search

A) Heuristic Ordering of D-node Expansions

The above algorithms explore the reachable configuration space regions in a simple depth-first order until one or more parts can be removed. Since the branching factor is usually high, there are numerous alternative successors for each visited D-node. So far, the order in which they should be visited was not specified. In a general framework such as the one above, it is impossible to specify an ordering scheme that works equally well in all conceivable cases. If the parts are not separable at all, the reachable component must be expanded completely, regardless of the order in which the successors are examined. But in practical settings, heuristic ordering is reasonable, and can greatly improve the performance of the search. We list some possibilities:

- **Static Heuristics.** The search chooses the successor D-node that allows for the greatest relative progress with respect to the current D-node. This progress can be defined in terms of some dis-

tance measure between two placements, one in the current D-node and the other in the successor candidate. (Recall that we implicitly compute a placement in the door cell between the D-nodes when we test whether the successor is reachable.) Since this heuristics favors the greatest local progress, it resembles the hill-climbing strategy. Another possibility is to measure the distance from the original placement instead of the current placement. These two heuristics can be viewed as static, because they do not depend on the history of the search. We evaluated both of them in practice (cf. Chapter 6).

- **Dynamic Heuristics.** A dynamic heuristics collects information during the search and uses this information to rate candidate D-nodes. We describe two reasonable possibilities.

  - **Floorgraph Node Counters.** An independent counter is attached to each single node in a floorgraph. Each time a floorgraph node is seen in a D-node, its counter is incremented. (Note that the counters are global and not assigned to the individual D-nodes.) The weight of a D-node is computed as the sum of the counters that are associated with its floorgraph nodes. It can be used to assess the degree of novelty of the D-node. As the search proceeds, the weighting of D-nodes that contain further combinations of already visited floorgraph nodes will increase. We can thus pick the more favourable nodes to move to other configuration space regions. This heuristics has shown to be very helpful for overcoming combinatorial traps when a problem contains many temporarily unrelated subproblems. We will explain this in greater detail in Section 3.5.3.
  - **Minimizing the Set of Active Parts.** At the first step, the search switches from the initial D-node to one of its successors by exchanging an arbitrary floorgraph node (relabelling an edge in the graph representation of the (reduced) D-node). Thereafter, we mark the corresponding two parts as active. Subsequent steps from the current D-node to one of its successors prefer to switch a floorgraph node that involves two active parts. When no further progress can be made, we chose a part that is connected to the most active parts in the graph representation of the (reduced) D-node, switch one of these edges, and add the new part to the set of active parts. In intuitive terms, such a heuristics will focus the search on a particular subproblem involving as few parts as possible. In contrast, the above floorgraph counter heuristics favors simultaneous progress in all pairwise configuration spaces.

- **User-defined Constraints and Heuristics.** In a practical setting, the user may add application-specific constraints that substantially narrow the search space. For example, envelopes of allowed relative placements in the pairwise configuration spaces may be used to outline the solution. Other possibilities include specifying costs for undesired part motions, or defining implicit rules for allowed part motions. (Examples for the latter would be to ask for removability of a specific single part or state that 'part A must move out of part C before part B'.) The consequence is that a rough

sketch (a skeleton) of a desired plan is given. An additional depth bound can be introduced, and the search can restrict itself on the region around the skeleton, without danger of getting stuck in combinatorial traps.

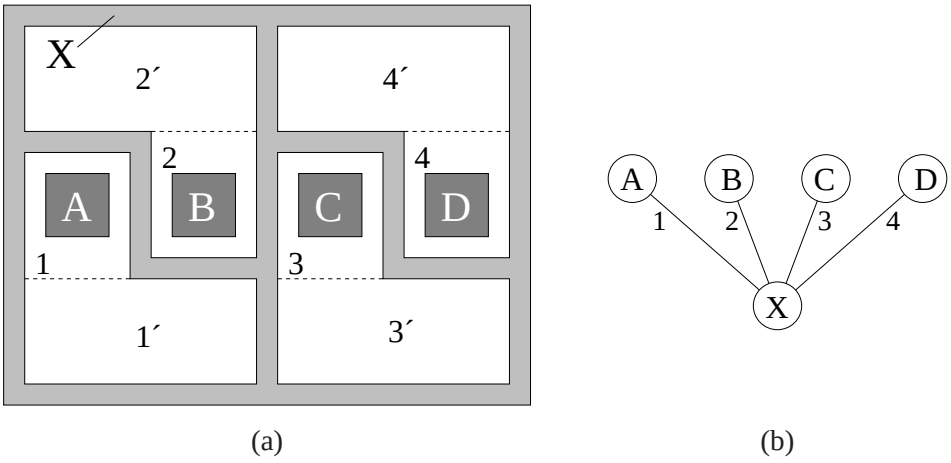
B) Parallelization

Another standard technique for acceleration is parallelization. Common problems arising in this context are communication overhead and interdependencies between subtasks. In our case, required communication is moderate, and the costly operations are sufficiently independent of each other. Most time is consumed by solving linear programs for testing door cells which can be viewed as independent problems. The computational effort for search control itself is negligible. Further measurable costs may arise from computation of pairwise floorgraphs and constraints. These may be computed in a parallel preprocessing step or upon request during the search. Assume we have three modules, running in parallel, and sharing common memory. One module will guide the search and maintain visited D-nodes, another will test of door cells (which can consist of several parallel submodules) and a third one will derive pairwise constraints. Most communication will concern D-nodes and requests for pairwise constraints. These messages are reasonably small in comparison to the effort of feasibility testing and geometric computations so that we may expect definite improvements by parallelization. We will not further evaluate parallelization in this thesis.

3.5.2 Application-specific Extensions

Our general configuration space computation techniques allow for various practical applications different from, but related to pure separability analysis. Since the required extensions will depend on the specific application, we limit our discussion and just point out how we can allow for small overlap between parts and identify collisions that prevent nearly feasible removal motions. This may be an important extension in practice, for example to account for part tolerances.

To allow for overlapping parts, floorgraph cells can be relaxed by introducing additional linear parameters. In particular, a constraint $\mathbf{a}(\mathbf{p}_i - \mathbf{p}_j) \geq b$ on the relative positions of parts i and j can be reformulated as $\mathbf{a}(\mathbf{p}_i - \mathbf{p}_j) \geq b - \epsilon_{ij}$ where ϵ_{ij} is a small bounded positive parameter allowing for overlap of the two parts. When testing a door cell, the linear feasibility test will be replaced by a linear minimization of the sum of all overlap parameters. We rate the successors of the current D-node according to the required overlap. If there is at least one successor with zero overlap, we continue with the search as before. Otherwise, we temporarily defer the search at the current D-node and backtrack to another D-node that may have successors without overlap. Only if there are no further D-nodes left that were reachable without overlap, the search will continue at the D-node that was reachable with the minimum overlap.



(a) (b)

Figure 3.13: (a) Simple example that cannot be properly described by a composite configuration space. (b) Reduced D-node.

3.5.3 Open Problems

D-node Reduction. Our methods for configuration space expansion incrementally compute a convex decomposition of the reachable free regions in the high-dimensional composite configuration space. To avoid cycling and unnecessary duplicate work, we have to mark expanded cells as visited. This is currently done on a floorgraph node resolution. That is, we store reduced D-node tuples in a data structure that allows for efficient insertion and retrieval, and eliminated floorgraph nodes are replaced by a special wildcard character in the tuple. However, eliminating a floorgraph node is only possible if all of its constraints can be discarded. For each cell, we can therefore store only a subset of actually covered space, and cannot guarantee to avoid all unnecessary effort. (Note that this affects only the search efficiency but not the completeness or correctness.) It would therefore be interesting to leave the floorgraph node level to obtain a finer-grained representation of visited space. An important problem in this context is to develop an efficient data structure that allows for storing a union of high-dimensional convex cells and testing whether a high-dimensional convex cell is covered by this union.

Another direction for future investigations is to extend the notion of pairwise constraints. Currently, we assume that a constraint hyperplane for a pair of parts is supported by a face of the corresponding pairwise configuration obstacle. This may not appropriate in the context of a placement of all parts, and one could allow for arbitrary pairwise hyperplanes instead.

Independent Subproblems. An important goal of future research should be to find general strategies that allow for efficient detection and handling of unrelated subproblems. Even when using optimally reduced D-nodes in the sense of Section 3.4.3.1, we may run into a combinatorial trap caused by unrelated subsets of parts. We illustrate the problem using a simple variant of the running example in Section 3.4, shown in Fig. 3.13a. Fig. 3.13b shows the optimal D-node which is unique for the given placement. In this example, working with the optimal D-node does not suffice to eliminate unnecessary combinatorial

work. When we generalize the example to k boxes in different holes of the container, 2^k cells will be expanded, because all parts are considered in a composite configuration space. Note that this is not a problem of the algorithm but inherent to the composite configuration space: at least 2^k convex cells are necessary to cover the free regions.

For the human, it is not hard to see that the example essentially consists of four unrelated problems each of which is non-separable. Of course, a simple heuristics based on pairs of parts could be used to detect that the boxes are in different connected components of the container. But it is in general easy to find counterexamples that make such specialized approaches fail: in our case, assume the boxes are contained in different channels, each with several sharp bends, converging at some region of the container. The above heuristics will not be applicable, and all boxes will have to be considered in a single composite configuration space. A realistic example for such a situation is the bookcase in Fig. 6.9 on page 107. There, the hooks that support the shelves must be lifted and then pulled out of the side panels. These tasks are essentially independent from each other, but the parts are not completely unrelated, since pairs of hooks might intersect during removal.

When we are testing for removability, and there is indeed a removal path for one or more parts, the dynamic heuristics based on floorgraph node counters in Section 3.5.1 provides a solution. First, we attach a counter to each node in a floorgraph. The counters are incremented each time their corresponding floorgraph node is seen in a D-node. We can then compute a weight for a D-node by simply adding the values of the counters associated to its floorgraph nodes. The search algorithm is slightly modified to favor D-nodes with smaller weight over D-nodes that have a greater weight. Thus, the heuristics favors exploration of new floorgraph nodes whenever possible. In the case of unrelated subproblems, the search is guaranteed to explore new regions before wasting exponential effort in mere recombinations of previously visited pairwise relative placements. This simple extension resulted in great improvements in our experiments with the bookcase example (cf. Chapter 6).

However, such an approach does not help when the problem is infeasible. For instance, consider our above example of boxes in interconnected but bounded channels. Another heuristics motivated by this example could simply determine that no box can be removed from the container, so the problem is infeasible. But ending up with a collection of heuristics inspired by special-cases or observations in low-dimensional space was not the goal in this work. Note that in contrast to that, our above dynamic heuristics using the counters tackles the problem on a more general level within our framework for implicit representation and expansion of composite configuration spaces.

It remains to be studied in future work, whether there are general concepts to identify independent subproblems, and exploit this to temporarily plan in separate subspaces. Beyond elimination of combinatorial traps, this would have two advantages: unnecessary constraints would be disregarded, and the dimension of the search space would be (temporarily) reduced.

Chapter 4

Linear Unboundedness Testing

In this chapter, we address the problem of testing for linear unboundedness, that is, finding a direction in which the feasible region of a given set of linear constraints is unbounded. We first consider only conjunctions of constraints. In this case, the feasible region is the intersection of d-dimensional half-spaces forming a (possibly unbounded) convex polyhedron. We describe a new efficient unboundedness testing algorithm that improves by a factor of d over the naive approach. It is optimal when optimal algorithms are used for the two main steps: solving a single homogeneous system of linear equations followed by a single linear feasibility test. To support this claim, we will show that testing for unboundedness is computationally at least as hard as these two subproblems. Our algorithm has specific practical applications in separability analysis (Chapters 2, 3) and can also be of more general use.

We then show that the problem of finding an unbounded direction for linear AND/OR constraint sets is NP-complete. Interestingly, two constraints per disjunction are sufficient to make the problem NP-hard, while the apparently corresponding problem for boolean formulae, 2-SAT, is in P.

4.1 Unboundedness Testing for Conjunctions of Constraints

We begin by formally defining the unboundedness testing problem for conjunctions of linear constraints and describe a simple algorithm for solving it based on $2d$ linear programming problems in d dimensions. We then propose a new algorithm that answers the query by solving a single system of homogeneous linear equations followed by a single linear feasibility test. By then reducing these two subproblems to unboundedness testing, we show that testing for unboundedness is computationally at least as hard as the two dominating steps of our new algorithm. Using the best known algorithms for solving homogeneous equations and linear feasibility testing, the new algorithm is faster than the simple algorithm by a factor of d.

4.1.1 Problem Definition

Consider a set of half-spaces in d dimensions, defined by the inequalities:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1d}x_d &\geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2d}x_d &\geq b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nd}x_d &\geq b_n \end{aligned}$$

or, using matrix notation:

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}.$$

Each of the half-spaces is bounded by a hyperplane of the form $\mathbf{a}_i\mathbf{x} = b_i$, where $\mathbf{x} = (x_1, \ldots, x_d)$, and $\mathbf{a}_i = (a_{i1}, \ldots, a_{id})$ are the hyperplane normal vectors.

The intersection of these half-spaces defines a *convex cell* S in d-dimensional space. Notice that the dimension of S can be lower than d. Without loss of generality, we assume that the intersection of the half-spaces is non-empty, because verifying this condition is a standard problem in linear programming. A non-empty cell is said to be unbounded if it entirely contains a ray.

Definition 11 (Linear unboundedness testing) *The problem of linear unboundedness testing is stated as follows: Given half-spaces in d dimensions with non-empty intersection, decide whether there is a ray $r = \{\mathbf{p} + t\mathbf{u} \mid t \geq 0\}$ that is contained in all half-spaces. If positive, return a valid direction vector $\mathbf{u} \neq 0$.*

Since the cell is convex, the choice of the direction vector $\mathbf{u}$ is independent of $\mathbf{p}$. Thus, any combination of a valid direction vector and a valid point will yield a ray that is contained in the cell. Note that we do not require the coordinates of $\mathbf{u}$ to be all non-negative and that a valid point $\mathbf{p}$ will be delivered as a by-product of the linear feasibility test used initially to verify that the cell is non-empty.

4.1.2 Basics and a Direct Approach

There are two difficulties in solving the unboundedness testing problem with standard tools for linear programming. First, although it can be formulated as a linear optimization problem, the actual objective function, that is, the direction in which the maximum is to be found, is not known in advance. Second, even if a direction is known, unboundedness is usually regarded as an error condition in linear programming: the maximization will fail if the given objective function has no upper bound inside the cell.

In fact, any base of the d-dimensional space provides a sufficient set of directions to test. Let $\mathbf{s}_1, \ldots, \mathbf{s}_d$ be a linear base of $d-$space. We can perform a maximization for each of $+\mathbf{s}_i$ and $-\mathbf{s}_i$, subject to the half-space inequalities defining the given cell. If one of these maximizations fails, the cell is unbounded. Notice, however, that a direction $\pm\mathbf{s}_i$ that causes the maximization to fail is not necessarily a valid direction vector for a ray inside the given cell. For example, consider the half-plane defined

by the constraint $y - x \geq 1$ in E^2. A maximization along $\mathbf{e}_1 = (1, 0)$ will report unboundedness, since x can become arbitrarily large, but no ray with direction vector $\mathbf{e}_1$ is contained in the half-plane.

To compute a valid direction vector, we first transform the cell S by shifting the bounding hyperplanes to the origin, that is, we replace the defining half-spaces $\mathbf{a}_i\mathbf{x} \geq b_i$ by the corresponding homogeneous half-spaces $\mathbf{a}_i\mathbf{x} \geq 0$. We denote the transformed cell by S_0, since it always contains the origin. The following lemma and corollary show that this transformation does not affect the set of unbounded directions: S is unbounded in a direction $\mathbf{u}$ if and only if S_0 is unbounded in direction $\mathbf{u}$.

Lemma 9 *S_0 contains a point $\mathbf{u} \neq 0$ if and only if S is unbounded in direction $\mathbf{u}$.*

Proof: Let $\mathbf{u} \neq 0$ be a vector in S_0 ($\mathbf{Au} \geq 0$) and $\mathbf{p}$ be a point in S ($\mathbf{Ap} \geq \mathbf{b}$). Thus, for $t \geq 0$, $\mathbf{A}(\mathbf{p} + t\mathbf{u}) \geq \mathbf{b}$ and S is unbounded in direction $\mathbf{u}$. Now assume that S contains a ray $\{\mathbf{p} + t\mathbf{u} \mid t \geq 0\}$. Thus $\mathbf{A}(\mathbf{p} + t\mathbf{u}) \geq \mathbf{b}$ and, equivalently, $t(\mathbf{Au}) \geq \mathbf{b} - \mathbf{Ap}$ for any $t \geq 0$. This implies $\mathbf{Au} \geq 0$. Thus $\mathbf{u}$ is in S_0 and $\mathbf{u} \neq 0$ by definition of the ray. □

Corollary 3 *S_0 contains a point $\mathbf{u} \neq 0$ if and only if S_0 is unbounded in direction $\mathbf{u}$.*

Now, testing for unboundedness can be reduced to finding a point $\mathbf{u} \neq 0$ within the transformed cell. If successful, the location vector of this point specifies a direction in which the original cell is unbounded. We can determine $\mathbf{u}$ by, e.g., intersecting the transformed cell S_0 with the bounding hyperplanes of a d-dimensional box centered at the origin. Instead of maximizing in the directions $+\mathbf{e}_i$ and $-\mathbf{e}_i$ for each unit vector $\mathbf{e}_i$, we perform feasibility tests on the constraint sets $\{\mathbf{Ax} \geq 0, \mathbf{e}_i\mathbf{x} = 1\}$ and $\{\mathbf{Ax} \geq 0, \mathbf{e}_i\mathbf{x} = -1\}$. Any solution is then a non-trivial point (not the origin) in S_0 and thus a valid direction vector $\mathbf{u}$.

This simple algorithm requires $2d$ linear feasibility tests in the worst case to decide if the set is unbounded. Is this the most efficient way of testing? In particular, can we detect boundedness (the worst case for the simple algorithm) with a faster method? We address these questions next.

4.1.3 An Efficient Algorithm

There is indeed an intuitive approach for unboundedness testing using a single linear maximization that works in most cases: compute the direction of optimization as the sum of all constraint hyperplane normal vectors.

This amounts to simultaneously maximizing the distance from all hyperplanes and yields a correct but incomplete unboundedness test. That is, in certain cases an unbounded cell might be classified as bounded. In the following, we show how this problem can be fixed. Tab. 4.1 presents a new algorithm which is motivated by the above idea but includes an initial step to identify the critical cases. It is therefore correct and complete. We begin with an informal description and then formally prove correctness and completeness.

First, we test if the homogeneous system of *equations*, derived from the given inequalities, is underconstrained (step 1). If it is, the cell is unbounded since the solution space is at least of dimension one,

Input: Half-spaces $H_1^+, \ldots, H_n^+$ defined by inequalities $H_i^+ : \mathbf{a}_i\mathbf{x} \geq b_i$.

Output: unbounded direction $\mathbf{u}$ when the cell defined by $H_1^+ \cap \ldots \cap H_n^+$ is *unbounded* or *bounded* otherwise.

1. If the homogeneous system of equations $\mathbf{a}_i\mathbf{x} = 0, 1 \leq i \leq n$, has a non-trivial solution $\mathbf{u} \neq 0$, return *unbounded*, the vector $\mathbf{u}$, and exit.
2. Construct a combined direction $\mathbf{a}_\Sigma := \sum_{i=1}^{n} \mathbf{a}_i$.
   If $\mathbf{a}_\Sigma = 0$ return *bounded* and exit.
3. Construct the constraint set C as:

$$\begin{aligned} \mathbf{a}_i\mathbf{x} &\geq 0, \quad 1 \leq i \leq n \\ \mathbf{a}_\Sigma\mathbf{x} &= 1 \end{aligned}$$

   If C is feasible, return *unbounded*. The solution of the feasibility test is the direction $\mathbf{u}$. Else return *bounded* and exit.

Table 4.1: The new linear unboundedness testing algorithm.

and we can compute non-trivial solutions directly. If the homogeneous system of equations is *not* under-constrained, we construct a combined vector as the sum of the hyperplane normal vectors (step 2). The hyperplane defined by this vector at a positive distance from the origin will intersect any ray emanating from the origin in the cell due to the full rank of the matrix. If the combined vector is zero, no such plane exists and thus the cell is bounded. Otherwise, we construct a new set of constraints consisting of the homogeneous inequalities and the new hyperplane (step 3). Now unboundedness can be detected with a single linear feasibility test on the new constraint set.

This algorithm reduces the number of hyperplanes to be tested from $2d$ hyperplanes (the simple test in Section 4.1.2) to a single hyperplane with combined normal $\mathbf{a}_\Sigma$. This reduction is only possible after we have established the range of the underlying constraint set, which means that step 1 is not redundant, as illustrated by the following example. Let $\mathbf{a}_1 = (-1, 0)$, $\mathbf{a}_2 = (2, 0)$, $b_1 = -2$, $b_2 = 2$. After transformation, the cell containing the origin, i.e. the cell defined by $\mathbf{a}_1\mathbf{x} \geq 0$ and $\mathbf{a}_2\mathbf{x} \geq 0$ is unbounded, but $\mathbf{a}_\Sigma = (1, 0)$, so that C is infeasible.

We now prove the correctness and completeness of the algorithm. The proof for the correctness of step 1 follows from Lemma 9 and the fact that the set of solutions of $\mathbf{A}\mathbf{x} = 0$ is a subset of S_0. Notice that step 1 fails if and only if $dim\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle = d$. To prove the correctness and completeness of steps 2 and 3 when applied after step 1, we state:

Lemma 10 (variant of Farkas' lemma) *Let $dim\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle = d$. Then S_0 contains a point $\mathbf{u} \neq \mathbf{0}$ if and only if C is feasible.*

Proof: Let $\mathbf{u} \neq \mathbf{0}$ be a point in S_0. We must show that C is feasible. Assume $\mathbf{a}_\Sigma \mathbf{u} = 0$. Since $\mathbf{u}$ is in S_0 ($\mathbf{a}_i \mathbf{u} \geq 0$), we have $\mathbf{a}_i \mathbf{u} = 0$ for each $i \leq n$, so $\mathbf{u}$ is orthogonal to each $\mathbf{a}_i$. But this would imply that $dim\langle \mathbf{a}_1, \ldots, \mathbf{a}_n, \mathbf{u} \rangle$ is strictly greater than d. Thus $\mathbf{a}_\Sigma \mathbf{u} \neq 0$. The same argument shows that $\mathbf{a}_\Sigma \neq \mathbf{0}$ if the space spanned by $\mathbf{a}_1, \ldots, \mathbf{a}_n$ has full dimension and S_0 contains a point $\mathbf{u} \neq 0$. For each scalar $t \geq 0$, the point $t\mathbf{u}$ is in S_0. Thus after appropriate scaling, we can assume $\mathbf{a}_\Sigma \mathbf{u} = 1$, so that $\mathbf{u}$ satisfies C. Conversely, a point satisfying C is clearly a non-zero point in S_0. □

4.1.4 Complexity Analysis

Is this test the fastest possible test? Our algorithm relies on finding a non-trivial solution for a homogeneous system of linear equations and linear feasibility testing. Is there a test that does not require one or both of these two steps? The following constructions show that the problem of testing for unboundedness and finding an unbounded direction is at least as hard as these two steps. Based on these reductions, the new algorithm is optimal when optimal algorithms for solving the homogeneous equations and for feasibility testing are used. We show that with known algorithms for these two problems the new unboundedness testing algorithm that is faster by a factor of d over the simple approach in Section 4.1.2.

Lemma 11 *Testing for linear unboundedness is at least as hard as finding a non-trivial solution for a homogeneous system of linear equations.*

Proof: We reduce the problem of finding a non-trivial solution of a homogeneous linear system to unboundedness testing. Given a homogeneous system of n linear equations $\mathbf{A}\mathbf{x} = 0$ (E), we construct an equivalent homogeneous system (U) of $2n$ linear inequalities consisting of $\mathbf{A}\mathbf{x} \geq 0$ and $\mathbf{A}\mathbf{x} \leq 0$. We determine a non-trivial solution $\mathbf{u} \neq 0$ for (E) by finding an unbounded direction for (U). Due to Corollary 3, (E) allows only for the trivial solution if and only if (U) is bounded. Notice that (U) has twice as many rows as (E). However, assuming that testing for unboundedness is polynomial in n, this increase does not affect the asymptotic computing time. Also, the described transformation itself is dominated by the unboundedness test and thus does not increase the complexity. □

Lemma 12 *Testing for linear unboundedness is at least as hard as linear feasibility testing for the constraint set C.*

Proof: We give a simple reduction of feasibility testing for C to unboundedness testing. Consider the linear feasibility problem (C)

$$\begin{aligned} \mathbf{A}\mathbf{x} &\geq 0 \\ \mathbf{a}_\Sigma \mathbf{x} &= 1 \end{aligned}$$

with $dim\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle = d$, i.e., the rank of the matrix $\mathbf{A}$ is full. We can transform this problem into a (homogeneous) test for unboundedness by setting (U)

$$\begin{aligned} \mathbf{A}\mathbf{x} &\geq 0 \\ \mathbf{a}_\Sigma \mathbf{x} - z &= 0 \\ z &\geq 0 \end{aligned}$$

We must show that (C) is feasible if and only if the cell defined by the system (U) is unbounded. First, assume that (C) is feasible, i.e. $\mathbf{A}\mathbf{x}_0 \geq 0$ and $\mathbf{a}_\Sigma \mathbf{x}_0 = 1$ for some $\mathbf{x}_0 \in E^d$. Thus $z(\mathbf{A}\mathbf{x}_0) \geq 0$ and $z(\mathbf{a}_\Sigma \mathbf{x}_0 - 1) = 0$ for any $z \geq 0$ and (U) is unbounded in direction $(\mathbf{x}_0, 1)$. Now assume that (U) is unbounded, i.e., there is a vector $\mathbf{u} = (\mathbf{x}_0, z) \neq 0$ in (U). We first show that $z > 0$. Assume that $z = 0$ (and thus $\mathbf{x}_0 \neq 0$). Since $\mathbf{u}$ is in (U) we would then have $\mathbf{a}_\Sigma \mathbf{x}_0 = 0$ and $\mathbf{A}\mathbf{x}_0 \geq 0$. But this implies that $\mathbf{a}_i \mathbf{x}_0 = 0$ for all rows of $\mathbf{A}$ and we have a non-zero vector $\mathbf{x}_0$ that is orthogonal to all $\mathbf{a}_i$, in contradiction to $dim\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle = d$. Thus, after appropriate scaling of $\mathbf{u}$, we can assume that $z = 1$, so that $\mathbf{u}' = (\mathbf{x}_0', 1)$ is in (U). Thus, $\mathbf{x}_0'$ is in (C).

The transformation from (C) to (U) has increased both the dimension d and the number n of constraints of the problem by one. Assuming the number of steps required for a linear feasibility test (with n constraints in d dimensions) is polynomial in n and in d, this increase does not affect the asymptotic computing time. □

The following more general lemma implies that we can in fact reduce any linear feasibility testing problem to unboundedness testing. Thus, even without allowing explicit feasibility testing, our initial prerequisite that the input cell is non-empty is not a restriction.

Lemma 13 *Testing for linear unboundedness is at least as hard as linear feasibility testing.*

Proof: Given a linear feasibility problem (F)

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}$$

we construct a homogeneous set of inequalities (U) that is unbounded if and only if (F) is feasible. Note that there are no implicit non-negativity constraints on the variables in $\mathbf{x} = (x_1, \ldots, x_d)$. We define (U) using the variables $z \geq 0, \mathbf{u} = (u_1, \ldots, u_d)$ and $\mathbf{v} = (v_1, \ldots, v_d)$:

$$\begin{aligned} \mathbf{A}(\mathbf{u} - \mathbf{v}) &\geq z\mathbf{b} \\ cz &\geq u_1 + v_1 + \ldots + u_d + v_d \\ u_j &\geq 0, \quad j = 1, \ldots, d \\ v_j &\geq 0, \quad j = 1, \ldots, d \end{aligned}$$

where $c > 0$ is an appropriately large constant (see below).

We first show that (F) is feasible if (U) is unbounded. If (U) is unbounded then there is a vector $\mathbf{w} = (z, u_1, v_1, \ldots, u_d, v_d) \neq 0$ that satisfies the constraints in (U). Since $cz \geq u_1 + v_1 + \ldots + u_d + v_d$ we must have $z > 0$. Therefore $\mathbf{A}(\mathbf{u} - \mathbf{v})/z \geq \mathbf{b}$ and $\mathbf{x} = (\mathbf{u} - \mathbf{v})/z$ satisfies (F).

Now assume that (F) is feasible, i.e., there is a point $\mathbf{x}$ that satisfies the constraints in (F). We show that (U) is unbounded by specifying a non-trivial point $\mathbf{w} = (z, u_1, v_1, \ldots, u_d, v_d)$ in (U). We set $z = 1$ and $u_j = x_j, v_j = 0$ if $x_j \geq 0$ and $u_j = 0, v_j = x_j$ otherwise. Thus $\mathbf{A}(\mathbf{u} - \mathbf{v}) \geq z\mathbf{b}$. To satisfy $cz \geq u_1 + v_1 + \ldots + u_d + v_d$, the constant c must have been chosen sufficiently large in advance. In Appendix B, it is shown that such a bound can be computed in a preprocessing step that is guaranteed to be not slower than any possible unboundedness test and therefore does not spoil the total running time.

Finally, as in the previous lemma, the added variables and constraints do not affect the asymptotic running times, that is, an $O(n^\alpha d^\beta)$-time unboundedness test can be used to construct a feasibility test with the same asymptotic complexity. □

The new algorithm in Section 4.1.3 does not specify how solving the homogeneous equations and linear feasibility testing are to be performed. Based on known algorithms for these two sub-problems, we obtain the following time bound for our test:

Lemma 14 *The complexity of the algorithm in Tab. 4.1 is $O(nd^{4.5})$ where n is the number of constraints and d is the number of variables.*

Proof: Based on Karmarkar's method, linear maximization and feasibility testing take at most $O(Ld^{3.5})$ steps, where L is the accumulated length of the input coefficients [74]. If the length of each individual coefficient is fixed and constant, i.e., the number of bits necessary to represent each number is at most l, then $L = O(nd)$, so that the described feasibility test takes at most $O(nd^{4.5})$ steps. Step 1 can be performed in $O(nd^2)$ steps (for n constraints and d variables, [67]). Thus the complexity of the algorithm in Tab. 4.1 is $O(nd^{4.5})$ versus $O(nd^{5.5})$ for the simple method in Section 4.1.2, and our algorithm allows for a reduction of time bounds by a factor of d when compared to the simple method. □

4.2 Unboundedness Testing for AND/OR Constraint Sets

Allowing both conjunctions and disjunctions of constraints, is it still possible to test for unboundedness in deterministic polynomial time? We show that with two constraints per disjunction, unboundedness testing becomes NP-hard, even when all variables are required to be non-negative. This is not obvious, since the apparently corresponding problem for the case of boolean formulae, known as 2-SAT, can be decided in deterministic polynomial time. The difference is that in our case each constraint may contain up to three variables, whereas in 2-SAT, a literal contains information of a single variable only. We exploit this observation in the proof of the following

Lemma 15 *The problem of finding an unbounded direction (with non-negative variables) for a linear AND/OR constraint set with two constraints per disjunction and at most three variables per constraint is NP-hard.*

Proof: By reduction from 3-SAT. (A somewhat similar reduction from 3-SAT to integer linear programming is given in [26].) For a given 3-SAT expression E we construct a AND/OR constraint set with at most two constraints per disjunction and at most three variables per constraint that is unbounded in a direction with non-negative coordinates if and only if there is a satisfying assignment for the variables in E.

We represent a boolean variable X_i in E by two non-negative real-valued variables t_i and f_i. $X_i = TRUE$ will be expressed by $t_i > 0, f_i = 0$ while $X_i = FALSE$ will be expressed by $t_i = 0, f_i > 0$.

First, we must ensure that if the set of solutions is unbounded then exactly one of t_i, f_i is strictly greater than zero while the other one is equal to zero. To this end, we introduce a new non-negative variable z and add the constraints (V_i)

$$t_i \leq 0 \quad \vee \quad f_i \leq 0$$

$$t_i \geq z \quad \vee \quad f_i \geq z$$

$$t_i + f_i \leq z$$

Together with $t_i, f_i, z \geq 0$ it follows that either $z = 0$ and $t_i = f_i = 0$ or $z > 0$ and either $t_i = 0, f_i = z$ or $f_i = 0, t_i = z$. Note that the same z can be used in these constraints for all pairs (t_i, f_i).

We now describe constraints representing the clauses in the given boolean expression E. Let α_i be a literal, that is, $\alpha_i = X_i$ or $\alpha_i = \neg X_i$. A clause $\alpha_a \vee \alpha_b \vee \alpha_c$ will be transformed into the constraint (for the moment we drop the requirement that a constraint contain at most three variables)

$$x_a + x_b + x_c \geq z \tag{4.1}$$

where $x_i = t_i$ if $\alpha_i = X_i$ and $x_i = f_i$ if $\alpha_i = \neg X_i$. Note that we have the implicit non-negativity constraints on all variables.

Representing all clauses this way, the complete AND/OR set of constraints constructed so far is unbounded if and only if there is a satisfying truth assignment for the variables in E. First assume that E has a satisfying truth assignment. We may freely choose $z \geq 0$ as long as we set $t_i = z, f_i = 0$ if $X_i = TRUE$ in the satisfying assignment for E and $t_i = 0, f_i = z$ otherwise. Thus the solution space of the constraint set is unbounded. Conversely, assume that the solution space of the AND/OR constraint set is unbounded and consider a non-trivial solution. The constraints (V_i) require $z > 0$ in this case. Furthermore, for each pair (t_i, f_i), we have either $t_i = z, f_i = 0$ or $t_i = 0, f_i = z$. To satisfy an inequality of the form (4.1), at least one of the three involved variables on the left side must be equal to z. Thus the variables pairs (t_i, f_i) indicate a truth assignment that satisfies the expression.

Finally, each constraint with four variables, $x_a + x_b + x_c \geq z$ can always be substituted by introducing a new variable $y \geq 0$ and two constraints with three variables each, $x_a + x_b \geq y$ and $y + x_c \geq z$. □

Note that the overhead of the described transformations is bounded by a polynomial. To be more precise, for the reduction the input size is not increased more than by a polynomial function asymptotically. A simpler variant of the proof shows that AND/OR feasibility testing, under the same conditions, is NP-hard. Specifically, the constraints (V_i) that assert consistent variable assignment simplify to $t_i \geq 1 \vee f_i \geq 1$ and $t_i + f_i \leq 1$, and the constraints (4.1) become $x_a + x_b + x_c \geq 1$.

Theorem 6 *The problem of finding an unbounded direction for binary AND/OR constraint sets with non-negative variables is NP-complete.*

Proof: It remains to show that the problem is in NP. This is straight-forward: a non-deterministic algorithm guesses an unbounded combination of constraints (one constraint from each disjunction) and then calls our efficient unboundedness testing algorithm for the resulting conjunction of constraints. □

Chapter 5

Implementation

In this chapter, we address three problems that are important for a practical implementation of our algorithms in Chapter 3:

Computation of Configuration Spaces for Pairs of Parts. Computing configuration spaces for pairs of polygons is well-understood and reasonably efficient in practice. We give a short summary of literature and known algorithms. For general non-convex polyhedra, the problem can become more intricate. We discuss the practical challenges and point out how to approach them.

Book-keeping of Visited D-nodes. Visited D-nodes must be stored for two purposes. First, we must prevent the search from cycling and unnecessary overhead. Second, for most search strategies, a tree of D-nodes has to be maintained in order to store candidate subpaths.

Linear Programming in a Composite Configuration Space. Testing a door cell between the two D-nodes requires a linear feasibility test. On the whole, these tests consume most of the running time of our basic incremental algorithm. We show what kind of improvements are possible here.

5.1 Computation of Configuration Spaces for Pairs of Parts

The general problem of computing pairwise configuration spaces comprises several well-studied subclasses. For some of them, efficient algorithmic solutions have been proposed (for example, cf. [42, 52, 57]). In our context, Minkowski sums and convex decompositions have to be computed in order to obtain linear constraints for pairs of parts.

In the planar case, both the Minkowski sum and the equally useful convolution of polygons have been studied in [33, 55]. For computation of convex decompositions in two dimensions, cf. [49].

In the case of polyhedral parts, preprocessing becomes more intricate. Guibas [20] computes convolutions of convex polyhedra by reciprocal search. Kaul [32] considers Minkowski sums of regular polyhedra. In the context of satellite antenna layout, Boissonnat [2] computes projections of three-dimensional Minkowski sums to a plane directly in the lower-dimensional projection space. Recently,

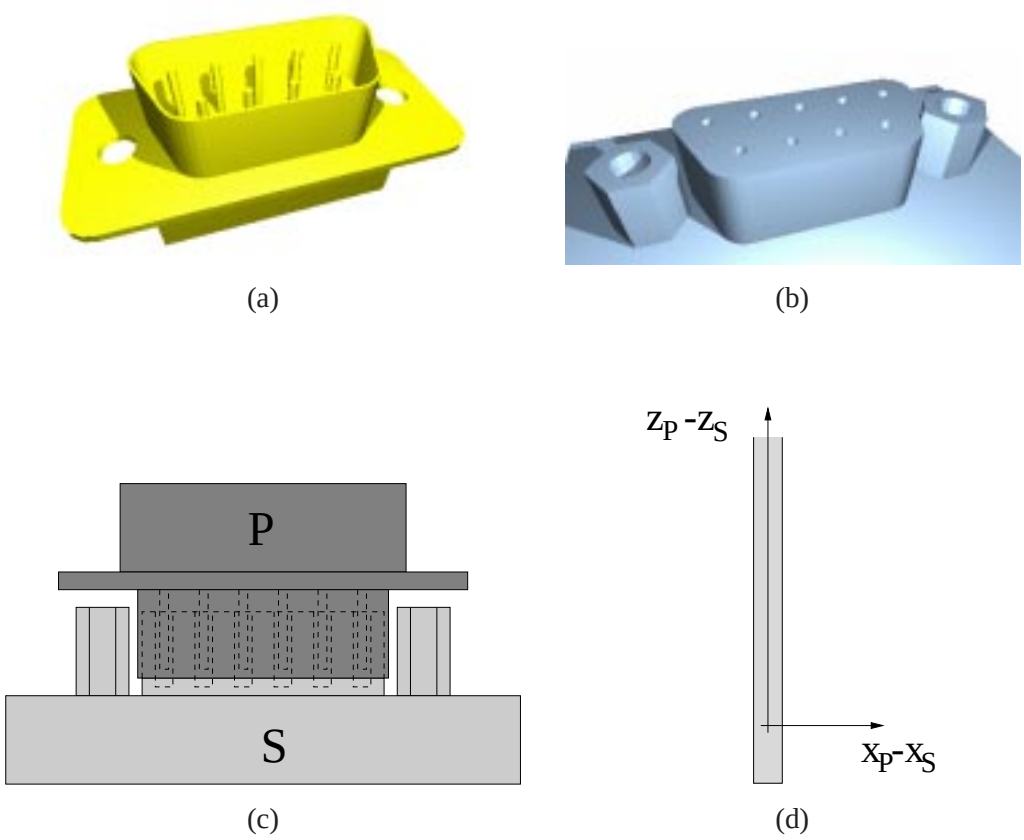


Figure 5.1: (a) RS-232 plug P and (b) socket S. (c) Initial configuration (side view). (d) Projection of cell for configuration in (c).

Ramkumar [55] developed a theory of convolutions for non-convex polyhedra, thus extending the work of Guibas et al. [19] to three dimensions.

When the considered parts have relatively few faces, complete floorgraphs can be computed in a preprocessing step. This approach was used in our implementation. To obtain the Minkowski sums, our program decomposes the parts into convex pieces and computes the Minkowski sums of all pairs of pieces. It then obtains complete floorgraphs by performing a plane-sweep over the resulting arrangement. Running times for preprocessing all pairs of parts ranged from fractions of a second up to several seconds in our examples.

For complex models with many faces, complete configuration space computation for pairs of parts can become a major obstacle. Note however, that the high-dimensional search in our application proceeds incrementally. We therefore do not need to pre-compute complete configuration spaces. Instead, it is sufficient to compute a current cell and its neighbors for each pair of parts. During the search, only the required parts of the pairwise configuration spaces will be expanded. Although parts may be quite complex, the portions of the Minkowski sums that actually have to be computed are usually much simpler in assembly planning applications (e.g., for a situation similar to a peg in a hole, the current cell may be

only a ray or a cylinder as illustrated in Fig. 5.1).

The idea of computing configuration spaces locally is not new. Ramkumar [55] proposes in his conclusions to compute local portions of the convolution from which parts of the Minkowski sum surface can be derived. However, no algorithm or experimental results are given there. Bounding volume hierarchies as described e.g. in [38] are a common approach for computing the distance between non-convex polyhedra. They are practical for large models and can be easily extended to extract relevant pairs of triangles for incremental configuration space cell computation. To illustrate this, we created two coaxial cylinders with 512 triangles each. The cell for the selected configuration is also a cylinder. Filtering the closest triangle pairs required testing $13,000$ bounding volume pairs and $2,600$ triangle pairs. The resulting triangle pairs are sufficient to construct the cell. The total running time was only a fraction of a second on a SUN Ultra-60 workstation with 300 MHz. In contrast, considering all $260,000$ pairs and computing the complete convolution of the two cylinders would be much slower.

5.2 Book-keeping of Visited D-nodes

As the search proceeds, the explored configuration space regions will be covered by visited D-nodes. To avoid visiting the same configuration space regions multiple times, we must be able to efficiently store and look-up visited D-nodes. In addition to that, candidate solution paths must be maintained.

To encode a D-node, we use a tuple whose entries are pointers to the corresponding floorgraph nodes. A special symbol (we use an asterisk $*$) indicates that the corresponding floorgraph node has been eliminated. The actually used domain of each entry is normally relatively small in practical cases. (The number of actually visited nodes in a single floorgraph depends on how many distinct critical relative placements the two corresponding parts will to take.) However, the length D of the tuple depends quadratically on the number of parts ($D = k(k-1)/2$ for k parts), and this is problematic: although a total order on D-nodes can be defined in a straight-forward way (e.g. the lexicographic order), it is not recommendable to store D-nodes using a standard comparison-based one-dimensional set data structure. Each comparison of two D-nodes may require scanning the whole tuple at quadratic cost in the worst case. For a set containing n D-nodes, insertion or look-up of a single D-node may require several such comparisons and takes time $\Theta(D \log n)$ in the worst case. Hashing could be used instead, but it may be difficult to find a hash function that produces few collisions. Beside that, it is difficult to assess a good size for the required hash table in advance.

Trie Approach

However, the greatest shortcoming of the above one-dimensional approaches is that they are wasteful in terms of space. A D-node and all its successors differ only at a single position. Even amongst all D-nodes generated during the whole search process, there may be large subsets that share the same substring.

We therefore propose to store D-nodes using the well-known trie-concept [36]: D-nodes are stored

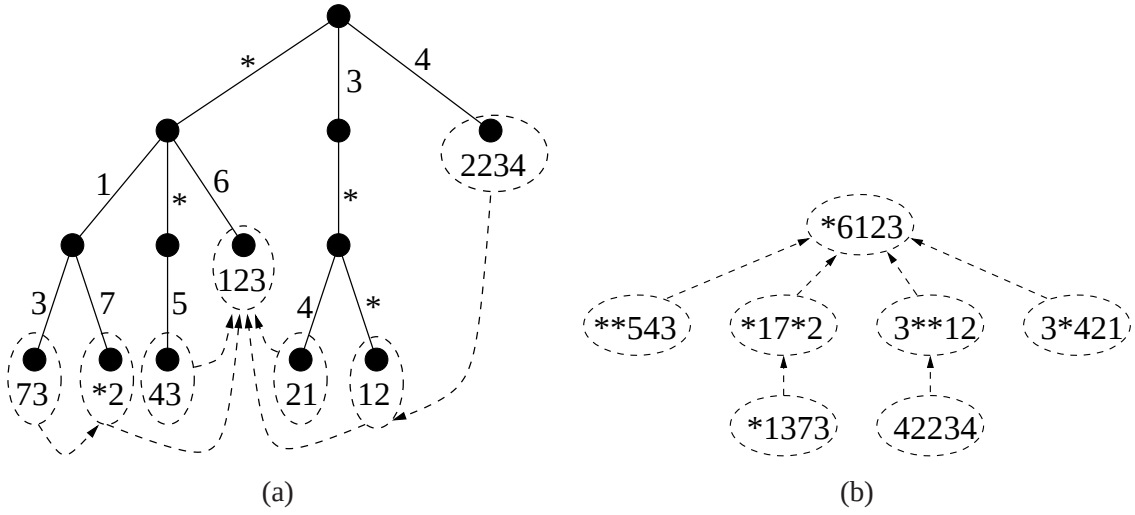


Figure 5.2: (a) Trie representation for efficient access to visited D-nodes. (b) Search tree with D-nodes.

as paths in a tree, starting at the root node. Edges along each path are labeled with the entries in the corresponding D-node, from left to right. Thus, two D-nodes with a common prefix share a subpath starting at the root and labeled with the maximum length common prefix. To reduce storage requirements, it is common to store unique suffixes at the leaf nodes. Fig. 5.2.a (solid drawing) shows an example trie representing the set

$$\{(*,1,3,7,3),(*,1,7,*,2),(*,*,5,4,3),(*,6,1,2,3),(3,*,4,2,1),(3,*,*,1,2),(4,2,2,3,4)\}.$$

A trie approach is simple to implement, and proved to be efficient in our experiments. Let m be the maximum number of children of a single node in the trie. (m is bounded by the number of floorgraph nodes that will be visited for a single pair of parts. Note that this bound is very conservative, especially for nodes at the lower levels in the trie.) In our case, m can be considered to remain much smaller than n, which was defined above as the number of currently stored D-nodes. The insert operation for a D-node into a trie is straight-forward and can be done in time $O(D \log m)$ in the worst case: we scan the D-node from the left to the right, following a possibly existing path in the trie and branching off when necessary. This compares favorably with the $\Theta(D \log n)$ effort for a comparison based one-dimensional data structure. Note that n may grow rapidly during the search. To look-up a D-node, the trie can be traversed in a similar way at the same $O(D \log m)$ cost. For efficient $O(\log m)$-time access, the children of each node have to be organized using a standard set data structure [43].

For a general discussion of multi-dimensional indexes, cf. [44]. Note that spatial access methods have also been investigated in the field of database applications [15].

Overlaid Search Tree

A general graph search algorithm maintains a tree of candidate subpaths. In our application, these consist of sequences of D-nodes. To save space, we can overlay the search tree over the above trie. Fig. 5.2.b shows a sample search tree. The dashed lines in Fig. 5.2a show how this tree is represented by supplementing leaf nodes in the trie with pointers.

5.3 Linear Programming in a Composite Configuration Space

In our incremental algorithms, a linear feasibility test determines whether the potential successor D-nodes are reachable via feasible door cells. These tests consume most of the running time in the basic incremental algorithm. Related operations such as linear maximizations are required for cell projections and D-node reduction.

We argue that for these operations, it suffices to consider non-negative reference point coordinates. First, assume arbitrary (including negative) coordinates for all reference points and imagine a rectangle (a cube in the case of 3D coordinates) that contains all of the reference points in an arbitrary configuration. We can shift this box back into the first quadrant (octant) to obtain positive reference point coordinates. In other words, any configuration can be represented by reference points with exclusively non-negative coordinates.

Why is it advantageous to use an additional restriction on the reference points? Non-negativity constraints on variables are the implicit default in most linear feasibility test implementations. Allowing for unconstrained variables requires replacing each variable by the difference of two new non-negative variables (this is a standard technique) and thus increases the problem dimension by a factor of two. For instance, when using the simplex algorithm for feasibility testing, non-negative reference point coordinates will considerably speed-up the door tests.

Can we further improve the door tests? It is known that the time complexity of the simplex algorithm is polynomial in most practical settings, and that certain (pathological) examples of constraint systems can be constructed to require an exponential number of simplex iterations. For other approaches (e.g. Karmarkar's algorithm [31]), polynomial worst case upper bounds on the running time have been shown. However, these bounds depend polynomially on the accumulated length of an encoding of all input coefficients rather than on the number of variables and constraints. In general, it is a long standing open theoretical question in the field of linear programming whether there exists a 'genuinely' polynomial-time algorithm for feasibility testing. (The running time of such an algorithm would have to depend polynomially on the numbers of variables and constraints in the worst case.)

In our case, the tested constraint sets have a special structure, and at most four (six) variables are contained in a single constraint. Can we exploit this fact for a more efficient feasibility test? The answer is yes, but only in certain cases. In general, it can be shown that linear constraints with three variables are sufficient to replace any linear constraint, introducing at most polynomial overhead. (To this end,

new variables have to be introduced, and an inequality with d variables is replaced by a collection of inequalities with three variables each. We omit the details here.) A genuinely polynomial-time algorithm for feasibility testing with three or more variables per constraint would therefore immediately imply a genuinely polynomial-time algorithm for general linear feasibility testing. Thus, the fact that our floorgraph constraints have only four (six) variables does not seem to help directly. (For the case of two variables per constraint, an efficient $O(mn^2 \log m)$ feasibility test has been given in [24], where n is the total number of variables and m is the number of constraints.)

There is an interesting special case in which we can indeed further improve the door tests. Assume polygons with axis-parallel segments (polyhedra with orthogonal faces). In this case, pairwise part constraints have the particularly simple form of *difference constraints*:

$$x_i - x_j \geq a, \quad y_i - y_j \geq b, \quad (z_i - z_j \geq c)$$

A system of difference constraints can be represented by a weighted directed graph such that it is feasible if and only if there is no cycle with negative weight in the graph [53]. Feasibility testing can therefore be done in time $O(mn)$, where n is the number of variables and m is the number of constraints [54]. Even further reductions are possible because during the search, we dynamically maintain a set of floorgraph constraints (in the current D-node). When testing potential successor D-nodes, only a small set of constraints (corresponding to a single floorgraph node) has to be added to the current set of constraints. A special property of difference constraints is that incremental addition and deletion of constraints can be done very efficiently [54]: addition of a single constraint takes time $O(m + n \log n)$. Deletion of a constraint can be performed in constant time if the original system is feasible. In our case, constraints have to be deleted only when stepping to a successor D-node. Thus the constraint system will be feasible when we delete constraints. Note that when the constraints become infeasible, we do not delete constraints but simply restart with the constraints from the current D-node (which are known to be feasible) and test another potential successor D-node.

To summarize, we can take advantage of using non-negative variables for the door feasibility tests. Furthermore, if all constraints are difference constraints, linear feasibility testing should be based on efficient incremental algorithms.

Chapter 6

Examples and Experimental Results

We implemented and tested all major algorithms described in the previous chapters. In particular, we evaluated the practicality of our algorithms for efficient linear unboundedness testing, for m-handed separability and for general translational separability. This chapter summarizes results and reports running times for a variety of two- and three-dimensional examples. The experiments show that the algorithms are of practical use and can solve problems that are completely infeasible for other planning approaches such as random planners.

Since the focus in this work is on computations in composite configuration spaces of several parts, we computed complete floorgraphs for all pairs of parts in a preprocessing step. In our example problems, this is possible because the polygons and polyhedra have relatively few vertices. Required times for complete preprocessing range from a fraction of a second to several seconds and are not included.

Note that for both 2- and 3-dimensional examples with only axis-parallel part faces, each embedded linear constraint contains at most two variables (non-zero coefficients). Such constraint systems can be solved with very efficient special-case algorithms [54] (see Chapter 5 for more details). However, we did not optimize our prototypical implementation for orthogonal parts but instead used the simplex algorithm for all examples.

All experiments in this chapter were performed using the following equipment:

- SUN Ultra-60 workstation (300 MHz, 384 MB main memory)
- Solaris 2.6
- Programming languages: C/C++
- Tools: MINOS [46] (simplex feasibility testing), Numerical Recipes in C [14] (singular value decomposition), LEDA [45] (basic data types and geometric computations)

6.1 Linear Unboundedness Testing

We implemented both the simple unboundedness test in Section 4.1.2 and our new algorithm described in Section 4.1.3. Singular value decomposition was used for solving the homogeneous set of equations,

Cell characteristics					Simple algorithm			New algorithm			
#	d	$n^{\geq}$	$n^{=}$	unb	t	f	T	t_h	unb_h	t_f	unb_f
a	40	0	39	yes	0.1	1	0.1	0.2	yes	0.1	no
b	40	0	40	no	7.8	80	7.8	0.2	no	0.1	no
c	200	0	199	yes	10	1	10	20	yes	7.6	no
d	200	0	200	no	60	8	3,000	20	no	8.1	no
e	200	100	0	yes	1.1	1	1.1	6.6	yes	0.8	yes
f	200	500	0	no	976	8	48,800	54	no	119	no
g	500	0	499	yes	171	1	171	399	yes	135	no
h	500	200	400	no	1233	4	308,250	841	no	275	no

Table 6.1: Results of running the simple and new linear unboundedness testing algorithms on randomly generated homogeneous cells defined by d variables, $n^{\geq}$ half-space constraints, and $n^{=}$ hyperplane constraints.

and the simplex method was used for feasibility testing.

To quantify the performance of the new unboundedness testing algorithm with respect to the simple one, we created a series of problems consisting of randomly generated homogeneous cells. Tab. 6.1 summarizes the results. The first five columns (cell characteristics) describe the properties of the randomly generated cells: # the name, d the number of variables (problem dimensionality), $n^{\geq}$ the number of random half-space constraints of the form $\mathbf{a}_i\mathbf{x} \geq 0$, $n^{=}$ the number of random hyperplane constraints $\mathbf{a}_i\mathbf{x} = 0$ and unb whether the cell is unbounded or not. The next two columns (simple algorithm) describe the running time t that was measured for f actually performed feasibility tests. For all bounded cells except (b), we did not run the simple algorithm to completion, since this requires exactly $2d$ feasibility tests (all failing). In these cases, the following column specifies an estimate $T = 2dt/f$ for the complete running time of all $2d$ tests. The last four columns (new algorithm) describe the running times t_h and t_f of solving the homogeneous equations and performing the feasibility test and whether the cell was found to be unbounded by the respective step (unb_h and unb_f).

The experimental results show that the new algorithm can handle large, high-dimensional problems with many constraints. It is significantly faster on large bounded cells for which the simple algorithm leads to impractical running times (column T in cases d, f, and h). The simple test is only better in special cases (a, c and g) where the set of unbounded directions is a subset of the solutions of the homogeneous equations, and where the number of necessary feasibility tests (attempts to find an unbounded variable) is small. Note also that for the new algorithm, the shorter running times for the feasibility test suggest it can be advantageous to first perform this test before solving the homogeneous equations.

In our algorithms for general translational separability, the unboundedness test will operate almost exclusively on bounded cells. (The search can stop when the first unbounded cell is found.) It is therefore especially important that the test runs fast on bounded cells but does not miss any unbounded direction.

6.2 M-handed One-shot Translational Separability of Polygons

The experiments in this section follow our classification in Section 2.2.3 which subdivides the problem instances according to the number of pairs of parts that are separated by a straight line in the initial placement:

- **Only separated pairs of parts.** We showed that under this condition, the parts can always be separated by an 'explosion-like' simultaneous translation (cf. Section 2.2.3). During such a motion, all pairs of parts are translated away from each other. The required computations to determine the individual part motions are quite simple and straight-forward and were therefore not evaluated in practice.

- **Only non-separated pairs of parts.** This condition requires rather interlaced initial placements which can be especially complex to separate for the human and may imply surprising solutions. However, this subclass of the general m-handed separability problem can be handled efficiently with algorithmic approach. Examples and results of applying our polynomial-time algorithm in Section 2.2.3 are summarized in Section 6.2.1.

- **Examples with both non-separated and separated pairs of parts.** This class comprises instances that make the general m-handed separability problem NP-hard (cf. Section 2.2.6). In Section 6.2.2, we illustrate by means of two very similar examples, one separable and the other not, that general high-level configuration space based heuristics can be used to decide m-handed separability of placements with both non-separated and separated pairs of parts.

6.2.1 Examples With no Initially Separated Pair of Parts

We report experimental results obtained with a program that uses our worst-case polynomial time algorithm in Section 2.2.3.

The example in Fig. 6.1 shows a planar assembly in which no pair of parts is separated in the initial placement (top). The program establishes that four of the five parts must be translated simultaneously with distinct velocities and into different directions. The computed part velocities and magnitudes are indicated by arrows in the figure. A sequence of three snapshots (lower pictures) illustrates such a simultaneous one-shot separating translation. In this example, unboundedness testing takes a small fraction of a second, because the dimension is rather small and we have a single convex cell to test (no initially separated pair of parts). By examining subassemblies, the program also determined that no m-handed disassembly translation with less than four hands is possible.

A second example of a planar assembly with no initially separated pairs of parts is shown in Fig. 6.2. Because of its construction, it is easy to augment the number of parts. For an instance with 16 moving parts (not shown), the program took less than a second to find a one-shot separating translation.

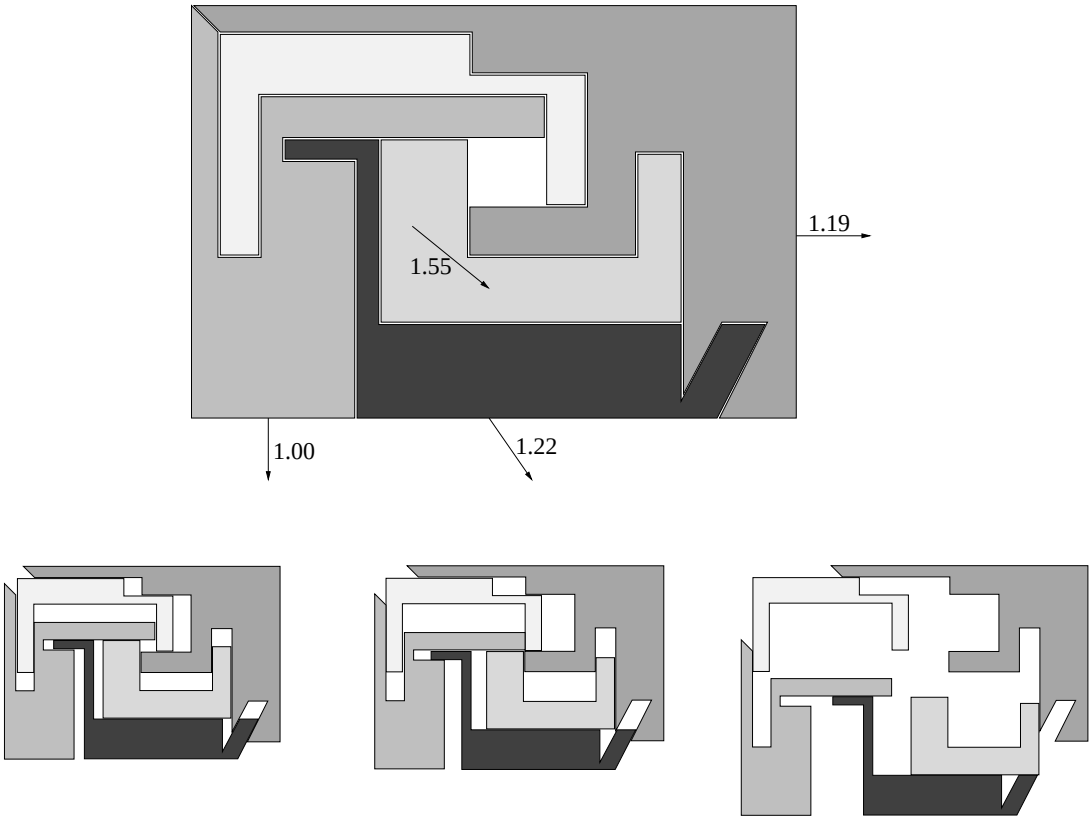


Figure 6.1: Parts can be separated by the motion indicated by arrows. Lengths of arrows (values shown with arrows) represent relative velocities.

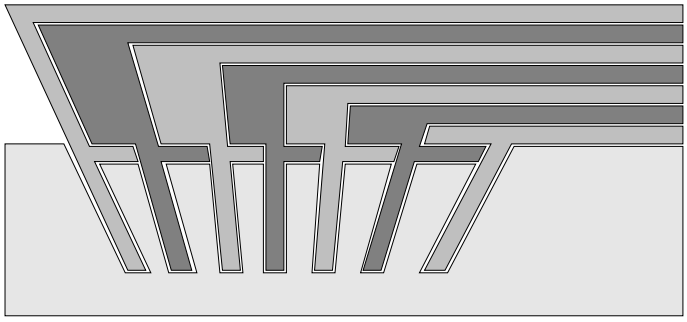
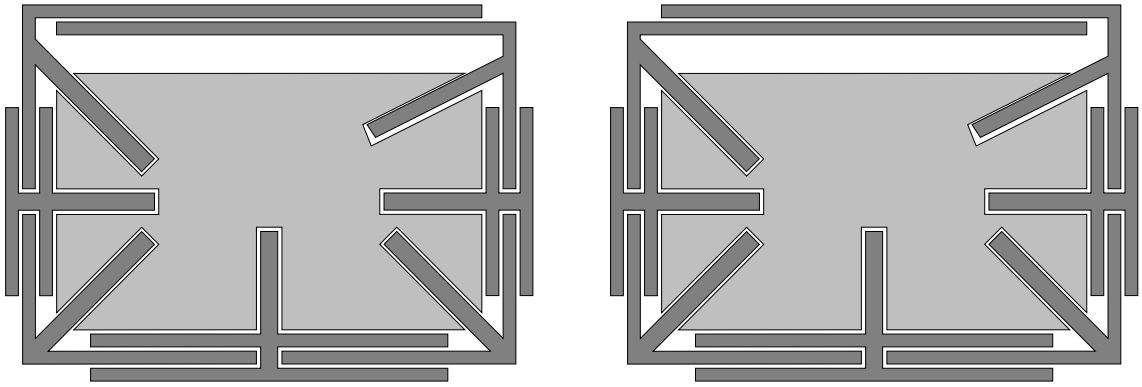


Figure 6.2: Example with no initially separated pair of parts that can be easily extended to an arbitrary number of parts.



(a) (b)

Figure 6.3: Examples for reduction by base cell projection.

6.2.2 Examples With Initially Separated and Non-separated Pairs of Parts

Fig. 6.3 shows two nearly identical eight-part assemblies with 14 separated pairs and 14 non-separated pairs, respectively. While the placement on the left can be separated by a simultaneous one-shot translation, the parts on the right interlock. In both cases, the nonconvex constraints from all separated pairs of parts can be eliminated using the base cell projection heuristics (as described in Section 2.2.4). Thus, in fact only a single convex cell has to be tested for unboundedness in each of the examples. The tests determine that the placement on the left can be disassembled, but not the one on the right. The total running time is about $0.55s$, consisting of $0.5s$ for base cell projections and removal of nonconvex constraints, and $0.05s$ for unboundedness testing of a single high-dimensional convex cone. For the infeasible case in Fig. 6.3.b, a naive approach based on exhaustive analysis of the combined pairwise constraints would have to test 2^{14} cells for unboundedness, resulting in a running time of almost $180s$ (under the favorable assumption that a single unboundedness test takes only $0.01s$).

6.3 Configuration Space Expansion for Polygons

We summarize experimental results obtained with the algorithms in Chapter 3 for several polygonal placements with different properties. Efficient unboundedness testing as discussed in Chapter 4 was used for testing cells during the search and finding translational separating motions. In several experiments, the complete reachable free regions in composite configuration space were expanded by running the algorithms without testing cells for unboundedness.

Fig. 6.4 and Fig. 3.2 (page 42) show the examples, and Tables 6.2,6.3 summarize the results of this section. Tab. 6.2 comprises only separable examples, and Tab. 6.3 contains both separable and non-separable examples for which the complete reachable configuration space was expanded. Two rows of numbers are given for each single example. The upper row contains the measurements for the basic incremental algorithm in Tab. 3.1 (p. 54), and the lower row contains the measurements for the improved

incremental algorithm in Tab. 3.3 (60) using triple based D-node reduction. The columns comprise the number of nodes in the search tree T, the number of door feasibility tests D, the number of generated, but already visited successor D-node candidates V, and the running time t (in seconds). Note that the total number of generated successor D-node candidates is $D + V$, and that $D - T$ is the number of infeasible or not reachable successor candidates. Note also that the running times do not comprise preprocessing, that is, complete computation of floorgraphs.

Section 6.3.1 summarizes results for Figs. 6.4.a–e, including some variations. Section 6.3.2 discusses Fig. 6.4.f, and Section 6.3.3 reports results for examples in Chapter 3 (Fig. 3.2, p. 42). Finally, Section 6.3.4 evaluates heuristics for advanced D-node reduction by construction.

6.3.1 Simple Experiments

We first illustrate the practicality of our concepts from Section 3.3 (*C-space Expansion, Part I*) by computing general translational separating motions as well as proving infeasibility for polygonal placements with very small clearances between parts.

The simple example in Fig. 6.4.a shows a container and two boxes, none of which can be removed. Our basic incremental algorithm can prove this fact rapidly. Row C1 in Tab. 6.3 shows the resulting size of the search tree T, the number of door feasibility tests D and the number of already visited D-nodes V (for which no door tests are necessary). By changing the container as indicated by dashed lines in the figure, the black box can be removed after the dark grey box has been moved into the enlarged cavity (see row S1 in Tab. 6.2). As in the above infeasible case, the improved algorithm reduces the number of cells to handle and runs faster than the basic algorithm despite the overhead of D-node management. This first example is particularly simple and, due to the few degrees of freedom (two moving parts), it could probably also be solved by computing a complete arrangement of hyperplanes in composite configuration space.

The example in Fig. 6.4.b is much more difficult because it involves six moving parts (the container is fixed). It is doubtful if a complete arrangement of valid and forbidden cells, bounded by contact hyperplanes, can be computed explicitly within reasonable time in this case. For six moving polygons, the composite configuration space has dimension 12. Rows S2 and S3 in Tab. 6.2 show the experimental results of our incremental algorithms. In S3, we used a simple heuristic ordering $\mathcal{H}$ for the successors of a D-node: the search continues with the successor that is closest to the current placement in the current D-node. (Note that when switching to a successor D-node, the door test computes a point contained in the door cell and therefore in the successor.)

The examples in Figs. 6.4.c,d are almost identical. In both cases, five of the six parts are free to translate. While the parts in (c) can be separated, (d) cannot be separated without causing overlap, because the peg of the right interior part has been slightly extended. For our incremental algorithms, the examples are rather simple yet, because the reachable component of configuration space contains only relatively few cells. Rows S4 and S5 in Tab. 6.2 show the measurements for computing a sequence

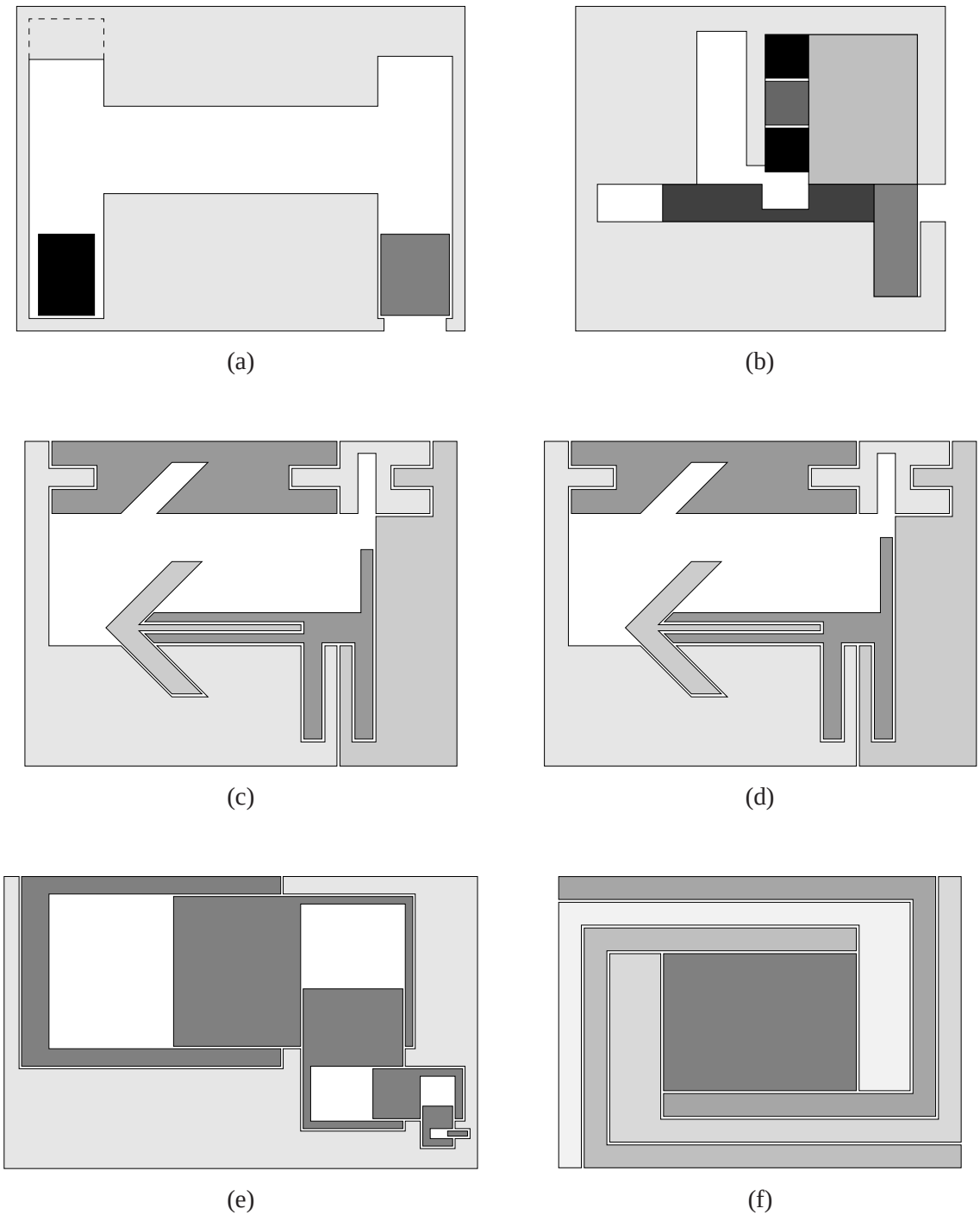


Figure 6.4: Examples for configuration space expansion and translational separability. Note the difference between (c) and (d).

Computation of Separating Translations

#	Example	Heur.	T	D	V	t[s]
S1	Fig. 6.4.a[1]	–	81	520	134	0.91
			64	231	64	0.63
S2	Fig. 6.4.b	–	284	8,582	415	51.4
			149	2,579	149	15.4
S3	Fig. 6.4.b	$\mathcal{H}$	195	4,123	118	25.1
			114	2,002	98	11.9
S4	Fig. 6.4.c	–	299	4,246	293	19.7
			76	380	45	5.55
S5	Fig. 6.4.c	$\mathcal{H}_0$	126	1,519	69	7.5
			47	239	18	3.42
S6	Fig. 6.4.e[2]	–	232	4,391	262	35.1
			141	1,538	134	14.3
S7	Fig. 6.4.e[2]	$\mathcal{H}_0$	204	3,066	133	24.7
			112	949	67	9.52
S8	Fig. 6.4.f[3]	–	46	129	18	0.34
			46	84	16	0.41
S9	Fig. 3.2.a, p. 42	$\mathcal{H}_0$	607	4,531	256	24.9
			117	731	48	5.24

Table 6.2: Computing separating translations. Results from basic algorithm (upper rows) and improved algorithm with triple-based D-node reduction (lower rows). T size of search tree, D number of door tests, V number of already visited nodes. [1]enlarged cavity in container as indicated by dashed lines in the figure, [2]slightly modified to allow for disassembly, [3]without center part.

Complete Expansion of Reachable C-space

#	Example	Separable	T	D	V	$t[s]$
C1	Fig. 6.4.a	no	48	533	177	0.83
			42	271	109	0.67
C2	Fig. 6.4.d	no	55	2,366	177	10.1
			10	132	14	1.29
C3	Fig. 6.4.e	no	216	9,113	798	61.7
			111	2,225	301	17.6
C4	Fig. 6.4.f	no	1	10	0	$\approx$0
			1	10	0	$\approx$0
C5	Fig. 3.2.b, p. 42	no	18,600	701,843	136,681	3,255
			6,935	154,526	40,241	963
C6	3 boxes	yes	48	119	169	0.11
			36	37	155	0.05
C7	4 boxes	yes	1,344	7,775	8,353	10.3
			619	786	5,164	2.57
C8	5 boxes	yes	71,160	782,519	640,681	1,554
			17,749	28,580	231,810	251
C9	Fig. 3.2.c[4], p. 42	no	103,254	2,687,333	851,895	9,280
			13,722	169,829	69,847	925

Table 6.3: Complete expansion of reachable C-space. Results from basic algorithm (upper rows) and improved algorithm with triple-based D-node reduction (lower rows). T size of search tree, D number of door tests, V number of already visited nodes. [4]black bolt fixed.

of translations that separate (c). In row S5, we used another heuristics $\mathcal{H}_0$ to sort successor D-nodes according to the distance of their associated points from the initial placement (the origin in the composite configuration space). Row C2 in Tab. 6.3 shows that for the infeasible variant, D-node reduction (lower row of numbers) significantly reduces the number of cells to test and therefore the improved algorithm performs much faster than the basic algorithm despite the overhead for D-node maintenance.

Fig. 6.4.e shows an example of a latch assembly almost identical to the one in [73]. The difference is that our example is non-separable by modification of a small detail. Results for our variant are in row C3 in Tab. 6.3. The motions required to disassemble the original parts are non-monotone, since each of the parts must be brought into intermediate placements. We also analyzed the original placement which is separable. Measurements for finding a sequence of separating translations are in rows S6 and S7, where S7 uses the heuristics $\mathcal{H}_0$ described above.

To summarize, in nearly all examples the concept of reduced D-nodes showed great improvements both on the number of cells and the running times (despite overhead for maintenance).

6.3.2 Experiments With Combinatorial C-space Boundaries

The following examples illustrate that, using an implicit linear constraint formulation of cells instead of explicitly computing their boundaries and vertices, our algorithm avoids exponential overhead.

The first example in Fig. 6.4.f is a generalization of the example in Fig. 3.4 (page 46) to two feasible degrees of freedom for each part. Note that the parts interlock, and a series of similarly interlocking assemblies with growing number k of parts can be constructed by recursively replacing the inner rectangle with an appropriately scaled copy of the entire assembly. Since each pair of parts is separable (when considered alone), each of the pairwise floorgraphs consists of several nodes. However, only a single (the initial) D-node is examined, since none of its successor candidates is reachable. The total number of arcs in all floorgraphs is polynomial in k. Assuming linear feasibility testing of the door cells is polynomial, we obtain a polynomial time bound for performing a complete search. To show that in these examples, the boundary of the free (d-dimensional) component C containing the origin has exponential complexity, we observe that the tolerances allow for placing each inner part into two distinct corners of the part around it. Selecting one such corner for each of the inner parts corresponds to one d-dimensional vertex on the boundary of C. This gives rise to 2^{k-1} distinct vertices reachable from the origin. (The vertices constructed in this way are only a subset of the total vertex set of C.) Experimental results for the placement in (f) are in row C4 in Tab. 6.2. Further experiments with generalized instances (having 9 and 17 parts, as described above) are in good agreement with the above argumentation (see Tab. 6.4).

Interestingly, after deletion of the central part, the placement in Fig. 6.4.f can be separated by a sequence of translations (see row S8 in Tab. 6.2). Since no pair of parts is separated by a straight line in the initial placement, we also ran our efficient polynomial-time algorithm for m-handed separability in Section 2.2.3. The result is that the parts (without the central box) can also be separated by an m-handed one-shot translation.

k	t[s]	t/k^5[s]
5	0.6	$1.9 \cdot 10^{-4}$
9	3.2	$5.4 \cdot 10^{-5}$
17	19.3	$1.3 \cdot 10^{-5}$

Table 6.4: Computing times (in seconds CPU-time) for establishing infeasibility of the example in Fig. 6.4.f and of generalizations to 9 and 17 parts. The latter placements can be obtained by recursively replacing the central box with a downscaled copy of the original.

6.3.3 Experiments With Combinatorial Examples

The greatest improvements by D-node reduction were obtained for inherently combinatorial examples. We ran our algorithms on the examples in Fig. 3.2 (p. 42) and on random overlap-free initial placements of unit squares ('boxes').

***PARTITION* Examples.** Two simple instances reducing the *PARTITION* problem for the sets $\{1, 3, 3, 5, 6\}$ and $\{2, 2, 2, 6, 6\}$ are shown in Figs. 3.2.a,b (page 42). The numbers are represented by blocks of corresponding lengths. While the first set can be partitioned, the second cannot. As a consequence, the black bolt in the example (a) can be removed after properly rearranging the blocks in two rows within the space on the left inside the container. In (b) however, no such arrangement is possible. Thus, the reachable free configuration space regions comprise all permutations of the blocks within the space between the container and the lock.

Row S9 in Tab. 6.2 shows that using the heuristics $\mathcal{H}_0$ (which sorts the successor D-nodes according to their distances from the initial placement), our algorithms are capable of quickly finding the right critical placement of the blocks in the free space on the left to remove the bolt (case (a)). In the infeasible case (b), the numbers of cells and running times are much higher due to the inherent combinatorics (row C5 in Tab. 6.3). However, the number of costly door tests could be reduced by a factor of 4.5 by using D-node reduction.

Towers of Hanoi Simulation. We computed the complete reachable configuration space for the example in Fig. 3.2.c (p. 42). If the number of U-shaped parts were large enough, the rules of the well-known *Towers of Hanoi* nearly had to be followed to remove the black lock (cf. [49] for a discussion of the example). In the general case, such a construction is clearly impossible to handle with a complete algorithm. We therefore ran our algorithms on the quite simple version shown in the figure and evaluated the quality of the cell decomposition and the improvements obtained by D-node reduction. In our experiments, both the container and the black bolt were fixed. Thus, the program computed the complete reachable free configuration space regions (placements in which the U-shaped parts are in the container). Row C9 in Tab. 6.3 shows the results. Simple triple-based D-node reduction cuts down the number of door tests by a factor of almost 16, and improves the running time by a factor of almost 10.

Unit Boxes. To obtain further examples with cluttered, combinatorial configuration spaces, we ran the program on overlap-free placements of unit boxes in the plane. Unboundedness testing was disabled during the search, thus the algorithms computed a complete convex cell decomposition of free configuration space. (Note that the quite restricted coordinated motion planning problem for an arbitrary number of rectangles that are allowed to translate only horizontally and vertically within a rectangular workspace is PSPACE-complete [25, 27].)

Rows C6–C8 in Tab. 6.3 show the results. Recall that T is the number of D-nodes in the search tree, D is the number of door (feasibility) tests and V is the number of already visited candidate successor D-nodes. (The latter are generated as successor candidates, but not further examined after determining that they have been visited before.) The total number of examined D-nodes is $D + V$. Only feasible and reachable D-nodes are inserted in the search tree, so $T < D$. Note that the number of failed door tests equals $D - T$. The last column contains the total running times for the search (including D-node maintainance). Note that although all non-empty D-nodes are reachable (boxes can be brought into any overlap-free placement), we have $T < D$ because by switching floorgraph nodes, the algorithm also generates infeasible D-nodes yielding empty cells.

Although we used only the simple triple based elimination scheme, significant improvements over the basic algorithm can be observed. Especially the number of expensive feasibility tests was reduced greatly. To compare the basic algorithm and the improved algorithm with D-node reduction, we define the ratio of examined nodes

$$R = \frac{V_2 + D_2}{V_1 + D_1}$$

where the values in the numerator come from the improved algorithm while the values in the denominator come from the basic algorithm. In our experiments, R decreased from $R = 0.67$ for three boxes to 0.38 for four boxes and 0.17 for five boxes. This suggests that R is reduced by half for each additional box and the D-node reduction becomes increasingly more effective for more parts.

It can be observed in general, that the effect of D-node reduction becomes stronger both for examples with more combinatorial freedom and for increasing number of parts (cf. results from the previous subsections). Note that the parts must move in a space of small constant dimension (the plane). We conjecture that the average number of constraints required for forming reduced D-nodes depends linearly on the number of parts. This will be supported by the experimental results in the next subsection.

6.3.4 Heuristic Incremental Cell Construction

We illustrate the approach of D-node reduction by cell construction as proposed in Section 3.4.4.2. Since optimal D-node cell construction is intractable (cf. Section 3.4.3.1), we evaluate simple heuristics that aim at finding a good order in which the individual pairwise constraints should be added to produce a cell that is as unconstrained as possible.

Heuristic cell construction for a given placement of parts starts with an initially empty set of constraints. At each step of the construction, one or more best suited constraint hyperplanes (according the

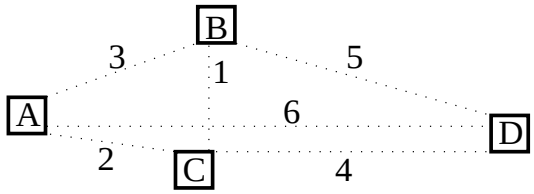


Figure 6.5: Random placement of four boxes illustrating the CF heuristics. The order in which pairwise constraints are considered for construction of reduced D-node cell depends on the pairwise part distances.

heuristics) from a single pair of parts are added to the cell description, and the obstacle faces behind them are marked as shadowed. The construction is complete when the cell around the given configuration contains only intersection-free placements of all parts. Since in our experiments parts were convex, it was sufficient to consider a single constraint hyperplane per pair of parts. Note that we consider only hyperplanes supported by the faces of the pairwise configuration obstacles. The following simple heuristics to select constraint hyperplanes were implemented and tested:

CF (Closest Face) Choose a constraint hyperplane whose supporting obstacle face is closest to the current configuration.

CH (Closest Hyperplane) Chose a constraint hyperplane which is itself closest to the current configuration.

RH (Random Hyperplane) Randomly choose a pair of parts and add any (random) valid hyperplane supported by the configuration obstacle of this pair. This is not really a heuristics, but serves as a benchmark to assess the quality of the two above heuristics. Furthermore, this strategy will show that even totally uninformed incremental cell construction is much better than considering constraints from all pairs of parts.

We performed experiments with random overlap-free sample placements of unit boxes, triangles and simple octagons. (In a single such placement, all parts have the same shape.) The focus here is on placements with many parts rather than placements of parts with complex shapes. We show that heuristics can be helpful to extract good (small) subsets of required constraints from the set of all pairwise constraints. Note that although the objects considered here are convex and simple, the resulting composite configuration spaces are highly non-convex and cluttered. (Assume a random direction in composite configuration space and simultaneously translate the objects, starting at a random initial placement; collisions will result in almost all directions.) For an illustrative example, consider Fig. 6.5 (showing only a few boxes). The first pair of boxes suggested by the CF heuristics would be (B, C) because this is the closest pair. As a consequence, the linear inequality $y_B - y_C \geq c_1$ is added to the cell representation as the first constraint. Thereafter, the pair (A, C) would be selected, introducing a constraint of the form $x_A - x_C \leq c_2$, and so on. Note that not all six pairs will have to contribute an explicit constraint on their positions.

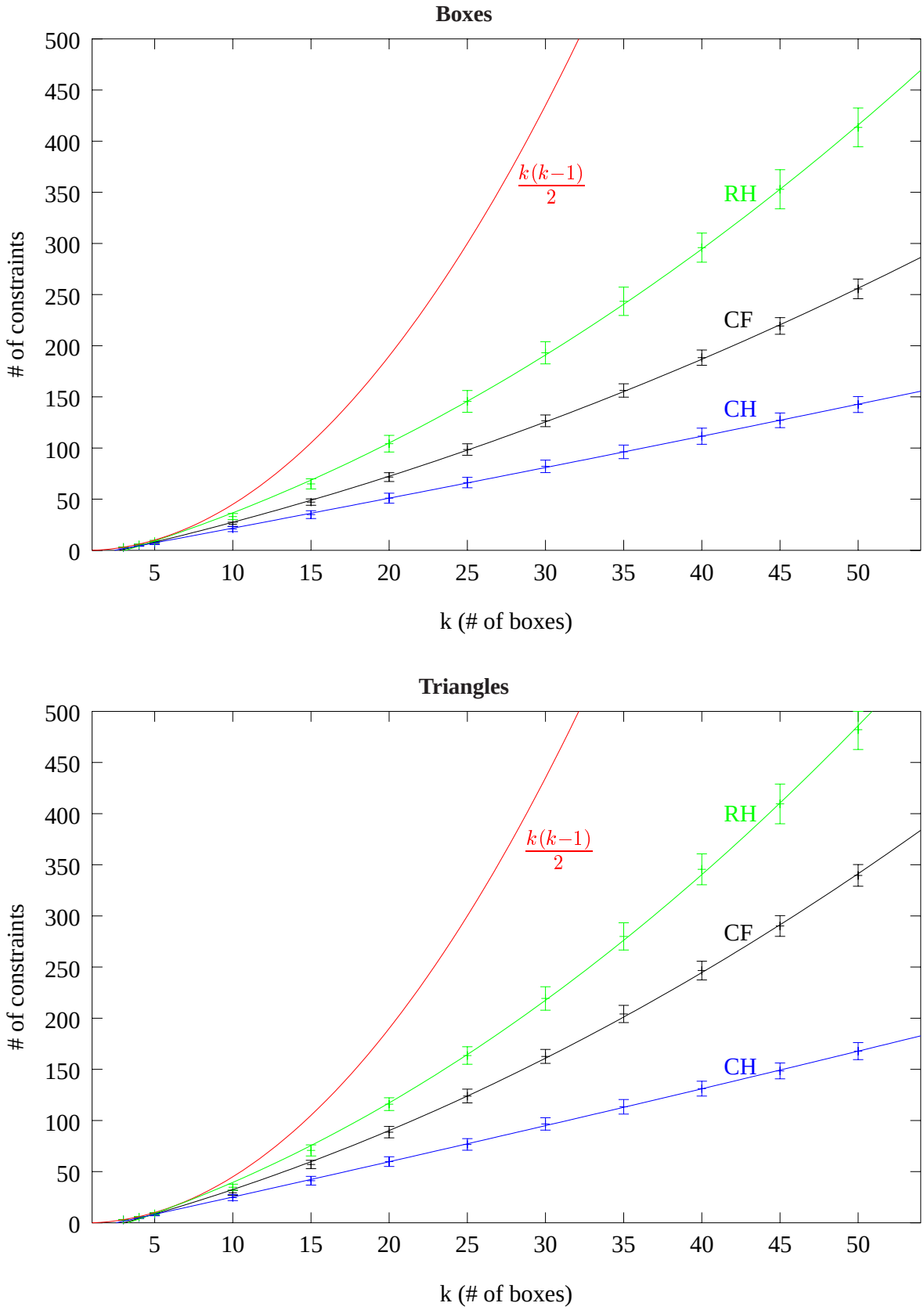


Figure 6.6: Data points: average value and standard deviation of cell description sizes for 100 random placements of k parts (top: unit boxes, bottom: triangles) in terms of number of constraints for different selection heuristics: CH (closest hyperplane), CF (closest face) and RH (random hyperplane). Curves: best fit of $ak^2 + bk + c$ (cf. Tab. 6.6). The curve $k(k-1)/2$ represents the upper bound.

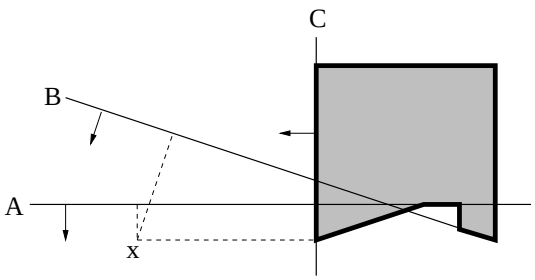


Figure 6.7: Closest constraint heuristics may generate redundant constraints.

	Boxes			Triangles		
k	CH	CF	RH	CH	CF	RH
3	2.68 (0.469)	2.68 (0.394)	2.68 (0.302)	2.76 (0.429)	2.82 (0.386)	2.93 (0.256)
4	4.74 (0.787)	4.74 (0.783)	4.74 (0.659)	5 (0.725)	5.35 (0.783)	5.6 (0.569)
5	6.98 (1.32)	6.98 (1.03)	6.98 (1.06)	7.86 (1.03)	8.43 (1.09)	8.95 (1.01)
10	20.8 (2.53)	20.8 (2.07)	20.8 (2.99)	24.6 (2.96)	29.3 (2.47)	34.8 (2.98)
15	34.9 (3.83)	34.9 (3.08)	34.9 (4.92)	41.1 (4.24)	57 (4.07)	70.8 (5.45)
20	51 (4.88)	51 (4.35)	51 (8.14)	59.8 (4.71)	88.6 (5.58)	116 (6.22)
25	66.2 (5.08)	66.2 (5.62)	66.2 (10.7)	76.6 (5.7)	124 (6.76)	164 (8.55)
30	82.1 (6.08)	82.1 (5.75)	82.1 (10.9)	96.7 (6.03)	163 (6.85)	219 (11.5)
35	96.3 (6.65)	96.3 (6.52)	96.3 (13.8)	113 (7.13)	204 (8.39)	280 (13.3)
40	112 (7.86)	112 (7.47)	112 (14.2)	131 (7.28)	247 (9.12)	346 (15.1)
45	127 (7.18)	127 (8.14)	127 (19.1)	148 (7.75)	290 (10.1)	410 (19.4)
50	143 (7.77)	143 (9.65)	143 (18.9)	168 (8.29)	340 (10.6)	482 (19.3)
50	143 (7.77)	143 (9.65)	143 (18.9)	168 (8.29)	340 (10.6)	482 (19.3)

Table 6.5: Average value and standard deviation (in parentheses) of cell description sizes (numbers of constraints) for 100 random placements of k boxes/triangles using different selection heuristics: CH (closest hyperplane), CF (closest face) and RH (random hyperplane).

	Boxes	Triangles
CH	$3.3 \cdot 10^{-3}k^2 + 2.8k - 6.9$	$3.9 \cdot 10^{-3}k^2 + 3.3k - 8.5$
CF	$4.1 \cdot 10^{-2}k^2 + 3.3k - 9.3$	$6.5 \cdot 10^{-2}k^2 + 3.8k - 12$
RH	$8.8 \cdot 10^{-2}k^2 + 4.2k - 14$	$1.1 \cdot 10^{-1}k^2 + 4.4k - 16$

Table 6.6: Best fit of $ak^2 + bk + c$ to the data in Tab. 6.5.

We created 100 random overlap-free placements for $k = 3, 4, 5, 10, 15, \ldots, 50$ parts and constructed D-node cells around each single placements using each of the above heuristics. Tab. 6.5 contains the resulting average cell description sizes (in terms of required pairwise separating hyperplanes) and the empirical standard deviations in parentheses. To assess the dependence of the average cell description size on k (the number of parts), we used standard least squares curve fitting of a quadratic template $f(k) = ak^2 + bk + c$ to the measurements. Fig. 6.6 shows the data points, the resulting curves and error bars indicating the standard deviations of the data. Tab. 6.6 confirms that the curves for CH are almost linear. We performed the same experiments with octagons, resulting in the same qualitative curves (not shown) and an almost linear dependence on k. The constant factors appear to increase from boxes over triangles to octagons.

These results suggest that even in very cluttered configuration spaces in which every pair of parts is a potential candidate for collisions, incremental cell construction (together with simple heuristics) can reduce the number of required constraints from quadratic to linear. It is clear that already for a few parts, a sample size of only 100 configurations is very small. We performed limited experiments with sampling sets of 1000 configurations (and less than 40 parts) and observed that the linear dependence on the number of parts was preserved.

Note that CH may, although apparently superior on average, yield redundant constraints in the general case. This is illustrated in Fig. 6.7 (the current configuration is indicated by an x). Although A and B are closer than C, they are redundant because C is required in any case. To show that CH is not optimal even for orthogonal objects, we also performed experiments in which the boxes were arranged with some clearance between them to form a regular square. For these arrangements, CF performed better than CH, showing that CH is not optimal. However, Fig. 6.6 strongly suggests that on average, CH is the best heuristics in the above scenario. Future work should investigate constraint selection heuristics for more complicated part shapes and placements as well as the three-dimensional case. For placements with interlaced non-convex parts, we conjecture that CF yields better results.

6.4 Spatial Examples for General Translational Separability

We tested our algorithms for general translational separability in Chapter 3 with several more realistic spatial examples. To show the special importance of fast and complete unboundedness testing, we incorporated both the simple algorithm from Section 4.1.2 and our new algorithm from Section 4.1.3 and compared their performances. Running times were recorded for the following two tasks:

- **Search for the first unbounded cell.** The search stops when the first unbounded cell is found, that is, at least one part can be removed from the remaining parts. Tab. 6.7 summarizes the results. The columns specify the problem name, the number of parts, the number of cells tested for unboundedness during the search, the times of the simple and new unboundedness test for all cells, and the time required for for the search (includes door feasibility tests).

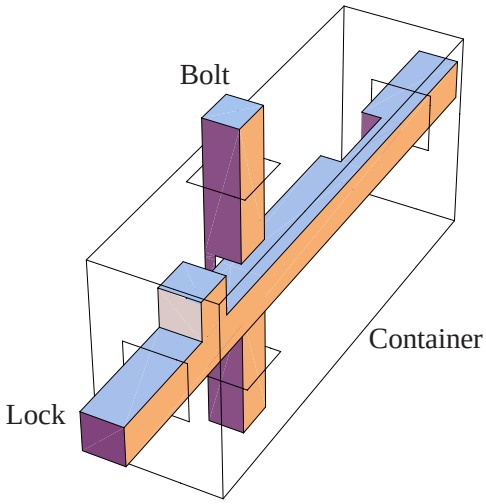


Figure 6.8: To decide whether the bolt can be removed, reachable placements of the lock must be examined (example from the introduction).

- **Complete disassembly.** Having found an unbounded cell, the problem is split into separable subsets and the subproblems are processed recursively until they consist of single parts or cannot be disassembled any further. Tab. 6.8 summarizes the cumulative running times for complete disassembly.

6.4.1 Lock and Bolt

As a first very simple example, we revisit the lock/bolt mechanism from our introduction (cf. Fig. 6.8). The bolt can be removed if and only if the notch of the lock can be brought into alignment with it. Due to the peg, the lock can only move back and forth in a limited range within the container. We created two instances of this construction in which the notch of the lock is at two different locations. Only one of them allows for removal of the bolt. Note that although the parts are quite simple, complete configuration space computation (direct arrangement computation) can be quite costly. Our basic incremental algorithm from Chapter 3 solves both examples almost instantly by proving infeasibility in one case and determining separating translations in the other case. Since the lock cannot be removed from the container, we did not consider complete disassembly here. The running times are within fractions of a second in both cases.

6.4.2 Wooden Cube Puzzle

We revisit the 12-part wooden cube in Fig. 3.1 (p. 41). A total of 38 unboundedness tests were necessary to find a removal sequence for the first subset of parts (Tab. 6.7). Our new unboundedness test performs almost six times faster than the simple algorithm in this example. For complete disassembly, the required non-monotone sequence of translations was found by our program in about 13 seconds. About 2 seconds

Example	Parts	U-tests	Simple	New	Search
Cube	12	38	5.91	1.00	7.24
Bookcase	17	147	50.71	9.82	14.83
Desk1*					
(not decoupled)	9	504	25.17	2.61	7.64
(decoupled)	2×5	25	0.37	0.03	0.18
Desk2†					
(not decoupled)	9	256	11.14	1.23	13.09
(decoupled)	2×5	32	0.50	0.08	0.52

Table 6.7: Searching for first unbounded cell (*can be disassembled, †cannot be disassembled). Total running times (in seconds) are the sum of the numbers in columns Simple and Search or New and Search (according to which unboundedness test was used).

Example	Parts	U-tests	Simple	New	Search
Cube	12	132	12.42	2.05	10.94
Bookcase	17	277	55.82	10.42	16.23
Desk1*					
(not decoupled)	9	960	37.10	3.86	10.35
(decoupled)	2×5	118	1.24	0.10	0.54

Table 6.8: Complete disassembly (*can be disassembled). Total running times (in seconds) are the sum of the numbers in columns Simple and Search (using the simple unboundedness test) or New and Search (using our new unboundedness test).

Example	Simple		New		Search	
Cube	22.59	(12.42)	3.34	(2.05)	33.85	(10.94)
Bookcase	115.31	(55.82)	16.62	(10.42)	77.90	(16.23)
Desk2†	195.93	(11.14)	22.7	(1.23)	1635.18	(13.09)
(not decoupled)						

Table 6.9: Running times for complete disassembly without D-node reduction (†cannot be disassembled). Times using D-node reduction from Tab. 6.8 are shown in parentheses.

are for the new unboundedness test whereas the simple test would have taken more than 12 seconds for all cells (cf. Tab. 6.8). The first row in Tab. 6.9 shows the running times of the algorithm without D-node reduction (for comparison, the results with D-node reduction from Tab. 6.8 are in parentheses). The speed-up factor of about three shows that triple based D-node reduction can be very effective for tightly packed three-dimensional assemblies with many interlaced parts.

6.4.3 Book-case

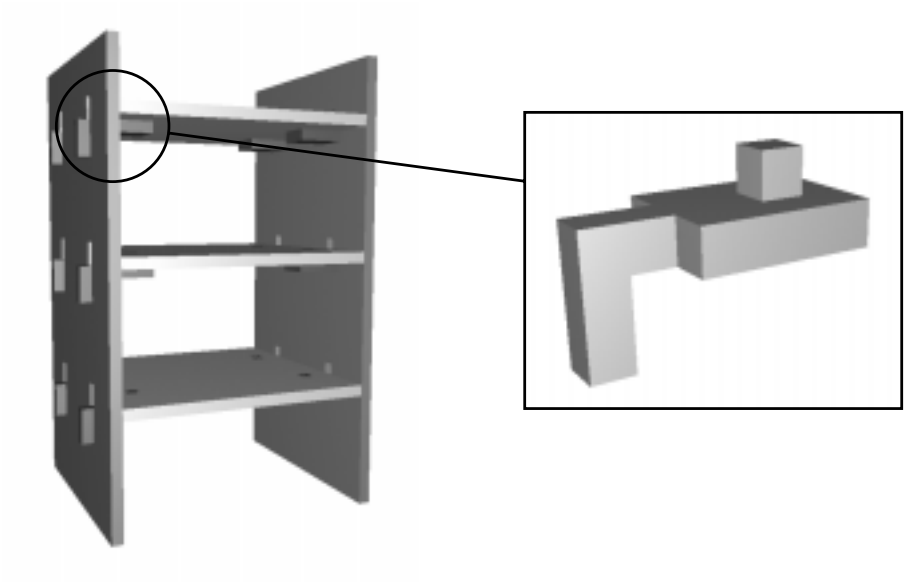
For the 17-part bookcase in Fig. 6.9.a, constraints from many pairs of parts can be eliminated due to redundancy. This is illustrated in Fig. 6.9.b, which shows exploded side view of one half of the bookcase (11 parts instead of 17) and in Fig. 6.9.c, showing the reduced D-node for the initial mounted configuration. Only 22 out of 55 pairwise constraints (edges) are necessary for the 11 parts. For the entire bookcase assembly, the graph representation of the initial reduced D-node contains 50 out of 136 edges. A complete disassembly plan was computed in about 16 seconds for the search and 10 seconds for the new unboundedness test (277 cells were tested for unboundedness). In contrast, the simple algorithm took almost six times longer (56 seconds). Tab. 6.9 shows that D-node reduction with the new unboundedness test speeds up the computation by a factor of about 3.5. Note that for this example, we used the dynamic heuristics with floorgraph node counters as described in Section 3.5 for sorting successor D-nodes.

6.4.4 Desk

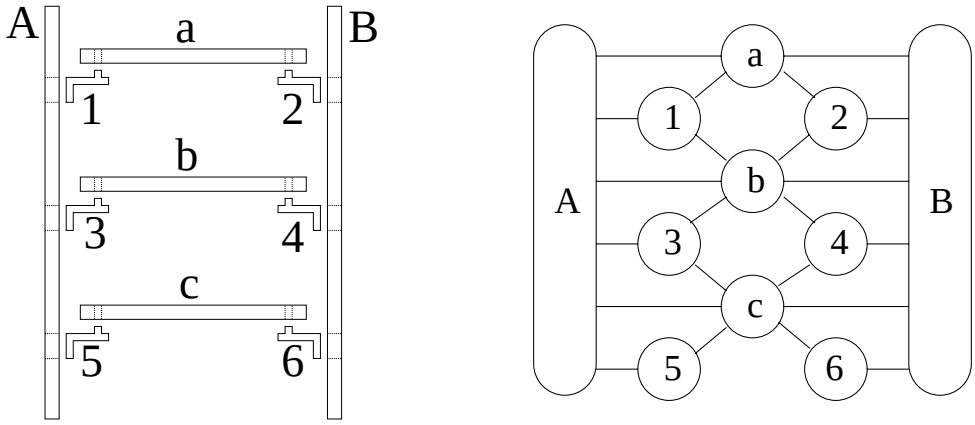
Fig. 6.10.a shows a desk with eight drawers. The container of the drawers is a single part. Each of the drawers slides along two splints, secured against falling out. The drawers must be pulled out, lifted and then pulled further out to be removed. Before a single drawer can be lifted enough for removal, the one above it must have been removed. We created two versions, Desk1 which is demountable and Desk2 which is not demountable. The second model is nearly identical to the first but the back panels of the topmost drawers were heightened so that these drawers could no longer be lifted enough for removal. Consequently, none of the drawers can be removed, although they can all be moved back and forth independently. Fig. 6.10.b shows the initial reduced D-node. The container (C) separates the left (L1–L4) and right (R1–R4) drawers: there are no edges between any L- and R-nodes.

The running times for finding the first unbounded cell and for complete disassembly of Desk1 are shown in Tab. 6.7 and Tab. 6.8 (rows indicated by 'not decoupled'). For comparison, Tab. 6.9 shows that the total running time (using our new unboundedness test) is more than 100 times slower without D-node reduction.

In the case of Desk2 (non-separable), D-node reduction cannot avoid that 2^8 cells are examined since the two cabinets are not checked independently: each of the eight drawers is checked in two positions: pushed in and pulled out. This is exactly the situation described in Section 3.5.3 (Independent Subproblems). Actually, the drawers in the left cabinet cannot interact with the drawers in the right cabinet in

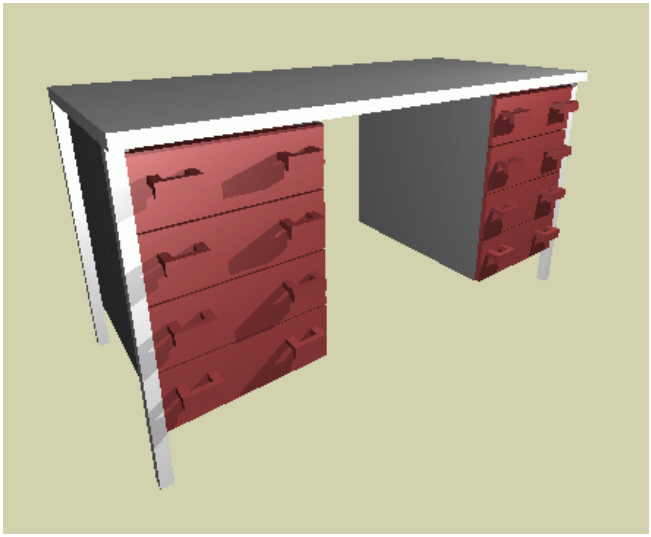


(a)

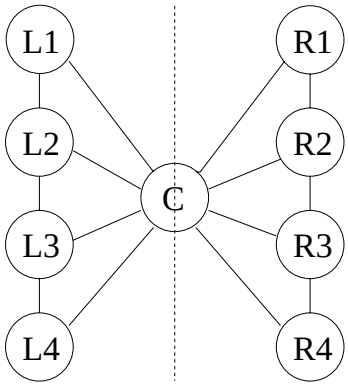


(b) (c)

Figure 6.9: (a) A bookcase that requires several sequential motion direction changes to remove the shelves. (b) Exploded side view (front side only) and (c) graph representation of reduced initial D-node.



(a)



(b)

Figure 6.10: (a) Desk with eight drawers and (b) Initial D-node allowing for decoupling into two subproblems.

any relative placement. Therefore, the problem can be split in two simpler decoupled subproblems (left and right cabinet). We decoupled the left and right cabinets and ran the program twice, once on the left cabinet alone and the second time only on the right cabinet. The resulting running times are included in the tables (rows indicated by 'decoupled' in the leftmost column). The two separate problems with 5 parts each allow for faster detection of infeasibility, resulting in significant speedup (up to a factor of almost 50).

Chapter 7

Conclusion

7.1 One-shot Translational Separability

We discussed the problem of translational one-shot separability for overlap-free placements of polygons or polyhedra. A classification of different subproblems was given, and the problems of m-handed and c-handed separability were considered in detail. To the best of our knowledge, this contribution is the first, except for our own recent work, to delve with the case of multiple moving subsets in the context of simultaneous infinite translations.

In the m-handed case, we started with polygonal parts and derived a linear constraint framework for describing the set of all valid infinite one-shot separating translations. We then presented efficient algorithms for the cases where either all or no pairs of polygons are separated in the initial placement. Since the general m-handed separability problem is NP-complete, we developed effective heuristic preprocessing and a branch-and-bound algorithm to tackle instances with both separated and non-separated pairs. We then proved NP-completeness of m-handed separability of polygons by a reduction from the *PARTITION* problem for integers. We showed how our algorithms and heuristics can be extended to polyhedral parts, and adapted the NP-completeness proof from the polygonal case.

Motivated by the complexity of m-handed separability and the existence of polynomial-time algorithms for single-handed separability, we studied the problem of c-handed separability. There, the number of different subset velocities must not exceed a constant c. We showed that the problem of finding a c-handed separating translation can be reduced to a velocity or direction assignment problem, but both problems are NP-complete for $c = 2$. We consider this an important result, since deterministic polynomial-time algorithms are known for the special case $c = 1$.

For the proof of NP-hardness of two-handed separability of polygons and two-handed velocity assignment for polygons, we needed additional fixture constraints. (In the polyhedral case, no fixture constraints were necessary.) It remains an open question in this work, whether the hardness result can be obtained without these fixture constraints in the planar case.

Many practically motivated extensions may be studied in the future. Note that the basic principles for one-shot separability also apply to curved planar parts, as long as separating directions for pairs of

parts can be derived. Further heuristic preprocessing strategies can be envisaged beyond the proposed base cell projection to simplify various separability problems. From the practical perspective, it would be important to incorporate additional constraints and possibly reformulate the problem as (approximate) optimization of some cost function. For instance, one may be interested in finding the minimum number of hands required to separate a given placement of parts, or the maximum number of parts that can be removed simultaneously. To execute separating motions without costly fixturing mechanisms, it may be required that parts moving with the same velocity form a connected and stable subassembly.

7.2 General Translational Separability

General translational separability is known to be PSPACE-hard. But this fact and a number of related hardness results for subclasses of the problem concern the worst-case complexity. In contrast to this, the complexity of many practically relevant instances is much simpler when we consider only the set actually feasible translations.

We first discussed various general problems of convex cell decompositions in the context of multiple translating parts (polygons or polyhedra), and then proposed a framework for high-dimensional *C-space expansion*. Based on this idea, our methods incrementally compute reachable free regions in the composite configuration space. We then discussed further general defragmentation methods to eliminate remaining redundancies.

Our approach will work best for composite configuration spaces with narrow channels and non-combinatorial properties (in those regions that are reachable from the initial placement). An example with narrow channels but excessive combinatorics that will remain hard for our approach would be the well-known sliding block puzzles. The conditions under which our approach can be successfully used are often met in assembly planning applications. There, we usually have highly coordinated motions of parts and constrained configuration spaces. Combinatorics in assembly planning usually arises when enumerating all possible disassembly sequences. In this work, however, we considered the problem of finding a single separating sequence or proving that no such sequence exists. When parts have been separated into at least two subsets, the configuration space becomes combinatorial, but then the search can stop and can recursively work on the separated subassemblies.

In the above scenario, our approach compares favorably with other planning methods for the following reasons. Our algorithms are exact and complete (under the above assumptions). Random and heuristic planners will inevitably fail in narrow configuration space channels. A great advantage is that our approach can be extended to allow for bounded overlap. This could be used to identify the contact pairs that prevent an almost feasible separating motion. We regard this as a practically very relevant feature. Other exact and complete planning methods either limit the set of allowed translations or are too intricate for a reasonable implementation. Our algorithms have been implemented and evaluated, and are based on subproblems that have been extensively studied in the literature, such as computation of

configuration space for pairs of parts and linear programming.

The following open problem should be studied in the future: unrelated subsets of parts may introduce a combinatorial explosion if it is not obvious how to decouple these subproblems. This was illustrated using simple examples in Section 3.5.3. The main problem is that, since in our approach we combine pairwise configuration spaces, all combinations of convex pieces from independent non-convex subspaces may be feasible and have to be tested in the composite configuration space (for the sake of completeness). Note that only eliminating the redundant constraints (in whatever sense) does not help here. Better ways than the mentioned very simple heuristics have to be found in order to decouple subproblems without loosing completeness.

To conclude, it is hard to find high-level techniques for tackling the problem of translational separability in its full generality. Previous approaches have analyzed various subclasses of the problem by restricting the allowed translations to either single translations, small constant-length sequences of translations or one moving part per translation. Our algorithms are presented in the very general framework of the composite configuration space, under the quite abstract assumption that the set of feasible translations is not too complex. They proved to be practical in many interesting examples for which other implemented methods such as random planners are not suited. A variety of theoretical methods that have been proposed to solve general motion planning problems do not seem to be of practical applicability at all, if the number of independently moving objects is not limited to a small constant.

7.3 Efficient Linear Unboundedness Testing

Efficient linear unboundedness testing is a key to practical algorithms for translational separability. We first defined the problem and pointed out why it cannot be solved using standard methods of linear programming in a straight-forward way. We then proposed a new $O(nd^{4.5})$-time algorithm for unboundedness testing with n constraints and d variables. It improves by a factor of d over the naive approach using $2d$ linear maximizations or feasibility tests. We showed that unboundedness testing is at least as hard as the dominating steps of our new algorithm: solving homogeneous equations and feasibility testing. This suggests that unboundedness testing cannot be performed faster asymptotically unless faster methods for feasibility testing become available.

Allowing disjunctions of constraints, we proved that the problem of finding an unbounded direction is NP-complete already for a maximum of two constraints per disjunction and three variables per constraint. An open question is whether this still holds for at most two variables per constraint.

7.4 Implementation

We discussed some practically relevant subproblems that were omitted in Chapter 3 in order to keep the presentation clear. In particular, we showed

- that non-negativity constraints on all reference point coordinates can be advantageous,

- how visited D-nodes and solution paths can be stored and retrieved efficiently, and
- that pairwise configuration space computation is well-understood and should be performed incrementally during the search if polyhedral parts of high complexity are considered. However, the latter is a topic for future research.

7.5 Experiments and Evaluation

The experimental results show that our algorithms can solve a variety 2- and 3-dimensional translational separability problems. Proving infeasibility and computing the complete set of reachable placements can be more or less effort than finding a removal motion: on the one hand, a non-separable placement of parts may be very constrained, thus involving fewer reachable D-node cells than a separable example. On the other hand, proper heuristics can help finding removal motions quickly while proving infeasibility always requires the expansion of all reachable cells (cf. the *PARTITION* examples). The concept of D-node reduction turned out to be of great importance. We implemented and tested D-node reduction using a triple-based elimination scheme which showed to have a great impact on the numbers of examined cells. Despite the overhead caused by additional maintenance steps, the running times were, in almost all cases, reduced substantially by this cell defragmentation.

When a sequence of separating translations actually existed, both very constrained and very unconstrained problems could be solved rapidly. In the former case, free space is limited, thus forcing the search to follow the right channels. In the latter case, an unbounded cell maybe within reach. It is actually the type of problems in-between which are hard. That is, constrained problems with combinatorial properties, such as *PARTITION* or the *Towers of Hanoi* simulation.

To test different D-node cell construction heuristics for improved cell defragmentation, we discussed our experiments with random sample placements of boxes, triangles and octagons. The results show that the number of actually required constraint hyperplanes can be reduced drastically by incremental cell construction and use of proper heuristics for selection of promising separating hyperplanes. Even for the considered very unconstrained placements, a linear number of constraints appears to be sufficient. There is much room for future work here. Especially for placements with many non-convex parts or realistic assemblies, the effectiveness of the proposed (or similar) cell construction heuristics should be investigated.

Appendix A

Cell Projection

Given a set of d-dimensional constraints $\mathbf{Ax} \geq \mathbf{b}$, compute a linear projection of the set of feasible points to an arbitrary linear subspace (called *projection space*). If the projection space is spanned by a subset of the canonical basis of the input set, an equivalent problem is eliminating all variables whose basis vectors do not contribute to the subspace.

A.1 Convex Hull Method

When the dimension of the projection space is a small constant (in our cases, 2 or 3), we can compute an explicit vertex representation of the projection using the *Convex Hull Method* presented in [28] for parameter elimination. The method combines principles from high dimensional linear optimization with explicit vertex-face representation in low-dimensional projection space. Since the input set is linear and convex, its projection must be a linear convex set, too. We can therefore incrementally construct the projection starting with an initial inner approximation and adding new vertices and faces that are the result of linear optimizations subject to the input constraints.

Growing an Inner Approximation

A facet of the approximation is terminal, that is, it belongs to the projection, if a high-dimensional linear optimization orthogonally outwards to the facet yields a vertex whose projection is contained in the facet. Otherwise, the new vertex is outside the current hull and must be added to the representation, thus generating new facets to be processed next. The algorithm stops when all facets are terminal.

To obtain the initial inner approximation of the projection, we first run a minimization and maximization along an arbitrary basis vector of the projection space. If the extrema coincide, we select another basis vector. (If all basis vectors of projection space yield identical extrema, the projection is a single point.) Next, we minimize in an arbitrary direction that is orthogonal to the segment connecting the two initial vertices. The new vertex is added, and we continue until the approximation is full-dimensional. We continue testing and expanding facets as described above until we have obtained all terminal facets.

In the worst case, the running time of the method is exponential in the dimension of the projection

space, because we explicitly compute all vertices of the projection. The following example illustrates the above principles to obtain projections to a part-relative subspace (which is the case required in our context).

Example: Given the placement of four planar parts in Fig. A.1.a, we get the following pairwise linear constraints:

$$\begin{aligned}(A,B)&: \quad \{0 \le x_A - x_B \le 1, y_A - y_B - x_A + x_B = 0\}\\ (B,C)&: \quad \{x_B - x_C = 0, 0 \le y_B - y_C \le 1\}\\ (C,D)&: \quad \{0 \le x_C - x_D \le 1, y_C - y_D = 0\}\end{aligned}$$

The constraints state that part A can move diagonally in the hole inside B, part B can move vertically within C and C can move horizontally within D. We assume that part C is fixed, i.e. we add $x_C = y_C = 0$. These constraints form our system $\mathbf{Ax} \ge \mathbf{b}$, where $\mathbf{x} = (x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D)$.

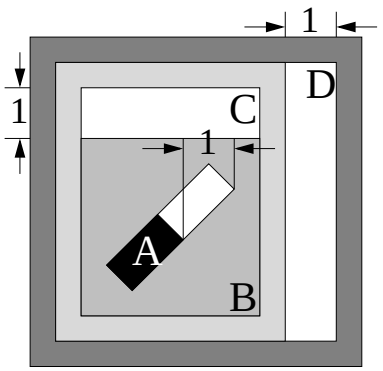
We are interested in the possible relative placements of part A with respect to part D. This amounts to computing the projection of the above constraints to the $(x_A - x_D, y_A - y_D)$-subspace. Note that here, the projection space is not spanned by a subset of canonical basis vectors of the input set.

In a first step, we minimize and maximize $x_A - x_D$ subject to $\mathbf{Ax} \ge \mathbf{b}$ which yields the points $(x_A - x_D, y_A - y_D) = (0, 0)$ and $(x_A - x_D, y_A - y_D) = (2, 1)$, respectively (Fig. A.1.a). Note that both extrema are not unique. In fact, another minimal vertex would be $(0, 1)$ and another maximial vertex $(2, 2)$. The correctness of the following steps is of course independent of the chosen minimum and maximum vertex.

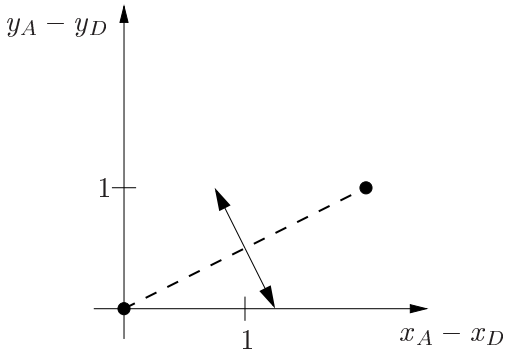
Since we project into a two-dimensional space, the directions for the next minimization and maximization are unique: the unique straight line $(x_A - x_D) - 2(y_A - y_D) = 0$ is determined by the initial two points $(0, 0)$ and $(2, 1)$ (Fig. A.1.b). Minimizing and maximizing $(x_A - x_D) - 2(y_A - y_D)$ yields $(1, 2)$ and $(1, 0)$, respectively. Now we have an initial approximation of the projection, shown in Fig. A.1.c, which is bounded by the constraints

$$\begin{aligned} y_A - y_D &\ge 0\\ -(x_A - x_D) + (y_A - y_D) &\ge -1\\ -(x_A - x_D) - (y_A - y_D) &\le 3\\ 2(x_A - x_D) - (y_A - y_D) &\le 0\end{aligned}$$

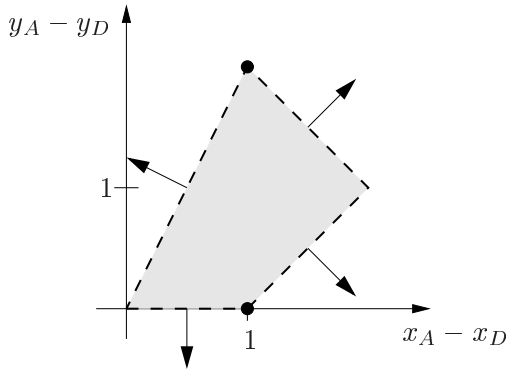
The following three minimizations orthogonally to the bounding straight lines of these constraints reveal that the first two faces already belong to the final projection, and that two new vertices, must be inserted: minimizing $-(x_A - x_D) - (y_A - y_D)$ yields $(2, 2)$ and minimizing $2(x_A - x_D) - (y_A - y_D)$ yields $(0, 1)$. The new approximation (Fig. A.1.d, bold segments indicate terminal facets) is therefore bounded by



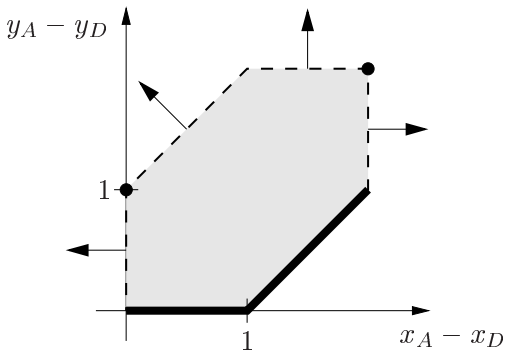
(a)



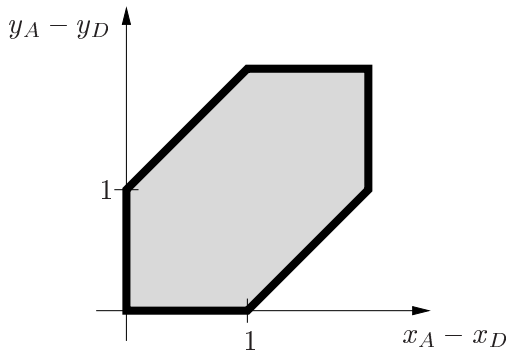
(b)



(c)



(d)



(e)

Figure A.1: Example for cell projection by growing an inner approximation using the Convex Hull Method.

$$\begin{aligned} y_A - y_D &\geq 0 \\ -(x_A - x_D) + (y_A - y_D) &\geq -1 \\ x_A - x_D &\leq 2 \\ y_A - y_D &\leq 2 \\ (x_A - x_D) - (y_A - y_D) &\leq 1 \\ x_A - x_D &\geq 0 \end{aligned}$$

Minimizing orthogonally to the bounding lines of the four non-terminal constraints yields no further vertices and we can terminate with the projection in Fig. A.1.e.

Simultaneously Shrinking an Outer Approximation

While growing the inner approximation, we can simulteneously maintain a shrinking outer approximation of the projection. The bounding faces of the outer approximation are obtained by establishing hyperplanes through the vertices of the inner approximation. The normal vector of such a hyperplane is the direction in which the corresponding supporting vertex was obtained. This is illustrated in Fig. A.2 for the above example. The dashed lines show the inner approximation. Since the faces of the inner approximation define the directions of the next optimizations, each bounding face of the outer approximation is parallel to a face of the previous inner approximation.

A.2 Base Cell Projection

This algorithm is a simplification of the above *Convex Hull Method* for the special case that the input cell is given by $\mathbf{Au} \geq 0$ ($\mathbf{u} = (x_1, y_1, \ldots, x_k, y_k)$). We compute the projection to the space of relative positions of part i with respect to part j.

In the general case, this projection will be a single convex cone in the plane with the apex at the origin. We therefore need to find a sector of feasible directions. Two special cases are possible: the projection can be (1) the whole plane (all directions feasible) or (2) the origin only (empty set of feasible directions).

To find the feasible directions, we test which points on a closed curve around the origin in the target c-space satisfy the cell constraints. For the curve we chose a square, because its edges can be most easily described by linear constraints in the variables $x_i - x_j$ and $y_i - y_j$. To test all points on the square, we run linear programs in both directions on each edge of the square subject to the additional constraints $\mathbf{Au} \geq 0$ of the given cell. Two optimizations per edge are sufficient since all points between the extrema must satisfy the constraints, too. The result is a connected sequence of feasible segments on the square which represents the feasible directions.

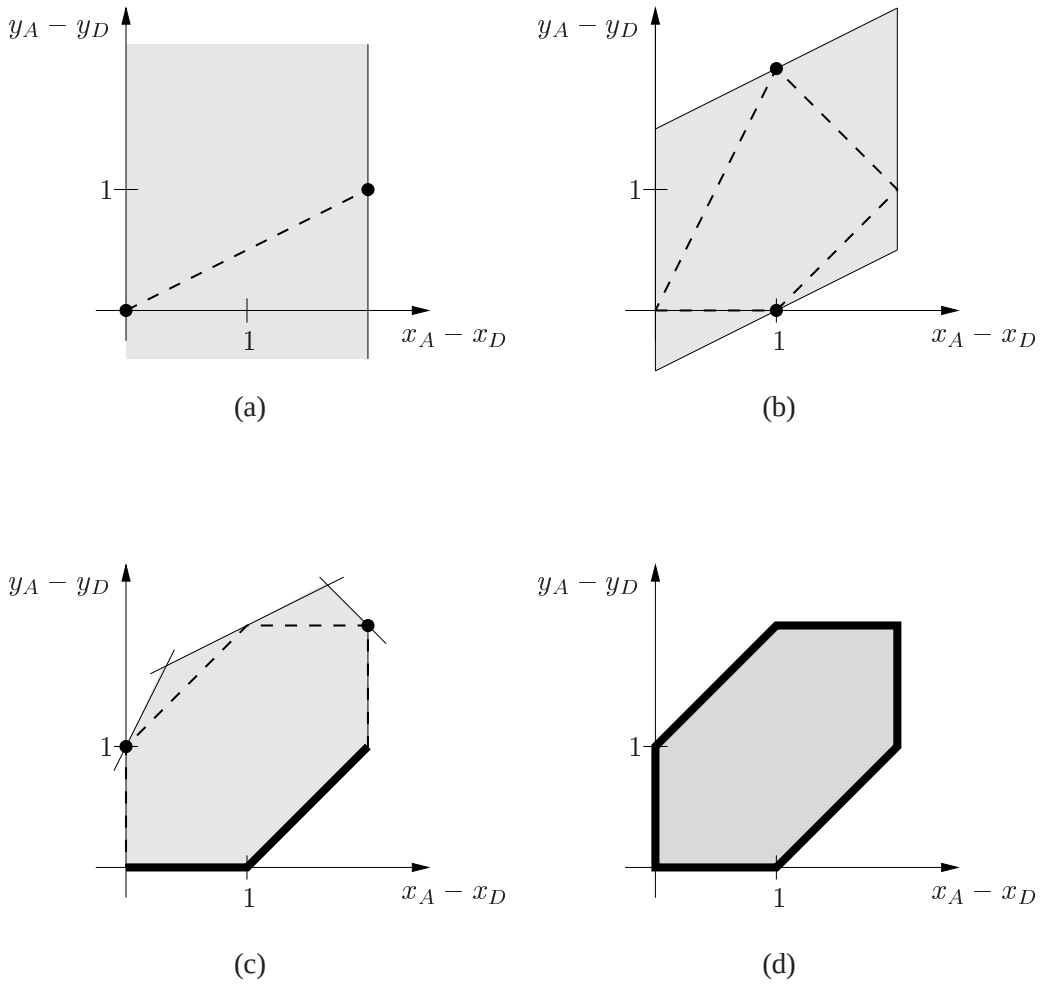


Figure A.2: Simultaneously shrinking an outer approximation (shaded) in the example. Inner approximation shown with dashed lines

The example in Fig. A.3 illustrates the concepts. Tab. A.1 summarizes the edge constraints which we add to the cell constraints $\mathbf{Au} \geq 0$, the different objective functions and the resulting extremal points of the maximizations. Here, e.g. maximization along the right vertical edge in positive y-direction ($\max y_i - y_j$) will yield the point q. In contrast, both optimizations using the bottom edge constraints will fail since the bottom edge is outside the projection. From the extremal points we can deduce the intersection of the square with the projection of the given cell. This intersection is shown with bold lines in the figure. Finally, the projection is constructed by casting two rays from the origin through the endpoints of the intersection (dashed lines in the figure).

If all optimizations succeed and each extremal points coincides with some square vertex, then the projection covers the whole plane. In contrast, if all optimizations fail, then the projection must be the origin only.

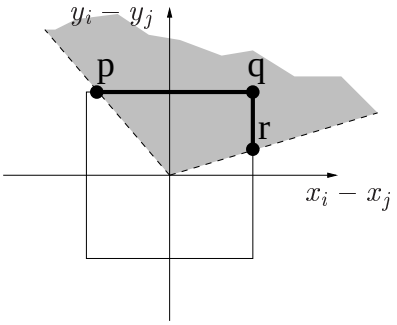


Figure A.3: Projection of a homogenous convex cell of directions to an arbitrary plane (shaded cone bounded by dashed lines). Extremal points p, q and r on centered square.

edge	additional edge constraints	objective functions	directions	results
right	$x_i - x_j = 1$, $-1 \leq y_i - y_j \leq 1$	$\pm(y_i - y_j)$	up(+),down(−)	$q(+), r(-)$
left	$x_i - x_j = -1$, $-1 \leq y_i - y_j \leq 1$	$\pm(y_i - y_j)$	up(+),down(−)	infeasible
top	$y_i - y_j = 1$, $-1 \leq x_i - x_j \leq 1$	$\pm(x_i - x_j)$	right(+),left(−)	$q(+), p(-)$
bottom	$y_i - y_j = -1$, $-1 \leq x_i - x_j \leq 1$	$\pm(x_i - x_j)$	right(+),left(−)	infeasible

Table A.1: Constraints specifying the edges of a centered square, objective functions for optimizing along edges and extremal points (p, q, r) for the situation in Fig. A.3.

Appendix B

Bounds on Vertex Coordinates

Given an $(n \times d)$-matrix $\mathbf{A}$ and an n-dimensional column-vector $\mathbf{b}$, both integral, we consider the polyhedron defined by

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}$$

We show that upper bounds for the 1-norm (l_1-norm) of all vertices of this set can be computed

1. with $O(nd)$ basic arithmetic operations, and
2. in time $O(L \log d + \log^2 d)$ where L denotes the number of bits required to represent $\mathbf{A}$ and $\mathbf{b}$.

Note that both time bounds do certainly not exceed the asymptotic time complexity of any conceivable unboundedness testing algorithm, as required in the proof of Lemma 13 (p. 78).

To derive the first bound, we can assume that there is a non-singular subsystem of $\mathbf{A}\mathbf{x} \geq \mathbf{b}$, which we denote $\mathbf{E}\mathbf{x} \geq \mathbf{f}$, such that a vertex $\mathbf{v}$ is the unique solution of $\mathbf{E}\mathbf{x} = \mathbf{f}$. Without loss of generality, $\mathbf{E}$ is a $(d \times d)$-matrix and $\mathbf{f}$ a d-vector. By Cramer's rule, $v_j = \det \mathbf{E}_j / \det \mathbf{E}$, where $\mathbf{E}_j$ is obtained from $\mathbf{E}$ by replacing the j-th column by $\mathbf{f}$. Since $|\det \mathbf{E}| \geq 1$ (the matrix is integral and non-singular), $|v_j| \leq |\det \mathbf{E}_j|$. The *Hadamard inequality* [17] states that for a $(d \times d)$-matrix $\mathbf{C}$,

$$|\det \mathbf{C}| \leq \prod_{i=1}^{d} \|\mathbf{c}_i\|$$

For the integral matrix $\mathbf{E}_j$, this inequality implies

$$|\det \mathbf{E}_j| \;\leq\; \prod_{i=1}^{d} \sqrt{f_i^2 - e_{ij}^2 + \sum_{k=1}^{d} e_{ik}^2} \;\leq\; \prod_{i=1}^{n} (b_i^2 + \sum_{k=1}^{d} a_{ik}^2)$$

Thus, an upper bound $c = d \cdot \max_j |\det \mathbf{E}_j|$ for the 1-norm of $\mathbf{v}$ can be computed with $O(nd)$ basic arithmetic operations.

Second, we analyze the required time to compute an upper bound c using the Turing machine model, that is, we additionally consider the size of the numbers occurring in our computations. From Lemma 3.1.33 in [17] we obtain $|v_j| \leq 2^{L-d^2}$ for any vertex $\mathbf{v}$, where L denotes the accumulated length of the (integral) input coefficients (using binary encoding). We may for instance choose $c = d2^L$. Since $L \geq d$, we can compute $d2^L$ in time $O(L \log d + \log^2 d)$.

Bibliography

[1] P.K. Agarwal, M. de Berg, D. Halperin, and M. Sharir. Efficient generation of k-directional assembly sequences. In *Proc. of the 7th ACM-SIAM Symp. Discr. Algorithms*, pages 122–131. SODA, 1996.

[2] J.-D. Boissonnat, E. de Lange, and M. Teillaud. Minkowski operations for satellite antenna layout. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*. Association for Computing Machinery, 1997.

[3] J.F. Canny. *Complexity of Robot Motion Planning*. PhD thesis, Massachusetts Institute of Technology, 1988.

[4] B. Chazelle. Approximation and decomposition of shapes. In J.T. Schwartz and C.K. Yap, editors, *Advances in Robotics*, volume 1, pages 145–185, Hillsdale, NJ, 1987. Erlbaum.

[5] B. Chazelle and D.P. Dobkin. Optimal convex decompositions. In G.T. Toussaint, editor, *Computational Geometry*, pages 63–133. North–Holland, Amsterdam, 1985.

[6] B. Chazelle, T.A. Ottmann, E. Soisalon-Soininen, and D. Wood. The complexity and decidability of SEPARATION. In *Proc. of the 11th Int. Coll. Autom. Lang. Prog.*, pages 119–127. LNCS, 1984.

[7] B. Chazelle and L. Palios. Triangulating a nonconvex polytope. *Driscrete Comput. Geom.*, 5:505–526, 1990.

[8] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In J. Mylopoulos and R. Reiter, editors, *Proc. of IJCAI-91*, pages 331–337. Morgan Kaufmann, 1991.

[9] J.C. Culberson and R.A. Reckhow. Covering polygons is hard. *Journal of Algorithms*, 17(1):2–44, July 1994.

[10] K. Daniels. *Containment Algorithms for Nonconvex Polygons with Applications to Layout*. PhD thesis, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, May 1995.

[11] R.J. Dawson. On removing a ball without disturbing the others. *Mathematics Magazine*, 57(1):27–30, January 1984.

[12] L.S. Homem de Mello and S. Lee, editors. *Computer-Aided Mechanical Assembly Planning*. Kluwer Academic Publishers, 1991.

[13] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer, Heidelberg, 1987.

[14] B.P. Flannery, S.A. Teukolsky, W.H. Press, and W.T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1993.

[15] V. Gaede and O. Gñther. Multidimensional access methods. *ACM Computing Surveys*, 20(2):170–232, 1998.

[16] M.R. Garey and D.S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H Freeman and Co., San Francisco, 1979.

[17] M. Grötschel, László Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1993.

[18] L. Guibas, D. Halperin, H. Hirukawa, J.-C. Latombe, and R.H. Wilson. A simple and efficient procedure for polyhedral assembly partitioning under infinitesimal motions. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 2553–2560. IEEE, 1995.

[19] L.J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 100–111. IEEE, 1983.

[20] L.J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.

[21] K. Gupta and A.P. del Pobil, editors. *Practical Motion Planning in Robotics*. John Wiley & Sons, 1998.

[22] D. Halperin, J.-C. Latombe, and R.H. Wilson. A general framework for assembly planning: The motion space approach. In *Proc. of the 14th Annual ACM Symposium on Computational Geometry*, Minneapolis, Minnesota, 1998.

[23] D. Halperin and R.H. Wilson. Assembly partitioning along simple paths: the case of multiple translations. *Advanced Robotics*, 11:127–146, 1997.

[24] D.S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, 1994.

[25] J.E. Hopcroft, Schwartz J.T., and M. Sharir. On the complexity of motion planning for multiple independent objects. *Intl. J. of Robotics Research*, 3(4):76–88, 1984.

[26] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 1979.

[27] J.E. Hopcroft and G.T. Wilfong. Motion of objects in contact. *International Journal of Robotics Research*, 4(4):32–46, 1986.

[28] T. Huynh, L. Joskowicz, C. Lassez, and J.L. Lassez. Practical tools for reasoning about linear constraints. *Fundamenta Informaticae*, 15(3–4), 1991.

[29] L. Joskowicz and E. Sacks. Incremental configuration space construction for mechanism analysis. In *Proc. of the 9th National Conference on Artificial Intelligence*, pages 888–893. Morgan Kauffman Publishers, 1991.

[30] L. Joskowicz and R.H. Taylor. Interference-free insertion of a solid body into a cavity: An algorithm and a medical application. *International Journal of Robotics Research*, 15(3):211–229, 1996. http://www.cs.huji.ac.il/ josko/mplan.html.

[31] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC*, pages 302–311, 1984.

[32] A. Kaul and M.A. O'Connor. Computing minkowski sums of regular polyhedra. Report RC 18891 (82557) 5/12/93, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, 1992.

[33] A. Kaul, M.A. O'Connor, and V. Srinivasan. Computing minkowski sums of regular polygons. In *Proc. of the 3rd Canad. Conf. Comput. Geom.*, pages 74–77, 1991.

[34] L.E. Kavraki and M.N. Kolountzakis. Partitioning a planar assembly into two connected parts is NP-complete. *Information Processing Letters*, 55(3):159–165, 1995.

[35] K. Kedem and M. Sharir. An efficient motion-planning algorithm for a convex polygonal object in two-dimensional polygonal space. *Discrete and Computational Geometry*, 5:43–75, 1990.

[36] D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, 1973.

[37] H. Hirukawa et al. L. Guibas, D. Halperin. Polyhedral assembly partitioning using maximally covered cells in arrangements of convex polytopes. To appear: Intl. J. Comp. Geometry and Applications.

[38] E. Larsen, S. Gottschalk, M.C. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, UNC Chapel Hill, 1999.

[39] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[40] Z. Li. *Compaction Algorithms for Non-Convex Polygons and Their Applications*. PhD thesis, Division of Applied Sciences, Harvard University, Cambridge, Massachusetts, May 1994.

[41] A. Lingas. The power of non-rectilinear holes. In *Proceedings of the Ninth International Colloquium on Automata, Languages and programming, Aarhus, Denmark*, pages 369–383, New York, July 1982. Lecture Notes in Computer Science 140, Springer-Verlag.

[42] T. Lozano-Pérez. Spatial planning: A configuration space approach. In *Transactions on Computers*, pages C–32(2):108–120. IEEE, 1983.

[43] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*. Springer-Verlag, Berlin, 1984.

[44] K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. Springer-Verlag, Berlin, 1984.

[45] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Max-Planck-Institut für Informatik, Saarbrücken, 1995. http://www.mpi-sb.mpg.de/LEDA/leda.html.

[46] B.A. Murtagh and M.A. Saunders. Minos 5.4 user's guide. Technical Report SOL 83-20R, Dept. of Operations Research, Stanford University, 1983.

[47] B.K. Natarajan. On planning assemblies. In *Proc. of the ACM Symp. on Computational Geometry*, pages 299–308, 1988.

[48] J. Ba non. Implementation and extension of the ladder algorithm. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1548–1553, Cincinnati, OH, 1990.

[49] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, Cambridge, 1994.

[50] R.S. Palmer. *Computational Complexity of Motion and Stability of Polygons*. PhD thesis, Cornell University, 1987.

[51] D.M. Pennock and Q.F. Stout. Exploiting a theory of phase transitions in three-satisfiability problems. In *Proc. American Association of Artificial Intelligence*, pages 253–258, 1996.

[52] R. Pollack, M. Sharir, and S. Sifrony. Separating two simple polygons by a sequence of translations. *Discrete and Computational Geometry*, 3:123–136, 1988.

[53] V.R. Pratt. Two easy theories whose combination is hard. Technical report, Massachusetts Institute of Technology, September 1977.

[54] G. Ramalingam, J. Song, L. Joskowicz, and R.E. Miller. Solving systems of difference constraints incrementally. *Algorithmica*, 23(3):261–275, 1998.

[55] G.D. Ramkumar. *Tracings And Their Convolution: Theory And Applications*. PhD thesis, Department of Computer Science, Stanford University, 1998.

[56] J.H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[57] E. Sacks and C. Bajaj. Sliced configuration spaces for curved planar bodies. *International Journal of Robotics Research*, 17(6):639–651, 1998.

[58] E. Sacks and L. Joskowicz. Automated modeling and kinematic simulation of mechanisms. *Computer-Aided Design*, 25(2):107–118, 1993.

[59] J.T. Schwartz and M. Sharir. On the piano movers' problem: I. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[60] J.T. Schwartz and M. Sharir. On the piano movers' problem: II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, pages 298–351, 1983.

[61] F. Schwarzer, F. Bieberbach, A. Schweikard, and L. Joskowicz. Efficiently testing for unboundedness and m-handed assembly. In *Proc. of the IEEE Intl. Conf. on Intelligent Robots and Systems*, pages 1164–1169. IEEE/RSJ, 1998.

[62] F. Schwarzer, A. Schweikard, and L. Joskowicz. Efficient unboundedness testing: Algorithm and applications to assembly planning. *International Journal of Robotics Research*, 19(9):817–834, September 2000.

[63] A. Schweikard and F. Schwarzer. General translational assembly planning. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 162–169. IEEE, 1997.

[64] A. Schweikard and F. Schwarzer. Detecting geometric infeasibility. *Artificial Intelligence*, 105:139–159, 1998.

[65] A. Schweikard and R.H. Wilson. Assembly sequences for polyhedra. *Algorithmica*, 13(6):539–552, June 1995.

[66] J. Snoeyink and J. Stolfi. Objects that cannot be taken apart with two hands. In *Proc. of the 9th ACM Symp. on Computational Geometry*, 1993.

[67] J. Stoer. *Einführung in die Numerische Mathematik I.* Springer-Verlag, Berlin, Heidelberg, New York, 2nd edition, 1976.

[68] S. Thienel. *ABACUS – A Branch-And-CUt System*. PhD thesis, Universität Köln, 1995.

[69] G.T. Toussaint. Movable separability of sets. In G.T. Toussaint, editor, *Computational Geometry*, pages 335–375. Elsevier. North-Holland, Amsterdam, 1985.

[70] G.T. Toussaint. On separating two simple polygons by a single translation. *Discrete and Computational Geometry*, 4:265–278, 1989.

[71] R. Wilson. *On Geometric Assembly Planning*. PhD thesis, Department of Computer Science, Stanford University, 1992.

[72] R.H. Wilson, L. Kavraki, J.-C. Latombe, and T. Lozano-Pérez. Two-handed assembly sequencing. *Intl. Journal of Robotics Research*, 14(4):335–350, 1995.

[73] J.D. Wolter. *On the Automatic Generation of Plans for Mechanical Assembly*. PhD thesis, University of Michigan, Sept 1988.

[74] M.H. Wright. *Interior Methods for Constrained Optimization*. Acta Numerica. Cambridge University Press, New York, 1992.