COMPLEXITY MEASURES FOR ASSEMBLY SEQUENCES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Michael Goldwasser

June 1997

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Rajeev Motwani
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Leonidas Guibas

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Jean-Claude Latombe

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

Our work focuses on various complexity measures for two-handed assembly sequences. For many products, there exist an exponentially large set of valid sequences, and a natural goal is to use automated systems to select wisely from the choices. Since assembly sequencing is a preprocessing phase for a long and expensive manufacturing process, any work towards finding a better assembly sequence is of great value when it comes time to assemble the physical product in mass quantities. Although there has been a great deal of algorithmic success for finding feasible assembly sequences, there has been very little success towards optimizing the costs of sequences. We attempt to explain this lack of progress, by proving the inherent difficulty in finding optimal, or even near-optimal, assembly sequences.

We begin by introducing a formal framework for studying the optimization of complexity measures for assembly sequencing. Based on the previous work of both researchers and practitioners, we collect a list of various cost measures, goals, and restrictions that others have considered desirable. Together with this, we define a graph-theoretic problem that is a generalization of assembly sequencing, focusing on the combinatorial aspect of the family of feasible assembly sequences, while temporarily separating out the specific geometric assumptions inherent to assembly sequencing. In this *virtual assembly sequencing* model, we are still able to explain the success of previous algorithms in finding feasible sequences efficiently.

At this point, we begin to consider the approximability of the various cost measures for sequencing, and we examine the lack of success in optimizing the costs. We examine several intuitive, yet unsuccessful, heuristics for optimizing the cost of sequences, giving constructions which result in worst case performance, while also testing these heuristics experimentally on several products, previously used as a test bed for research in assembly sequencing. Because of this lack of success in designing approximation algorithms, we continue by examining the source of difficulty in these problems. We show how these problems capture the combined difficulty of several covering, scheduling, and sequencing problems from the literature. We use techniques common to the theory of approximability to prove the hardness of finding even near–optimal sequences for most cost measures in our generalized framework. Our strongest lower bounds prove that finding any solution within a $2^{\log^{1-\gamma} n}$-factor of the optimal cost solution is quasi-NP-hard for any $\gamma > 0$. As a special case, we prove similar, strong inapproximability results for the problem of scheduling with AND/OR precedence constraints.

Of course, hardness results in our generalized framework do not immediately carry over to the original geometric problems. It may be that hard instances of our graph-theoretic problem do not arise from the geometric settings originally part of the assembly sequencing problems. Therefore, we re-introduce the geometry, and continue by realizing several of these hardness results in rather simple geometric settings. We are able to show strong inapproximability results in far simpler settings than the domain of most assembly sequencers, for example using an assembly consisting solely of unit disks in the plane.

In the face of our results, the overwhelming open problem is to design any non-trivial approximation algorithms for minimizing the cost of assembly sequencing. In our general setting, we give strong lower bounds, however there is still a gap between the trivial upper bound, and so it would be interesting to develop an algorithm that is able to match the lower bounds. Furthermore, it would be quite valuable to identify any practical geometric settings not considered by us, that would allow for approximation-algorithms which are able to break our generalized lower bounds.

Acknowledgments

I dedicate this dissertation to my parents, Marilyn and Bob, as they deserve the credit for the person whom I have become. Their parenting has instilled in me a sense of accomplishment, a sense of caring, a sense of responsibility, a sense of morality, and a sense of curiosity. When I become a parent myself, I only hope I can continue in their style. Additionally, I would like to thank my siblings for encouraging my intellectual curiosity, beginning with early math lessons on the blackboard in our basement, and continuing with endless hours spent playing game after game while growing up.

To my fiancé, Susan, I wish to express my heartfelt gratitude for her constant support in my life. In many ways, the completion of this thesis symbolizes an end to a period in my life; however a new chapter is about to begin with our marriage next month. I look forward to a life with Susan as my partner.

In my academic life, I wish to acknowledge not only those whose help has been so beneficial to by thesis, but also those who have been a part of my life over the years. With four years at Brown, followed by six years at Stanford, the list has grown quite rapidly.

To begin with, I would like to thank all the members of my reading committee. Always with an open door, Rajeev Motwani has served as both my advisor and also as a role model. Rajeev's enthusiasm for research seemed unlimited, and his commitment to quality teaching was remarkable. I often found myself sitting in his classes, admiring his organization, preparation and explanations; these lessons I take with me as I become a teacher myself. Leo Guibas, who served as my first advisor, has been enlightening, with both his vision of the field as well as his ability to explain his ideas to others. I would also like to thank Leo for the many times I found myself enjoying dinner, being entertained by personal stories about the history of computer science, the people involved, and about life in general. Finally, Jean-Claude Latombe has been instrumental in the creation of this thesis. His work in assembly planning is what brought me to studying this problem.

In the same regard, I would like to acknowledge all of the members of the Stanford assembly planning group, namely Leo Guibas, Jean-Claude Latombe, Danny Halperin, Cyprien Godard, G. Ramkumar, and Bruce Romney. In particular, I am especially grateful for my frequent meetings with Danny Halperin while at Stanford, as his interactions were invaluable. Additionally, there were several other people in the assembly planning community whom I have met over the years. Randy Wilson's thesis has been the springboard for a good deal of this work. Although he graduated from Stanford shortly after my arrival, continued conversations with Randy have been a key in shaping my thesis. Additionally, I would like to thank both Randy, Jack Snoeyink, and Russell Brown, for sharing various models of assemblies for use in the experiments of Section 4.5. Finally, I would like to thank both Ken Goldberg and Jan Wolter for several interesting conversations in the halls of robotics conferences.

A large part of my experience as a student has been in learning to be a teacher. A significant part of this lesson has been my exposure to the styles and techniques of so many different professors for whom I have served as a teaching assistant. These relationships have provided me with many ideas and thoughts in shaping my own style and image as a teacher. For this reason I would like to thank, Andrei Broder, Todd Feldman, Leo Guibas, Philip Klein, Leora Morgenstern, Vaughn Pratt, Lyle Ramshaw, Roberto Tamassia, and Jeff Vitter.

The computer science students at Stanford have been such a constant part of my life for so long, it is difficult to thank each of them properly. but I would like to thank the following for the role they have played in my daily life at work: Donald Aingworth, Julien Basch, Craig Becker, Jerry Cain, Moses Charikar, Chandra Chekuri, Scott Cohen, Steve Cousins, Harish Devarajan, Kathleen Fisher, Cyprien Godard, Ashish Goel, Sudipto Guha, Piotr Indyk, David Karger, Robert Kennedy, Sanjeev Khanna, Daphne Koller, Andrew Kosoresow, Jeffrey Oldham, Steven Phillips, G. Ramkumar, Bruce Romney, Craig Silverstein, Eric Torng, Eric Veach, and Suresh Venkatasubramanian.

I am also quite grateful to have kept in such close contact, even to this day, with friends from Brown, most notably, Ken Avenoso, David Luks, and David Pawson. It is a testament to the value of email, that I feel as if these people are still just down the hall from me. Finally, I would like to thank those who helped me so much in enjoying the time, while delaying my thesis. My bridge playing has improved, much in thanks to Greg Crawford, Jesse David, Rajendra Gangadean, Brian Murphy, and Jason Scott.

Financial support for this work has been given by a grant from the Stanford Integrated Manufacturing Association (SIMA), by NSF/ARPA Grant IRI-9306544, by NSF Grant CCR-9215219, by ARO MURI Grant DAAH04-96-1-0007 and by NSF Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

Contents

List of Figures

Chapter 1

Introduction

Given a set of parts and a geometric description of their relative positions in a product, the assembly sequencing problem is to devise a sequence of collision-free operations that results in the assembly of the product from the individual parts. Efficient algorithms have been developed, for many classes of motions, which guarantee to find a valid assembly sequence when one exists. The IEEE Technical Committee on Assembly and Task Planning summarized the current state of assembly sequencing by explaining [28], "after years of work in this field, a basic planning methodology has emerged that is capable of producing a feasible plan ... The challenges still facing the field are to develop efficient and robust analysis tools and to develop planners capable of finding optimal or near–optimal sequences rather than just feasible sequences." Indeed, better understanding the inherent complexity of assembling a product is critically important for bringing assembly planning systems into industrial use. When the resulting assembly sequence is used in manufacturing, it will be performed in mass quantities, and so the cost of the sequence is of great importance. In cases where we find that the optimal cost assembly sequence is quite expensive, this information can be used by engineers in redesigning the product in a design-for-assembly feedback loop. Additionally, the efficiency of such algorithms for finding these assembly sequences is of critical importance, as modern products are being designed with hundreds or thousands of parts, or more.

Unfortunately, there has been little algorithmic success in optimizing the cost of assembly sequences. Our work explains this lack of progress by formally proving the hardness of approximating the optimal cost sequence in a variety of settings. We attempt to classify the approximability of many variants of assembly sequencing, based on the desired cost measure, the specific goal required, and other restrictions placed on the sequence. Besides considering the standard goal of fully assembling a product from its individual parts, we consider several motivated partial disassembly tasks such as removing a key part from an assembly. Our list of possible cost measures includes many measures suggested by both industry and previous researchers. Examples include minimizing the number of distinct directions of motion used during a sequence, minimizing the number of steps in a sequence, or minimizing the number of re-orientations of the assembly.

We begin by studying a graph-theoretic generalization of assembly sequencing which we term *virtual assembly sequencing* (VAS). Much of the success in finding feasible sequences has been a result of the introduction of the *non-directional blocking graph* [74, 76]. For a given direction of motion, the geometric model of the product can be analyzed to construct a graph that represents the blocking relationships among the parts. Once a set of such graphs has been computed, it can be analyzed to compute a feasible (dis)assembly sequence, when one exists. This setting is responsible for much of the recent success in finding valid assembly sequences for a variety of settings. Our generalized model considers this set of blocking graphs as the original input to the problem, and we examine whether these graphs can be used to find near-optimal sequences, rather than simply feasible sequences.

In this model, we can make some immediate observations that explain the past success in finding feasible assembly sequences, as well as finding optimal sequences for a few limited situations. Unfortunately, for most desired cost measures, such immediate ideas are not successful. We consider several intuitive heuristics, and present counterexamples to show that in the worst case, these heuristics produce an assembly sequence with a cost that is not significantly better than the cost of the worst possible sequence. We feel that these counterexamples are not at all pathological, and that the failure of these heuristics is quite common. To justify this view, we provide experiments showing the results of these heuristics on a collection of products used previously, as a test bed of examples for previous research in assembly sequencing.

Facing the lack of success in optimizing assembly sequences, we consider the source of the difficulty. We find that the problem of optimizing assembly sequences in our model captures the difficulty of several known covering, scheduling, and supersequencing problems. This allows us to formally prove the hardness, not only for finding the optimal cost solution, but even for finding any near-optimal solutions. For many of the variants, our strongest results show that it is hard to find any sequence whose cost can be bounded to within a $2^{\log^{1-\gamma} n}$-factor of the optimal cost sequence for any $\gamma > 0$. As a special case, when sequences are restricted to move only one part at a time, this problem can be modeled as an instance of scheduling with AND/OR precedence constraints[1]. We prove similar inapproximability results for this scheduling problem. A more complete summary of our exact results is given later, in Tables 6.1–6.3.

Finally, since our virtual assembly sequencing model is a generalization, our lower bounds do not necessarily apply to the original problem as we no longer assume that the set of input graphs is the result of any original geometric setting. We continue by showing that many of our lower bounds can, in fact, be realized geometrically, thereby proving the hardness of the true assembly sequencing problems. As an example, we consider a setting consisting entirely of unit disks in the plane, and we look at the task of removing a given disk from the rest of the assembly using only individual translations to infinity. We prove that achieving a $2^{\log^{1-\gamma} n}$-approximation to minimizing the total number of disks which must be removed to access the given disks is hard for any $\gamma > 0$.

Much of this work has appeared previously in [26, 27, 67].

1.1 Presentation Overview

This paper proceeds as follows. In the following section, we define the assembly sequencing problem and introduce several terms and definitions used throughout the paper. Chapter 2 discusses previous work in the areas of assembly sequencing, approximation theory, and computational geometry, which relate to our work. We introduce the *virtual assembly sequencing* problem in Chapter 3, as we formalize a list of possible goals, restrictions, and cost measures for assembly sequencing.

Following this, we approach the problems of how well the choice of sequences can be optimized over such cost measures. Chapter 4 discusses those problems which can be solved successfully from examining the set of blocking graphs, as well as other problems which do not seem to afford the same success. In this chapter we consider several possible heuristics, showing poor performance in the worst case, and we report on experiments using these heuristics on a test bed of products. Chapter 5 discusses the problem of scheduling with AND/OR precedence constraints, a special case of virtual assembly sequencing, as we prove the inapproximability for this problem. We use this result in Chapter 6, along with many

[1]N.B.: this is not to be confused with the AND/OR tree used by Homem de Mello and Sanderson for representing all feasible assembly sequences[36, 38]

reductions between different variants of the VAS problem, to prove the inapproximability of most variants of the virtual assembly sequencing problems. Finally, we re-introduce the geometry in Chapter 7, proving the inapproximability for several cost measures, even in the original geometric settings. Conclusions and future directions of research are given in Chapter 8.

1.2 Definition of Assembly Sequencing

In general terms, the input to an assembly sequencer is a **product**, consisting of a set of **parts** and described by a geometric model of the parts and their relative positions, as well as a family of allowable **motions**. For example, an assembly may consist of a collection of unit disks in the plane, and the family of allowable motions may be translations to infinity. The classic goal is to produce a sequence of operations resulting in the construction of the product from its individual parts. Each operation combines a set of subassemblies, using motions from the allowable family.

In the assembly sequencing problem, we will concern ourselves only with finding a feasible sequence of collision-free motions. We are not concerned with grasping the objects, the forces involved, or the stability of the subassemblies; rather we will think of our parts as free-floating objects. Additionally, we assume that the product is made of **rigid** parts, we assume that each operation is **binary**, that is, combines exactly two subassemblies, and we consider only **monotone** assembly sequences, that is, when an operation has placed a part in a subassembly, that part may no longer be moved relative to the subassembly. Although restrictive, these assumptions are common in assembly sequencing and can be applied to a majority of products.

We may think of devising an assembly sequence by constructing a **disassembly** sequence and then reversing the entire sequence. In general, these tasks are not necessarily symmetric, for instance when considering flexible parts which may be deformed during assembly (e.g., snap-fit parts), or when considering stability, insertion forces or fixturing [55]. However, under our assumptions, these two tasks are indeed symmetrical. The advantage of the assembly-by-disassembly approach is that the final assembled product is usually much more constrained than the initial configuration of parts, and so infeasible plans can be more quickly eliminated in this way. Additionally, there are several jobs related to maintenance or recycling of products that require a partial disassembly of a complete product.

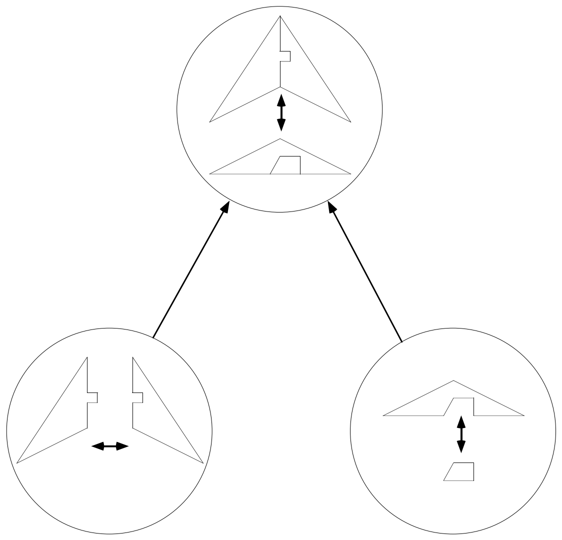


Figure 1.1: Assembly tree for a simple product

With this in mind, the goal of a binary, monotone assembly sequencer is to start with the fully assembled product, and partition the set of parts into two groups that can be **separated** by a collision-free motion. Once this is done, each of the resulting subassemblies can be disassembled in a similar manner. The structure of this decomposition can be represented naturally as a binary **assembly tree**. Figure 1.1 gives an example of such an assembly tree, taken from [76], for a simple two-dimensional product. The root of the tree represents the fully assembled product, and the children of an internal node represent two subassemblies that can be combined together to produce the larger subassembly represented by the parent. Note however, that the assembly tree only represents the structure of the decomposition, but not the desired sequence in which the operations are performed.

Chapter 2

Background

2.1 Assembly Sequencing

The use of automation in assembly sequencing has increased rapidly over the years [7, 21, 35, 37, 38, 52, 56, 74, 77, 78]. Progressing from days when assembly sequencing was purely a craft of the human designers, computers have become a powerful tool in the sequencing process. Early systems resulted in potentially exponential time generate-and-test sequencers, operating by generating candidate operations and testing their feasibility [38, 77]. The problem of finding a valid assembly sequence was later shown to be intractable in many general settings [39, 47, 48, 63, 75, 78]. This led some researchers to consider restricted, but still interesting, versions of the problem, for instance requiring *monotone* sequences, where each operation generates a final subassembly, *two-handed* sequences, where every operation merges exactly two subassemblies. For many classes of motions parameterized by a constant number of degrees of freedom, polynomial algorithms were then developed to find a binary, monotone assembly sequence when one exists [30, 33, 74, 76]. A good deal of this success was achieved within the framework of *non-directional blocking graphs* [32, 74, 76]. As our work is intricately related to this approach, in the following section, we will review in greater detail the concept of non-directional blocking graphs and the subsequent results. Other techniques allow for the enumeration of all possible assembly sequences in time proportional to the number of such sequences [21], however for most products, there will be exponentially many such sequences.

The *assembly-by-disassembly* approach for assembly sequencing has become quite popular. When considering non-rigid parts, stability, fixturing, and insertion forces, assembly and disassembly sequences are no longer symmetric [55]. However, even under our assumptions, when there is a symmetry, there are several important problems, such as removing a given part for service, which arise as partial disassembly problems. For this reason, some researchers have focused on models for disassembly [54, 57, 68, 69, 72].

With the ability to find feasible sequences efficiently, researchers have noted the importance of evaluating the inherent complexity of a product in terms of the optimal cost for assembly. In fact, in 1994, the IEEE Technical Committee on Assembly and Task Planning summarized the current state of assembly sequencing by explaining [28], "after years of work in this field, a basic planning methodology has emerged that is capable of producing a feasible plan . . . The challenges still facing the field are to develop efficient and robust analysis tools and to develop planners capable of finding optimal or near–optimal sequences rather than just feasible sequences." This focus on evaluating the cost of assembly sequences has grown [55, 72, 76, 78, 79].

The first issue for finding low cost sequences, of course, is that there are many possible ways to define the cost of a sequence, depending on how the sequence will be used in a manufacturing system. Based on a great deal of work with industrial applications, Boothroyd *et al.* suggest several empirical measures that effect the cost of assembly for a product [10, 11]. More formal complexity measures have been defined by both Wilson and Latombe [76], as well as Wolter [78]. A collection of cost measures for assembly planning gathered by Jones and Wilson is included in [42].

Once a cost measure has been chosen, the question becomes how to find a low cost assembly sequence. As we mentioned, it is possible to generate all possible sequences, and thus evaluate the cost of each, choosing the best. However, as there may be exponentially many such valid sequences, this approach becomes impractical for increasingly complex assemblies. Several people have looked at practical ways for grouping parts of an assembly in a way, so as to reduce the effective number of parts in an assembly, thus allowing a quicker search for the optimal sequence. A hierarchical approach is used by Chakrabarty and Wolter to identify common subassemblies in products, such as the many passenger chairs in an airplane [13]. Moradi *et al.* consider the automatic identification of groups of parts that either can be or must be assembled together, again reducing the effective number of sequences to consider [61]. Although both of these techniques are quite practical for reducing the effective size of the problems, they simply delay the exponential computation required to overcome increasingly large data sets, and thus the need for better automated reasoning for finding low cost sequences.

Others have looked at general heuristics that attempt to minimize the cost of assembly sequences. Millner *et al.* consider using simulated annealing in selecting least-cost assembly sequences [60], and Caselli and Zanichelli consider the use of petri nets for finding assembly sequences [12]. Both of these techniques suffer either in requiring possibly exponential time in finding the optimal sequence or else in quickly finding a sequence without any provable guarantee as to its quality.

For a restricted class of inputs that have a so-called "total ordering" property, a greedy algorithm is given that claims to produce the minimal length sequence to remove any give part [20, 79], however the required input property does not have a clear definition. For the general setting, our results in Section 7.5 will prove not only the difficulty of finding an optimal sequence by this cost measure, but even a near-optimal solution. We directly question these previous claims in Appendix B.

Finally, several software systems offer the user the option of optimizing the sequence over a choice of complexity measures [46, 67, 78], however these systems must rely on current techniques and thus either require possibly exponential search techniques to find the true optimal, or else polynomial heuristics with no performance guarantees on the cost of the resulting sequence.

2.1.1 A Review of Non-Directional Blocking Graphs

A key concept in understanding current techniques in assembly sequencing is that of a *directional blocking graph* (DBG). For a specific motion, a DBG can be defined as a directed graph with a node for each part of the assembly, and an edge $A \rightarrow B$, if part A collides with part B when that motion is applied to A, while B remains stationary. Figure 2.1 gives an example of a two-dimensional product as well as two DBGs for infinitesimal translation [76]. The blocking graph for a given motion provides a compact representation of all collision-free operations for that motion, as each directed cut between some subset S and subset T in a DBG corresponds to a collision-free partition. Since a DBG represents all possible operations for a single direction of motion, by constructing a DBG for each possible motion, we can fully represent all possible operations. Unfortunately, there may be infinitely many different motions.

The key insight is that many distinct motions may be represented by the identical DBG, since slight changes in a direction may not effect the blocking relationships between any of the parts. This fact led to the development of the *non-directional blocking graph* (NDBG) [74, 76].

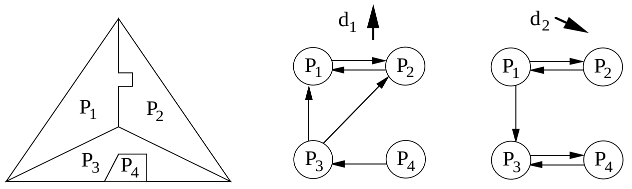


Figure 2.1: A simple assembly and two DBG's for infinitesimal translation

During the construction of an NDBG, the space of motions are divided into equivalence classes based on the blocking graphs, and the resulting NDBG consists of a single DBG for each equivalence class. Thus the NDBG completely captures the necessary geometric information for identifying all valid operations for those motions. The only issue remaining is the number of equivalence classes and how to compute them.

In the original work [74], they show that the number of such equivalence classes is polynomially bounded in the complexity of the input, for three-dimensional polyhedra, when the operations involved are either infinitesimal translations, infinitesimal translations with rotations, or translations to infinity. In a series of work since then, geometric algorithms have been developed and improved for building the NDBG when the motion class allowed includes, infinitesimal translations [76], extended translations (i.e., to infinity) [76], multiple step translations in the plane [33], and infinitesimal generalized motions (i.e., rigid body motions) [30, 76]. As a general rule, it seems that a family of motions with a constant number of degrees of freedom leads to a polynomial number of distinct equivalence classes. A more recent survey presents a unified framework for understanding the collection of work surrounding the non-directional blocking graphs [32].

For each of these families of motions, the NDBG framework immediately provides a polynomial time algorithm for constructing a feasible assembly sequence, if one exists. After constructing a polynomial set of DBG's, an arbitrary assembly sequence is found by taking any legal separation using any of the directions, and then recursing on the resulting subassemblies. Since the removal of parts can only reduce the blocking relationships, there will be no false dead ends and this procedure will result in either producing an entire assembly tree, or else will reach a subassembly which cannot be partitioned by any of the motions, thereby proving that no assembly sequence exists. This algorithm runs in polynomially time, is quite simple, and has been implemented in assembly sequencing systems for many of the above motion classes [30, 46, 67, 74]. As we will see, searching for a "good" sequence in this way is not quite so simple.

2.2 Approximation Theory

For most variants, finding the optimal cost assembly sequence will turn out to be NP-hard. As the number of parts and complexity of products keeps increasing, it quickly becomes infeasible to rely on an exponential time search to find the best possibilities. Unless P=NP, there is little hope of efficiently finding the true optimal solution, however this does not rule out the possibility of finding near-optimal solutions efficiently. Since assembly sequencing is a preprocessing phase for a long and expensive manufacturing process, any work towards finding as good of an assembly sequence as possible is of great value when it comes time to assemble the physical product in mass quantities. There is nothing particularly magical about the exact best solution from an industrial point of view; if an efficient algorithm could guarantee that it could find a sequence whose cost was, for example, within 1% of the optimal sequence, this would probably be quite adequate.

Research in the theory of approximability has consider exactly this issue for other NP-hard optimization problems [4, 23, 41, 62]. Since we cannot expect to find the optimal solution in polynomial time, the goal is to develop a polynomial time *approximation algorithm* that returns a solution whose cost can be bounded by some function of the true optimal cost. A standard measure for the quality of an approximation algorithm is the *approximation ratio*, comparing the cost of the solution returned by the algorithm versus the cost of the true optimal solution. Although all NP-complete decision problems can be reduced to one another, the approximability of such problems can be quite different, ranging from those which can be approximated arbitrarily closely to the optimal, to those for which getting even a very rough approximation is already NP-hard.

Many researchers have worked towards classifying the approximability of different NP-hard problems [4, 5, 15, 64]. We will consider four broad classes defined in [4], which group problems based on the strength of the inapproximability results that have been proven. Class I includes all problems for which approximating the optimal solution to within a factor of $(1+\epsilon)$ is NP-hard for some $\epsilon > 0$. The canonical problem for this class is MAX-3SAT, and the class includes all Max-SNP-complete problems [64], for example VERTEX COVER, METRIC TSP, MAX CUT, and others. Class II groups those problems for which it is quasi-NP-hard[1] to achieve an approximation ratio of $c \cdot \log n$ for some $c > 0$. The typical such problem in this class is SET COVER, for which the threshold of approximability has been placed at

[1]that is, this would imply NP $\subseteq$ DTIME$(n^{\text{poly}(\log n)})$. "A proof of quasi-NP-hardness is good evidence that the problem has no polynomial-time algorithm." [4]

Class	Factor of Approximation that is hard	Representative Problems
I	$1+\epsilon$	MAX-3SAT
II	$O(\log n)$	SET COVER
III	$2^{\log^{1-\gamma} n}$	LABEL COVER
IV	n^{ϵ}	CLIQUE

Table 2.1: The four classes and their representative problems [Arora/Lund]

$\ln n(1+o(1))$ [22]. For problems in Class III, it is quasi-NP-hard to achieve a $2^{\log^{1-\gamma} n}$ factor[2] approximation for any $\gamma > 0$. LABEL COVER is the canonical problem in this class [3], although the class contains several other natural problems such as LONGEST PATH [45] and NEAREST LATTICE VECTOR [3]. Finally, Class IV consists of the hardest problems, namely those for which it is NP-hard to achieve an n^{ϵ} approximation factor for some $\epsilon > 0$. This class includes problems such as CLIQUE [34] and COLORING [58].

Because we will use these problems in several reductions, we give both definitions and notation for the SET COVER and LABEL COVER problems. For the SET COVER problem, we are given a ground set U of items, and a collection of subsets of these items, $S_1, S_2, \ldots, S_n$. The output is a subcollection of subsets so that every item of U is contained in at least one of the chosen subsets. The goal is to minimize the number of chosen subsets.

The LABEL COVER problem is defined as follows. The input is a regular, bipartite graph, $G = (U, V, E)$, a set of labels $\{1, 2, \ldots, N\}$, and for each edge $e \in E$, a partial function $\Pi_e : \{1, 2, \ldots, N\} \longrightarrow \{1, 2, \ldots, N\}$. A labeling associates a non-empty set of labels with every vertex in $U \cup V$, and is said to *cover* an edge $e = (u, v)$, if for every label b assigned to v, there is some label a assigned to u such that $\Pi_e(a) = b$. The goal of LABEL COVER$_{\min}$ is to give a labeling that covers all edges, while minimizing the total number of labels assigned to nodes of U.

Finally, we need to define our notion of *approximation-preserving reductions* [62, 64]. Classical reductions, for instance those equating all NP-complete problems, show that finding the optimal solution for one problem can be used to find the optimal solution for another problem. Unfortunately, such reductions do not guarantee anything about the relation between

[2]This factor, $2^{\log^{1-\gamma} n}$, lies between polynomial and polylogarithmic in that $2^{\log^{1-\gamma} n} = o(n^{\epsilon})$ for any $\epsilon > 0$, and $2^{\log^{1-\gamma} n} = \omega(\log^c n)$ for any constant c.

approximate solutions, and this, in part, explains the vast difference between the approximability of various NP-complete problems. Therefore, to compare the approximability of difficult problems, it is necessary to use such approximation-preserving reductions that show not only that finding the optimum of one problem can be used to find the optimum of the other, but also that an approximate solution of one can be translated to an approximate solution of the other, with a similar performance ratio.

Throughout this paper, we will use two types of reductions. For our purposes, we say that **problem $\mathcal{A}$ reduces to problem $\mathcal{B}$** if a polynomial-time algorithm for $\mathcal{B}$ which achieves an $f(n)$-approximation can be used to give a polynomial-time algorithm for problem $\mathcal{A}$ which achieves a $(1+c)\cdot f(O(n)) - approximation$ for some constant c. Notice that this allows us to introduce a certain amount of additive error in the approximation. Secondly, we say that problem $\mathcal{A}$ reduces to problem $\mathcal{B}$ with a **polynomial blowup** if a polynomial-time algorithm for $\mathcal{B}$ which achieves an $f(n)$-approximation can be used to give a polynomial-time algorithm for problem $\mathcal{A}$ which achieves a $(1+c)\cdot f(\text{poly}(n))$-approximation. Such a reduction generally results from a construction in which the problem size undergoes a polynomial blowup. In this sense, even if the absolute ratio of the approximation remains the same through the construction, the ratio as a function of n may become worse as the value of n has increased.

If problem $\mathcal{A}$ is known to lie in one of the four particular approximation classes above, and if we shows that problem $\mathcal{A}$ can be reduced to problem $\mathcal{B}$ by either of the above reductions types, this shows that problem $\mathcal{B}$ also lies in that same approximation class[3]. Both the additive error and the polynomial blowup do not effect the class in which a problem lies; they merely effect the exact constants shown in the hardness result. The reason that we differentiate between a reduction with or without a polynomial blowup is because of the effect that it has on trying to use these reductions to prove upper bounds. In a sense, a reduction from $\mathcal{A}$ to $\mathcal{B}$ implies that $\mathcal{B}$ is "at least as hard" as $\mathcal{A}$, however this intuition may be a bit misleading. If we were able to find a non-trivial approximation algorithm for $\mathcal{B}$, we may assume that this also results in a non-trivial approximation for $\mathcal{A}$, yet this is not necessarily the case with a polynomially blowup. For example, if we exhibit an $n^{1/2}$-approximation for $\mathcal{B}$, our reduction provides us with an n^a-approximation for $\mathcal{A}$, however we would have no guarantee that $a < 1$, and hence this result may not beat the trivial n-approximation for our problems.

[3]Actually, to place a problem into Class I through this type of reduction, it is necessary to show that the additive error can be made arbitrarily small.

2.3 Computational Geometry

Assembly sequencing is an intriguing combination of a combinatorial and geometric problem. Quite naturally, research from computational geometry relates very closely to assembly sequencing.

The separability of objects received increasing attention in the geometric community beginning in the mid 1980's. If we consider a collection of convex parts in two dimensions, a classic result of Guibas and Yao from 1983 states that for any given direction, there will always exist some part that can be translated to infinity in that direction without disturbing the others, and thus the single direction of translation can be used to repeatedly remove parts one at a time [31]. For convex parts in three dimensions, a theorem of Dawson from 1984 shows that for a collection of n balls in d-dimensions, there must exist at least $\min(n, d+1)$ balls that can each be individually translated to infinity without disturbing any other balls, while at the same time he provides a construction of convex parts in three dimensions for which no individual part can be translated to infinity [16]. Almost a decade later, Snoeyink and Stolfi were able to settle an open question by providing a collection of convex polyhedra in three dimensions for which no subset of parts could be separated from the remaining set, using either translations or combined translations and rotations. Similar issues on the separability of polygons were studied by Toussaint in 1985 for more general classes of shapes in two dimensions, such as monotone or star-shaped polygons [73]. Constructing a sequence for the individual removal of polygons from a collection in two dimensions was studied by Dehne and Sack in 1987 [19]. In this work, they introduce what the term the *movability wheel*, which can be though of as a very preliminary version of the NDBG. As a special case of separability, a sequence that removes objects one at a time, using translations in a common direction, is known as a depth order (such as in the result of Guibas and Yao). More recent work has looked at the efficiency of computing and verifying depth orders [14, 18], and a very complete discussion of depth orders is given by de Berg [17].

Related to our work in a general sense, there is a good deal of research regarding the approximation of problems in geometric settings. A surprising element of our results is that some of our general lower bounds given in Chapter 6.5, are realized in Chapter 7, for a simple geometric setting consisting of a collection of unit disks in the plane. More often than not, an optimization problem becomes significantly easier when its input is restricted to a geometric setting. For example, there exists some $c > 0$ for which achieving a $(1 + c)$-approximation for the Metric tsp problem is NP-hard [65], however in the Euclidean plane, TSP can be approximated to within $(1 + \epsilon)$ for all $\epsilon > 0$ [2]. Similarly, achieving an n^{ϵ}-approximation for Minimum Independent Set is NP-hard [34], however for planar graphs, Minimum Independent Set can be approximated to within $(1 + \epsilon)$ [6]. Similar results hold for most optimization problem when restricted to planar graphs [49]. There exists a $(1 + o(1)) \ln n$ lower bound for approximating the Set Cover problem [22], however the Rectangle Cover problem, covering a set of axis-aligned rectangles with minimum number of points, has no such inapproximability results [62]. Our lower bounds provide some of the strongest such inapproximability results for a natural, combinatorial, geometric problem. Similar lower bounds have been shown for the Nearest Lattice Vector problem [3], however this problem is not combinatorial, and although it shares the same lower bound as our problem, we cannot directly relate the hardness of the two problems.

Chapter 3

Virtual Assembly Sequencing (VAS)

3.1 Introduction

In this chapter, we introduce a formal framework for studying the cost of assembly sequences. Our model is a generalization based on the previous success of using non-directional blocking graphs (NDBG's) for constructing valid assembly sequences. We assume that our problem begins as we are handed the complete, representative set of directional blocking graphs, and our goal is to use these graphs to design a binary, monotone assembly sequence that optimizes some cost measure. Because of our strong reliance on the previous work with NDBG's, the reader is encourages to review Section 2.1.1 before proceeding.

Our reason for considering the graphs as the input to the problem is that this flow of control has been used in all geometric settings for which the NDBG framework has been successfully. The existing algorithms for different settings are specialized only in the use of geometric techniques for constructing the full set of blocking graphs. Once these graphs have been built, the algorithms succeed purely through the analysis of the graphs. The fundamental property is that the set of graphs represents the blocking conditions for all possible motions. Therefore, the graphs themselves contain all of the information necessary for identifying any feasible assembly sequences.

We define the *virtual assembly sequencing problem* (VAS), as this graph-theoretic generalization of the original assembly sequencing problem. We ignore the underlying geometry for this problem, by considering the sole input to be an arbitrary set of directed blocking graphs. It is important to understand that VAS is truly more general than the original problem, and this is precisely because we do not make any assumptions about the structure of the individual graphs, or their interdependence on each other. For this reason, any positive algorithms for VAS will immediately apply to all settings for which the NDBG can be computed efficiently.

Negative results for this model, however, do not immediately apply to the geometric settings. In reality, when an NDBG is constructed from a geometric description of a product, the resulting set of blocking graphs may have some hidden structure. It is conceivable that this structure would allow for additional success in devising assembly sequences, and therefore VAS may indeed be a strictly harder problem than the original assembly sequencing problem. We will consider the original geometric settings in Chapter 7.

After defining the problem formally, the remainder of this chapter addresses the fact that there are many variants of the problem that are useful for assembly sequencers, but that may require very different techniques to solve. Most obvious is that there is not a single, unanimous cost measure that all people will hope to optimize. Some applications will desire a sequence with as few steps as possible, whereas others may desire a sequence that uses as few distinct directions of motion as possible. In similar spirit, there are several different tasks that an assembly sequence can accomplish. The most common task is to completely disassemble a product into its individual parts, however there are other well-motivated tasks. As an example, the maintenance of a product may require a sequence that efficiently removes a certain part from the fully assembled product. Finally, we consider additional restrictions that may be placed on the structure of the input or output.

We create a catalog of variants by categorizing them according to the combination of the desired goal, the various restrictions, and the relevant cost measure. In this way, we build the groundwork for analyzing the approximability of the various cost measures in different settings. The majority of these variants have been discussed previously in the assembly sequencing literature, and so whenever possible, we cite previous research in which the various goals, restrictions, and cost measures have been studied.

3.2 Definition of VAS

We define the VAS problem as follows.

INPUT:	A set, $\mathcal{P}$ of n items.
	A family, $\mathcal{F}$, of directed graphs on n nodes, with $\|\mathcal{F}\| = \text{poly}(n)$.
OUTPUT:	An assembly sequence using only "legal" operations.

We will conventionally call each member of the set, $\mathcal{P}$, a "part," and we will call each member of the family, $\mathcal{F}$, a "direction." We inherit the definition of "legal" operations from the notion of directed blocking graphs. An edge from part A to part B in a particular graph signifies that if part A is moved using the associated motion, while part B remains stationary, then part A will collide with part B. Given a subassembly consisting of parts $\mathcal{P}' \subseteq \mathcal{P}$, it follows that a direction $d \in \mathcal{F}$ can be used to partition $\mathcal{P}'$ into sets S_1 and S_2 if the graph d has no edges directed from a part in S_2 to a part in S_1. In graph-theoretic terms, an operation is legal if the partition provides a *directed cut* on the subgraph of d induced by the set $\mathcal{P}'$.

3.2.1 Possible Goals

Originally, we said that the goal of an assembly sequencer is to produce a disassembly sequence that completely decomposes the original product into its individual parts. Although this is a common task, there are other variants that are highly motivated by industrial applications. The following contains a list of possible goals, along with their motivations. Each of these goals is defined based on the structure required of the resulting assembly tree (see Section 1.2).

G1 **Full disassembly.**
This is the classical problem. The goal is to find a sequence of operations that begins with the fully assembled product, and results in the complete decomposition into individual parts. Each leaf of the assembly tree must consist of an individual part.

G2 **Remove a key part.** Instead of disassembling the entire product, it is often desirable to quickly remove a single, key part from an assembly without necessarily disassembling the entire product. The motivation for this stems from issues of maintenance and recycling. The classic maintenance example is to replace a spark plug without taking the entire car apart. A classic recycling example is to strip down old computers for valuable parts while throwing out the rest.

For this variant, we assume that we are given a product as well as the label for one *key part* that is to be removed. In the resulting assembly tree, the key part must be isolated at a leaf, however other leaves may represent many parts, since there is no need to further decompose subassemblies that do not contain the desired part.

G3 **Remove a given set of parts.**

Rather than removing a single part, we may be asked to remove an arbitrary subset of parts. In this variant, each of the requested parts must be isolated at a leaf of the assembly tree. When only one part is requested, this is identical to goal G2, and if the entire set is requested, this is identical to goal G1.

G4 **Separate a given pair.**

Given a key pair of parts, the goal in this variant is to decompose the fully assembled product until the two parts lie in different subassemblies. This task is motivated by products for which it is important to identify the first operation that requires both of the key parts to be brought together. This may be important in situations where the two parts are manufactured at different locations, or for sensitive materials that need to be treated specially when they are brought together.

For this variant, the two key parts must be located in different leaves in the resulting assembly tree. Note that the two parts need not be completely isolated from the entire assembly, simply separated into components that do not include the other key part.

G5 **Separate a given set.**

Rather than a pair of parts, a set of parts is given here, and the goal is to decompose the assembly so that no two of the key parts are in the same subassembly. When the key set consists of two parts, this is exactly goal G4, and when the set consists of all parts, this is identical to goal G1.

3.2.2 Possible Restrictions

Often, manufacturing systems impose additional constraints on assembly sequences other than simply geometric feasibility. We consider a few such restricted versions of the assembly sequencing problem.

R1 **Linear Sequence.** [67, 76]

A *linear* assembly sequence is one in which each operation brings together a single part with an existing subassembly. Such sequences are reminiscent of a classical assembly line, in which each station is responsible for adding one part. Although not all products can be assembled linearly, such sequences are used in manufacturing for several reasons. The organizational level of a linear assembly line is much simpler than for a sequence that requires building many subassemblies in parallel. Also, the fact that one of the subassemblies is a single part greatly reduces the fixturing costs.

Therefore, we consider the additional problem of choosing the best such sequence for a product, when restricted to linear assembly sequences. Note that, even when restricted to linear sequences, there still may be exponentially many valid sequences for a given product.

R2 **Constant Size Family of Motions.** [1]

Originally, our only assumption was that the number of blocking graphs is polynomially bounded in the number of parts. Now we consider instances where the number of graphs in bounded by some constant, k.

For automated assembly systems, each motion type or direction may require a specialized robot, and thus manufacturing systems may be constrained to use only a small set of pre-defined motions based on the existing robotics system. For example, a system may be constrained to using axis-aligned translations.

When constrained to a small number of input graphs, it may be possible that there exist better sequencing algorithms than for the more general problem.

3.2.3 Possible Complexity Measures

How do we decide which of two assembly sequences is the better one for a given product? Of course, every person asked will give a different definition of which is better for that application. Furthermore, the truest measure of cost-efficiency may be a combination of many different factors. We begin the study of cost measures for assembly sequencing by introducing a collection of *primitive* complexity measures, motivated by specific aspects of industrial applications. Our view is that success with these basic measures is a necessary first step before examining specialized combinations of complexity measures.

C1 **Fewest Number of Directions.** [67, 76, 78]

The cost of an assembly sequence is equal to the number of directions of $\mathcal{F}$ that are used. Once a direction has been used, future uses of the same direction are free of charge. The motivation here is that in manufacturing, each direction requires a different type of movement for a robot, and it is more efficient to have robots that have as few degrees of freedom as possible. Note that this differs from restriction R2, in that we are not told which directions to use in restricting our search.

C2 **Fewest Re-orientations.** [46, 55, 78]
The cost of an assembly sequence is equal to the number of operations that use a direction that is *different* from the previous operation (we will also charge the very first operation). In many manufacturing situations, the main cost of a robot is in orienting it to perform a type of motion, yet once it is oriented, it is fairly inexpensive to perform several motions of that type. Similarly, in some manufacturing systems all parts must be physically inserted from above and thus an operation in a different direction is performed by re-oriented the subassembly on the assembly line so that the desired direction is aligned vertically. This is typically slow and might require additional expensive fixtures. In both of these cases, using an orientation that was encountered earlier in the process is of no advantage unless the product is still in that orientation.

C3 **Fewest Number of Non-Linear Steps.** [76]
An operation is linear if one of the two subassemblies is a single part. The cost of an assembly sequence is equal to the number of non-linear operations. The motivations for this measure are similar to those for the R1 restriction, however rather than absolutely requiring that all steps are linear, we simply attempt to minimize the use of non-linear operations.

C4 **Fewest Number of Steps.** [79]
The cost of an assembly sequence is equal to the total number of operations used. Notice that for goal G1, full disassembly, every possible *binary* assembly sequence will require exactly $(n-1)$ steps. Therefore, this cost measure is only meaningful for the partial disassembly problems.

C5 **Minimum Depth of an Assembly Sequence.** [76]
The cost of an assembly sequence is equal to the depth of the corresponding tree. The motivation here is that in many assembly environments, parallelism in production is helpful, and the minimum depth tree has the quickest throughput, in a sense. As a special case, when the goal is to either remove a key part or separate a key pair, then this cost is exactly equal to the *number of steps* taken, and thus equivalent to C4.

Chapter 4

Algorithms, Heuristics and Experiments

4.1 Introduction

Having formalized in Chapter 3 several problems of interest involving cost measures in assembly sequencing, the natural question is how well we can design algorithms that find good assembly sequences. As was mentioned in Section 2.1, there has been a great deal of success for finding feasible assembly sequences in many settings. There are two different software systems for assembly sequencing based on the NDBG framework, which are quite robust [46, 67]. Both of these packages are complete in finding feasible assembly sequences when they exist. Additionally, both of these systems offer their users the opportunity to optimize sequences over several of the cost measures that we consider. Unfortunately, these systems do not have efficient algorithms for optimizing most of these cost measures, and so they must either provide the user with a solution whose cost is not bounded versus the optimal cost, or else they must rely on exponential search algorithms for finding the true optimal solution.

In this chapter, we make a preliminary examination of what can be done algorithmically for the VAS problem described in the previous chapter. In Section 4.2 we make some immediate observations to explain the success had by current systems in finding feasible sequences, and in finding low cost solutions in certain settings. Some of our later observations present new results for finding near-optimal cost sequences for some of the variants of VAS. Following this, in Section 4.3, we examine some natural greedy algorithms for these problems and show why they are unsuccessful. Our constructions give examples of products for which the greedy algorithm performance quite badly. Not only are these algorithms bad in the worst case, but we feel that they perform badly even on common products for assembly sequencing. For this reason, in Section 4.5 we describe experiments we perform using a set of products, previously used as a test bed for the existing assembly sequencing software packages. We show how these common heuristics exhibit poor performance, even for these examples. In Section 4.4, we also consider the efficiency of computing the exact optimal solution, although this requires exponential time.

4.2 Immediate Observations

In this section, we look at variants of VAS which can be solved through fairly immediate observations from the definitions in Chapter 3. The first several of these observations explain the success of the NDBG framework for solving the *decision* problem of finding feasible assembly sequences. We review these results, however we use the terminology of VAS in order to acclimate the reader. The latter observations describe some variants that offer polynomial solutions or approximations for minimizing cost measures.

Observation 1 A graph admits a legal operation on a subassembly if and only if there exists a directed cut on the node-induced subgraph for the parts in that subassembly. The existence of a directed cut is equivalent to the fact that the subgraph is not *strongly connected.* This condition can be checked for each graph in polynomial time.

Observation 2 The removal of additional parts can never invalidate an action. That is, if an action using direction d is legal on the current subassembly, then the corresponding action will still be legal on the induced graph remaining after any number of intermediate actions take place.

The proof of this property is evident from the definition of directed cuts. If there are no edges going from some set S_1 to a set S_2, then removing nodes will certainly not invalidate the cut. This monotonicity is quite important, as we will see momentarily, in the ability to successfully find legal assembly sequences. It also has important consequences for approximations in that the "penalty" for making a wrong move at any point is somewhat minimal, in that the cost of the wasted move must be paid for, but the correct continuation is still available.

Observation 3 In polynomial time, we can check whether the set of graphs admits a feasible sequence for any of our goals. We are able to find a legal operation, if one exists, which decomposes our problem into two subassemblies, and then recurse. No operation is a mistake in terms of feasibility due to Observation 2, as the removal of parts can only decrease the blocking relationships of future moves.

Observation 4 A graph admits a valid *linear* operation to remove part p, if and only if part p has no outgoing (incoming) edges. This can be checked in polynomial time.

Observation 5 In polynomial time, we may check whether a set of graphs admits a feasible *linear* sequence for any of our goals. This is a result of Observations 4 and 2.

Observation 6 For all of our variants, we can approximate the minimum cost solution within a factor of $(n-1)$ in polynomial time. For approximating the minimum depth for full disassembly, it is possible to achieve an $\frac{n-1}{\lceil \log_2 n \rceil}$ approximation.

Proof: This is actually a trivial result of two facts. The first of which is that we are always able to determine *some* feasible solution in polynomial time, if it exists. The second fact is that for all of our cost measures, the best possible sequence has cost at least 1, and the worst possible solution has cost at most $n-1$. This can be verified for each cost measure, but in general it can be seen that for an assembly of n parts, any binary, monotone assembly sequence will require at most $n-1$ operations for any of our goals.

When minimizing the depth for full disassembly, the worst possible cost is still $n-1$, however we know that the best possible depth for any full tree must be at least $\lceil \log_2 n \rceil$. ▮

Observation 7 A *stack* assembly is defined in [76] as a product that can be completely (dis)assembled using translations along a single direction. They observe that a product admits a stack assembly sequence, if and only if one of the blocking graphs is *acyclic*, and that this can be checked in polynomial time.

Observation 8 When the family of graphs has constant size, we can find the minimum number of directions required for any of the five goals in polynomial time (R2/C 1). This is true with or without the linear restriction.

With a constant number of graphs, there are a constant number of possible subsets of directions, and so we may simply try each possibility and check the feasibility of the problem with those graphs. Each such feasibility check can be done in polynomial time due to Observation 3 or Observation 5.

Observation 9 When $|\mathcal{F}| = 2$, we can find the optimal solution in polynomial time, for minimizing the number of re-orientations for any of the five goals (R2/C 2). This is true with or without the restriction to linear operations.

Proof: When minimizing the number of re-orientations, once we use a graph for an operation, Observation 2 assures us that we may as well greedily continue using that same direction until all legal actions have been exhausted. At that point, since there are only two graphs, the only way to make progress will be to switch to the second graph and continue. The only difficulty in finding the optimal solution is to know which graph to start with at the beginning. However it is easy to compute the sequence that results for either of the two choices, and to use the best. This is guaranteed to have optimal cost. ▮

Observation 10 In polynomial time, we are able to find an $(|\mathcal{F}| - 1)$-approximation for minimizing the number of re-orientations, for any of our five goals. This is true with or without the restriction to linear operations.

Proof: Our algorithm proceeds in stages as follows. If there is currently a direction that allows us to meet the final goal, then we use that direction. Otherwise, we compile a list of all directions that allow progress, and order them arbitrarily. Now we proceed through the list, for each direction exhausting all possible actions before re-orienting. After processing the entire list of candidates, we repeat this algorithm.

We claim that this achieves an $(|\mathcal{F}|-1)$-approximation. Our first claim is that the number of stages of our algorithm is equal to at most the cost of the optimal solution. We prove this by induction on the number of stages. If our algorithm needs only one stage, then it found a move to solve the problem, and only used one step. If our algorithm used many stages, consider the optimum cost solution on the original subassembly. We claim that by the end of the stage, the optimum cost solution from that point on will be at least one less than for the original problem. This is a direct consequence of Observation 2. The optimal solution from this point must choose some useful direction for the next step, and that direction will be in our stage's list of candidates. Therefore our algorithm has at most OPT stages.

Now we prove the $(|\mathcal{F}| - 1)$-approximation. Other than possibly the first stage, we note that there will be at most $(|\mathcal{F}| - 1)$ candidates in each stage. The direction that ended our previous phase will not be a candidate since we just exhausted all legal operations for that graph. In the first stage we may have all $(|\mathcal{F}|)$ candidates, however in our last stage, we are assured to use only one candidate. If the optimal solution using only one re-orientation, we are assured that our algorithm will also have cost one. Otherwise, our algorithm is assured to give us a solution with cost at most:

$$(|\mathcal{F}|-1)(\mathrm{OPT}-2)+1+|\mathcal{F}| = (|\mathcal{F}|-1)(\mathrm{OPT}) - 2(|\mathcal{F}|-1)+1+|\mathcal{F}|$$
$$= (|\mathcal{F}|-1)(\mathrm{OPT}) + |\mathcal{F}| - 3 \leq (|\mathcal{F}|-1)(\mathrm{OPT})$$

The techniques we have used in this proof are very similar to those used for approximating the Shortest Common Supersequence problem [8]. For more discussion of the relation between re-orientations and supersequences, see the proofs of Theorems 31 and 33. ▮

Observation 11 For the goals G2 and G4, the following relationship is true:

$$\mathrm{OPT}_{\mathrm{C2}} \leq \mathrm{OPT}_{\mathrm{C4}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{C2}}$$

where $\mathrm{OPT}_{\mathrm{C2}}$ is the minimum number of re-orientations required, and $\mathrm{OPT}_{\mathrm{C4}}$ is the minimum number of steps required.

Proof: The left inequality is trivial, in that if there is a solution with S steps, then clearly that same solution has at most S re-orientations. The right inequality is a more subtle fact, in that we show that if there exists a solution with R re-orientations, then there also exists a solution with at most $2R$ steps.

For the goal of removing a key part, the optimal assembly tree has a structure such that there is only one operation at each level. In more intuitive terms, there is always one "main" subassembly that contains the key part, and each operation removes some set of parts away from the current main subassembly. Once a set of parts has been removed, there is no need to further decompose them, since this does not effect the goal of removing the key part. The same structure is true for the goal of separating two parts from each other, as the main subassembly contains both parts, and the first time the two parts are separated into different subassemblies, the goal is complete.

Based on Observation 2, we have repeatedly notice that if trying to minimize the number of re-orientations, once an operation has been made using a specific graph, then without loss of generality, we may as well continue to use that same graph free of charge until it affords no legal operation. Considering the graph-theoretic definition of legal moves, we find that this greedy approach will result in reducing the main subassembly down to the strongly connected component containing the key part (or key pair).

However, it is always possible to reduce to this strongly connected component in at most two steps, since there exists a partial order on the strongly connected components. In the first move, we are able to remove the collection of all strongly connected components that have no edges to our key component, since this will be guaranteed to provide a directed cut. After this step, a second step will always allow us to remove the key component from the remaining subassembly. In this way, we can use the optimal solution for re-orientations to create a solution that uses two steps for each re-orientation. This solution provides our upper bound on the optimal solution for minimizing the number of steps. ∎

Observation 12 For the goals G2 and G4, there exists a polynomial time approximation algorithm, which achieves a factor of $2(|\mathcal{F}| - 1)$ for minimizing the number of steps.

Proof: In the proof of Observation 10 we saw how to find a solution which uses at most $(|\mathcal{F}| - 1) \cdot \mathrm{OPT}_{\mathrm{C2}}$ re-orientations, and from the proof of Observation 11 we see how each re-orientation can be replaced by at most two steps, giving us a solution with at most $2(|\mathcal{F}| - 1) \cdot \mathrm{OPT}_{\mathrm{C2}} \leq 2(|\mathcal{F}| - 1) \cdot \mathrm{OPT}_{\mathrm{C4}}$ steps. ∎

4.3 Greedy Heuristics

In trying to develop reasonable approximation algorithms for these problems, one of the first ideas is to try a *greedy* algorithm in some respect. That is, we can assign some easily computable measure of goodness for the result of each possible option, and then always select the option that appears to be best. This type of algorithm could take many forms. To be more concrete, we will consider a specific setting, namely removing a key part.

Consider the goal of removing a part using as few steps as possible (G2/C4). We start with the complete assembly, and if it is possible to simply remove the key part in a single step, we can of course do that right away. Otherwise, we will have to choose some operation that will strip some set of parts away from the subassembly containing the key part. Those other parts are no longer relevant, and we simply have the same problem again on a smaller subassembly. In this setting, we know that there will be at most $n - 1$ steps in a feasible solution, since at least one part is removed in each step. A natural greedy algorithm to consider is that which chooses the operation at each step that removes the most parts.

Unfortunately, this algorithm can have a performance ratio of $\Omega(n)$, even on an input with four graphs. Imagine that our goal is to remove part 1, and that we have n parts overall with

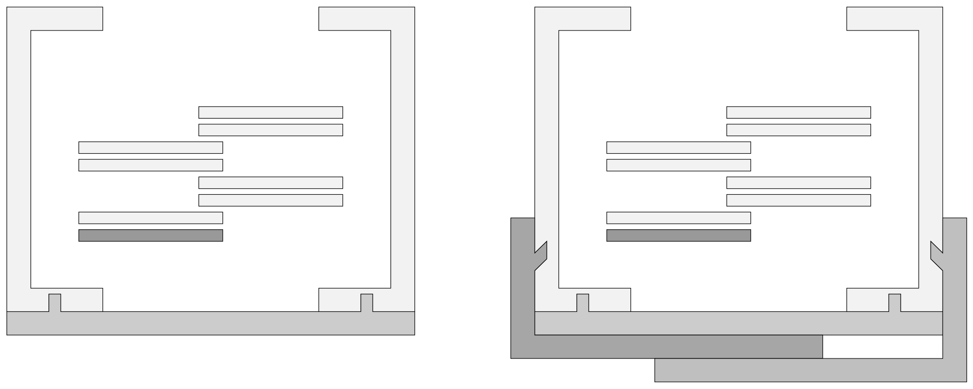


Figure 4.1: Bad Inputs for a Greedy Algorithm

n even. We create the following four graphs. G_A is a star graph with node 2 as the center, that is, it has edges $(k, 2)$ and $(2, k)$ for all nodes $k \neq 2$. Notice that if part 2 was gone, this graph would allow for the complete disassembly and thus also the removal of part 1, however with part 2 there, it is strongly connected and thus useless. Our second graph G_B is a clique on all nodes *except* node 2, and thus it allows node 2 to be removed by itself, but otherwise is useless[1]. Notice that these two graphs alone would allow a solution for removing part 1 in two steps, the first of which removes part 2. However we can easily distract a greedy algorithm with the addition of two more graphs. We define G_C to have edge (i, j) for all nodes $i < j$, and additionally edge $(k, 1)$ for all nodes with $k = 0, 1 \bmod 4$. Similarly, we define G_D to also have edge (i, j) for all nodes $i < j$, and additionally edge $(k, 1)$ for all nodes with $k = 2, 3 \bmod 4$. These new graphs have the following properties. At any point, one of the these two new graphs will allow a step that removes the two parts with the largest labels. Once those parts are gone, the other new graph will allow the next two parts to be removed, and so on.

Given this input, the greedy algorithm will choose to remove the two highest labeled parts at each step, and thus will use roughly $\frac{n}{2}$ steps to remove part 1, even though the optimal solution has cost two. This same construction provides an $\Omega(n)$ lower bound on the performance of this algorithm for minimizing the number of re-orientations.

This may seem like a somewhat artificial trick, however the construction can easily be realized geometrically, with polygons in two dimensions, as shown in the left side of Figure 4.1. In fact, such a worst case input is not at all pathological for assembly sequencing. Products

[1]These gadgets, and others, are formalized in Section 6.3, where $G_A = \text{STAR}(2)$ and $G_B = \text{RELEASE}(2)$.

designed by engineers do not at all resemble sparsely filled parts in general positions, rather they are extremely compact and often quite intentionally include many such gadgets as the one we have used.

Other Greedy Metrics

There are many other possible intuitive ideas for a greedy algorithm. We can think of the general framework for a greedy algorithm, based on a goodness function that it computes for all possibilities. That is, the function can compute all choices for the next choice, and the resulting subassembly for each choice. In the previous section, we considered our measure of goodness to be minimizing the number of parts in the resulting subassembly, however there are an endless number of other ways to define the concept of goodness. In general it seems rather easy to construct similar counter examples for each of them. In this section, we consider one more such algorithm.

The intuitive problem with our previous greedy algorithm is that there is a strongly connected graph, G_A, that seems useless, but of course becomes quite useful if only part 2 is removed. A natural intuition is to instead consider the goodness measure of a subassembly to be the number of edges in the sparsest of the graphs. Notice this goodness metric also incorporates the size of the subassembly in some respect, because if many parts are removed, then many edges will also disappear. However this goodness function has the additional intuitive advantage in that it will always make progress on breaking these more sparse graphs if possible. At least in the exact counter example we gave for the previous greedy algorithm, this new algorithm will succeed.

Unfortunately, it is possible to modify our previous construction in order to defeat this new greedy algorithm. Originally, we made graph G_A to be a star on node 2, and thus it was a sparse graph, and easily seen to be potentially useful. We can replace graph G_A by a graph that is a clique on all parts except for 1, and then has the edges $(1, 2)$ and $(2, 1)$ connecting our key part to the clique. In this new construction, the most sparse graphs will always be one of G_C or G_D, and the greedy choice will always to be to remove one of the pairs of extraneous parts.

Greedy with Lookahead

From an intuitive standpoint, the pitfall in the previous constructions should be easy to locate. In a sense it is very clear looking at graph G_A that the removal of part 2 would be extremely advantageous, and for this reason an algorithm may determine that it should remove that single part rather than follow the immediate greedy choice.

We can consider a modified greedy framework that uses a lookahead of t steps as follows. Originally, the algorithm first constructed a list of all possible immediate choices, computed the resulting subassembly for each, and then evaluated the goodness for each result, choosing the best. Instead of considering a single choice, we could instead consider all possible sequences of the next t choices, compute the result of each such sequence, and then evaluate the goodness and choose the best sequence. So long as t is a constant, this too can be done in polynomial time, although the degree of the polynomially depends heavily on t.

Unfortunately, we can modify our counter example construction quite easily to defeat any constant size lookahead as follows. For example, if $t = 2$, rather than allow the immediate release of the gadget part 2, we could instead modify graph G_B so that part 3 is used to glue part 2 to the rest of the assembly. Then we could introduce a new graph which allows for the immediate release of part 3. Now the optimal solution would require only three steps, but a lookahead of two steps would not provide any reason to choose that path. Again, we are able to realize this modification geometrically, as seen in the right side of Figure 4.1.

4.4 Computing the Optimal Sequence

Currently, assembly sequencing systems that offer the user the option of optimizing a cost measure must rely on either heuristics with no performance guarantees on the cost of the resulting sequence, or else on possibly exponential search techniques to find the true optimal. For our experiments in Section 4.5, we will be interested in the cost of the optimal solutions, in comparison with the cost of solutions produced by efficient heuristics, and so in this section, we discuss a bit about our experience in attempting to compute optimal solutions "efficiently." We were able to compute optimal sequences for many sample assemblies, reaching the limits of our resources for sequences requiring 10–15 steps, for an assembly of 30–40 parts. Of course, our resources can always be increased to extend the depth of our results further, however the exponential growth of both time and memory quickly becomes limiting for assemblies of reasonable complexity.

The techniques developed for pruning any search depend greatly on the exact variant of the problem, and the cost measure being minimized. We concentrate on the goal of removing a key part using the minimal number of steps. The first algorithm for finding the optimal is to do a (clever) brute force search through the space of possible sequences for removing the part, until a successful sequence is found. By exploring the possible sequences with a breadth first search, we can be sure that the first successful sequence that removes the desired part will be that of minimal cost.

More importantly, we can generally restrict a large part of the search space by only considering subassemblies which arise through valid operations. For instance we can start by computing all *valid* operations that can be performed on the original assembly, and then again consider all valid operations on each of those results. A data structure for representing all the feasible subassemblies has been suggested in [36], where the AND/OR graph is introduced[2]. The advantage of this approach is that, rather than examining all of the potential 2^n subsets of the parts, time can be saved by only considering those subassemblies which arise naturally. However the disadvantage is that the size of an AND/OR graphs is exponentially, and thus the memory required to restrict ourselves only to feasible subassemblies quickly becomes limited. In our experiments, the memory usage for an assembly of 42 parts had exhausted over 500M of swap space after computing only sequences of length 12 steps or less. Alternatively, it is possible to forego restricting the search to only valid subassemblies, and instead use a memory-less algorithm that simply performs a breadth-first search on all possible sequence of operations, however in this case the computation time becomes a limiting factor much earlier.

Finally, searches through exponential spaces can often be improved computationally using branch-and-bound search techniques. An important requirement for this success is to be able to compute efficiently a reasonable lower bound on the optimal cost solution from various points, in order to bound the search down certain paths of the space. For this reason, we investigate how well we can efficiently compute reasonable lower bounds on the optimal cost solutions in our experiments. For this work, we again consider removing a key part, however we restrict ourselves to linear operations, and consider the number of steps required. Equivalently, in this setting we are minimizing the number of parts removed in accessing the key part. We consider the following technique for constructing a lower bound on the cost of a solution.

A simple lower bound for the removal of a given part is the minimum number of others parts blocking it, over all possible directions of motion. That is, if a part is blocked by 5

[2]Not to be confused in any way with the model of scheduling with AND/OR precedence constraints in Chapter 5.

parts in all directions, then a solution for removing that part certainly will require at least 6 linear moves overall. This provides us with a lower bound, but it will not necessarily be a very good lower bound. For example, we can imagine our part being blocked in some direction only by a single part, yet one which is very difficult to remove in its own right. However, we can extend this type of lower bound to produce stronger and stronger lower bounds, at the expense of more and more computation. If we consider this first lower bound as having lookahead of 1, we can consider a lower bound with lookahead 2 as follows. For a single direction, we compute the parts that block our key part. We know that if we use that direction eventually, then this will require the prior removal of all of these blocking parts. Additionally, we can now consider one of these blocking parts, and recursively compute our lower bound on the number of *additional* parts that would have to be removed, in order to first remove this blocking part. If we choose the lower bound for the worst such blocking part, then we get a stronger lower bound for the removal of the keypart in this direction. Next, we can be assured that the minimum, over all possible original directions, of this new lower bound is a true lower bound for the removal of our key part. This technique can naturally be extended to an arbitrary level of lookahead, recursively, providing better and better lower bounds. A rough estimate of the computational time for computing this lower bound with lookahead t, is equal to $\Theta((n\mathcal{F})^t)$.

4.5 Experiments

We have seen how several simple algorithms can be made to perform quite badly on particular inputs. Our feeling is that the gadgets used in our construction are not at all pathological, and that these simple heuristics should perform quite badly on various products. For this reason, we performed experiments using input drawn from various sources. This section described these experiments, including the results and our conclusions.

4.5.1 The Model Assemblies

For our experiments, we consider several model assemblies that have been used as a test bed by previous research on assembly sequencing [46, 67, 74]. These models consist of three dimensional polygonal parts, and we consider their disassembly using infinitesimal translations to separate parts. The NDBG's for products have been constructed using either the STAAT [67] or ARCHIMEDES [46] assembly sequencers. Redundant blocking graphs were then removed, and the remaining family of blocking graphs is used as input for our experiment.

Description: An electric bell, modeled with varying levels of detail [74].
Courtesy of Randy Wilson.

Description: A model aircraft engine, modeled with varying levels of detail [74].
Courtesy of Randy Wilson.

Description: Snoeyink and Stolfi describe a model of a 30 part assembly of convex parts in three dimensions, for which there is no legal separation [70]. We have deleted one of the pieces to provide an interesting model that may be disassembled.
Courtesy of Jack Snoeyink.

File	$\|\mathcal{P}\|$	$\|\mathcal{F}\|$
bell9	9	5
bell17	17	5
bell22	22	5
eng12	12	5
eng23	23	12
eng30	30	13
eng42	42	13
sno29	29	1250

4.5.2 Experiment Setup

In our experiments, we consider two scenarios. The first is to minimize the number of total steps for removing a key part when restricted to linear operations (G2/R1/C4). We use the models from Section 4.5.1 for our input, however we enforce the restriction to linear operations.

When restricted to linear moves, the greedy heuristic for minimizing the size of the subassembly is irrelevant, as every step removes exactly one part. Minimizing the number of edges in a connected component is no longer relevant, as we are not allowed to remove a large subset, even if it were unconnected from the key part. We study the algorithm that chooses a move uniformly at random. We also compute several lower bounds for the optimal cost, as described in Section 4.4. For the removal of each individual part, we compute the following:

- **OPT-lin** – This is the cost of the optimal linear sequence. It was computed using techniques described in Section 4.4.
- **lb1, lb2, lb3, lb4** – These are the lower bounds on the optimal cost, computed with lookahead of 1 through 4 using the techniques described in Section 4.4.
- **rand1** – We consider the randomized algorithm that computes all possible choices, as before, but then picks a choice at random. For each experiment, we report the average cost of 1000 random trials.
- **rbest** – The cost of the *best* solution found in the 1000 random trials used above.

The second scenario we consider is to minimize the number of re-orientations while removing a key part (G2/C2). For each of our models, we considered the removal of each part individually, and computed the following.

- **OPT** – This is the cost of the optimal sequence. It was computed using techniques described in Section 4.4.
- **size1** – This is the greedy algorithm described in Section 4.3. This greedy algorithm considers all possible choices of direction, and then chooses the direction that will minimize the resulting number of parts left in the subassembly containing the key part after all legal actions for that direction are exhausted.
- **size2** – This is the same algorithm as the previous, except that it uses a lookahead of two, as described in Section 4.3.
- **edges1** – This is the greedy algorithm described in Section 4.3, with lookahead of one.
- **edges2** – This is the greedy algorithm described in Section 4.3, with lookahead of two.
- **rand1** – We consider the randomized algorithm that computes all possible choices, as before, but then picks a choice at random. For each experiment, we report the average cost of 1000 random trials.
- **rbest** – The cost of the *best* solution found in the 1000 random trials used above.

4.5.3 Experiment Timing

For all of our experiments, we measured the timing of our computations by counting the number of calls to some core procedure, such as the procedure which given a direction and a subassembly, checks whether there is a valid operation using that direction for partitioning the subassembly.

4.5.4 Experiment Results

In this section, we report the results of our experiments. First we show the results for the first scenario, with the linear restriction. Following this we report the results of the second scenario, which did not have the linear restriction. For each of the models in Section 4.5.1, we computed the various results for the removal of each individual part. Many parts for each subassembly can be removed in the first step, and hence do not provide interesting results since all of our heuristics first check to see whether the goal can be reached before trying other options. We have chosen to report the results for all parts that require two or more steps in the optimal sequence. At the bottom of each table, the time we report is the average over all parts listed in the table, using the timing units as discussed in Section 4.5.3. We provide the actual tables in Appendix A.

4.5.5 Experiment Conclusions

We discuss our results for the two scenarios separately. We will start by examining the *second* scenario in our experiment, that without the restriction to linear moves. Unfortunately, for the most part, it seems our models are not very challenging. Except for the 'sno29' assembly, the removal of a part never required more than three re-orientations in the optimal sequence. Although they do not always find the exact optimal sequence, the heuristics generally are at most a few steps away from the optimal cost. However, this is also an effect of the limited choices given to each heuristic, as we first remove all clearly dominated choices. To begin with, most of these examples only have 5 to 12 graphs, and so at each step there are relatively few graphs to which we can re-orient. This would help explain the very good success of the randomized heuristic that is given the same choices at each point, and simply chooses uniformly at random which move to follow. The 'sno29' assembly, however, provides quite a different image. Recall, this is very interlocked assembly, and as we can see, even the optimal sequences for removing many parts require a significant amount of re-orientations, given the relatively small number of parts. As we get a more rich example, we see that the gap between the optimal solution and some of the heuristics has begun to grow. Also, there are some cases where it seems that the greedy heuristics do fall into so-called "traps." For example, for removing part 14 from this product, we see that the optimal solution requires 4 re-orientations, however the greedy algorithm, *size1*, uses 13 re-orientations. However, as soon as the lookahead is increased to two, the algorithm, *size2*, finds the true optimal cost solution. Also, we see that the random choices are no longer doing quite as well.

If we consider the first scenario, which has a restriction to linear operations, these same products seem to provide us with a much richer test bed, in that there are a wide range of values for the optimal sequence for removing various parts. We see that the randomized algorithm performed quite poorly in these trials. When restricted to linear operations, there may still be many more choices at each point, even with few graphs, because a large number of the parts may be removable. In this sense, it is not surprising that choosing a random part to remove is not successful, as the solution can often be only steps away, while randomly chosen moves may remove parts in a completely different area of the product. We often see trials for which the optimal sequence only requires two or three steps, yet the random trial removes over half of the parts on average.

The more interesting result here is that the lower bound technique seems to converge quite well towards the optimal cost. We see that by the time we have applied our lower bound technique with a lookahead of three of four steps, we are rarely very far away from the true cost of the optimal solution. Unfortunately, computing these lower bounds with lookahead becomes quite expensive, and as we see, rarely is the time required for computing these best lower bounds very far away from the time spent for finding the true optimal solution. In general, these experiments confirm our intuitions, in that there are no readily available efficient heuristics that perform well as the complexity of products increases. Continuing to rely on exponential time procedures is quite restrictive, as we hope to be successful on larger and more complex products.

Chapter 5

AND/OR Scheduling

5.1 Introduction

In this chapter, we will focus entirely on VAS when restricted to *linear* sequences (R1), and when the cost measure is equal to the minimum number of steps required (C4). In this setting, we can view the VAS problem in a more simple manner by modeling it as a scheduling problem. We consider the removal of each part as a task that can be scheduled, and the linear disassembly sequence is simply a schedule for the order of removal. The cost of the solution is exactly equal to the number of tasks that are scheduled[1].

Of course, our tasks have certain precedence constraints relating their order of removal. That is, it may be the case that a certain part cannot be removed until after some other parts are removed. What distinguishes this setting from more traditional scheduling is the form of the precedence constraints. Commonly, a task may have what we term an AND-precedence constraint, in that it has an associated set of tasks, all of which must be scheduled before that task [23]. Unfortunately, this is not the case in our assembly sequencing problem. When considering a single direction, if a part is to be removed, then indeed there is a clear set of associated parts that block the removal, and thus all of these parts must be removed prior to removing our part in that direction. However, we may choose to remove that same part in some other direction, in which case a different set of parts may block the removal.

For this reason, we must consider another model for the structure of the precedence constraints. The blocking relationships for linear disassembly sequences can be modeled directly using what one may choose to call DNF scheduling, where the precedence constraint

[1]Much of this chapter involves work done in collaboration with Chandra Chekuri and Sanjeev Khanna.

for the removal of a part consists of a disjunction, with one disjunct for each distinct direction of removal, and where each disjunct is a conjunction of the parts that block the removal in the given direction. For example, it may be that the removal of some part F may have a precedence constraint of the form $(A \wedge B) \vee (A \wedge C \wedge D)$.

We will choose, however, to examine a more specific model, namely that of AND/OR precedence constraints. In this model, the precedence constraints for a given task must either be a disjunction or a conjunction. Notice that AND/OR scheduling is simply a special case of DNF scheduling. We choose to consider the AND/OR scheduling problem for several reasons, most notably because we will use this problem as the base of a reduction to prove hardness of a geometric setting in Chapter 7. However AND/OR scheduling is an interesting problem in its own right, and has been studied some in previous literature. Theorem 25, in Chapter 6, will specifically address the issues of modeling VAS using AND/OR precedence constraints (as opposed to DNF constraints).

We have not yet mentioned the exact goal for this scheduling problem. For the remainder of this chapter, we will assume that we are interested in removing a key part (G2), and therefore the goal of the scheduling problem is simply to schedule a key task. However, the other goals from Section 3.2.1 may also be translated naturally to the scheduling model.

The remainder of this chapter proceeds as follows. In Section 5.2, we give a formal definition of the AND/OR scheduling problem, as well as some associated notation and terminology. Previous work concerning this scheduling problem is reviewed in Section 5.3. Our main results of this chapter are strong inapproximability results for several variants of AND/OR scheduling, given in Section 5.4. We show that scheduling a key task, while approximating the minimum number of scheduled tasks falls into Class III of the Arora/Lund approximation hierarchy (see Section 2.2), and thus approximating the cost within a factor of $2^{\log^{1-\gamma} n}$ for any $\gamma > 0$ is quasi-NP-hard. We also show these hardness results for several constrained versions of the problem that will be used in proving hardness results in Chapters 6 & 7.

Finally, we feel that the problem of scheduling with AND/OR precedence constraints raises several important complexity issues, of considerable interest in their own right. This form of precedence constraints is a fairly natural extension to the traditional scheduling problem, yet our results show that the effect of this change on the difficulty of the problem is quite dramatic. We pose a series of open directions of research, in Section 5.5, related to the theory of approximability and where this problem fits in relation to several standard problems.

5.2 Notation and Definitions

We define the problem of scheduling with AND/OR precedence constraints as follows.

The input contains a set of tasks, $\mathcal{T}$. Each task, $t_i \in \mathcal{T}$, is labeled as either an **AND-task** or an **OR-task**. Each task, $t_i \in \mathcal{T}$, has an associated set of tasks, P_i, as direct **predecessors**; we refer to $|P_i|$ as the **degree** of the task. An AND-task, t_i, cannot be scheduled until after *all* tasks in P_i. An OR-task, t_j, cannot be scheduled until after *at least one* task of P_j. The **max AND-degree** of an instance is the maximum size $|P_i|$ over all AND-tasks t_i. The **max OR-degree** of an instance is the maximum size $|P_j|$ over all OR-tasks t_j.

The constraints can be represented by a **precedence graph**, with a node for each task, t_i, and a directed edge from t_i to t_j whenever $t_j \in P_i$, is a direct predecessor of t_i[2]. A **leaf-task** is one with $P_i = \emptyset$, and thus no outgoing edges. Such a task can be scheduled at any time. We say that an instance of AND/OR scheduling has **partial-order** precedence constraints if there are no cycles in the precedence graph. We say an instance of AND/OR scheduling has **internal-tree** precedence constraints if there are no cycles, and if all non-leaf nodes have at most one incoming edge.

The goal for this problem is to successfully schedule a specific task, and the cost is equal to the total number of tasks which must be scheduled. We consider a single processor and unit processing time for all tasks.

Additionally, we will consider the problem of minimizing the number of scheduled *leaves*, when constrained to internal-tree precedence constraints. Notice that internal-tree precedence constraints define a monotone, boolean formula on the leaf nodes, in which setting a leaf's variable to "one" signifies that the leaf will be scheduled. Minimizing the number of scheduled leaves is equivalent to satisfying a monotone, boolean formula with the minimum number of ones. We are unaware of any previous results for this exact approximation problem. Minimizing the number of ones in satisfying a 3CNF formula is known to be $n^{0.5-\epsilon}$-hard to approximate [44], and related minimization problems are studied in [50].

5.3 Previous Work

Although the topic of scheduling with precedence constraints has a long and rich history [23, 29, 53], there has been relatively little research focusing on models such as AND/OR

[2]We choose, in this situation, to direct an edge from t_i to t_j, to be consistent with the notion of edges in a directed blocking graph. Often the meaning of the directed edge is reversed in scheduling literature.

precedence constraints. However, a series of papers by Gillies *et al*, study several variants of scheduling with AND/OR precedence constraints [24, 25].

Our model for this problem was chosen to be similar to [24, 25], however with one key difference. As mentioned in Section 5.2, the precedence constraints for an instance can be represented as a directed graph. In this previous work, only the case of partial order precedence constraints is considered. Notice that in traditional scheduling, with AND-precedence constraints, the existence of a cycle in the precedence constraints makes the scheduling problem infeasible, and thus a partial order is a standard assumption. With AND/OR constraints, this absence of cycles is no longer a necessary condition for the existence of a valid solution. In fact, cycles will often exist in instances drawn from assembly sequencing, as it may be the case that part A blocks part B in one direction, part B blocks part C in another, and part C blocks part A in a third direction. For this reason, we make no apriori assumptions about the structure of the precedence constraints.

The work of [24, 25] studies a larger variety of settings, including multiple processors, deadlines, and individual processing times. They prove the NP-hardness of finding feasible schedules in many settings that are polynomially solvable with more traditional AND-precedence constraints, however they do not consider the approximability of the corresponding optimization problems. Additionally, they present several priority-driven heuristics that extend several multi-processor results from traditional precedence constraints to AND/OR precedence constraints. In their terminology, our setting is equivalent to minimizing the completion time of an AND/OR/skipped task system, with one processor, and unit processing times.

5.4 Inapproximability of AND/OR Scheduling

5.4.1 Our Results

It is worth noting that with classical AND precedence constraints, the problem of minimizing the number of scheduled tasks can be solved exactly, in polynomially time, by computing a depth order. Every task that blocks the goal must be scheduled, as does every task that blocks one of those tasks, and so on. As we will see, the situation is quite different with AND/OR precedence constraints.

In this section, we prove a series of results to show that it is quasi-NP-hard to find a solution to any of the following problems that is within a factor of $2^{\log^{1-\gamma} n}$ of the optimal

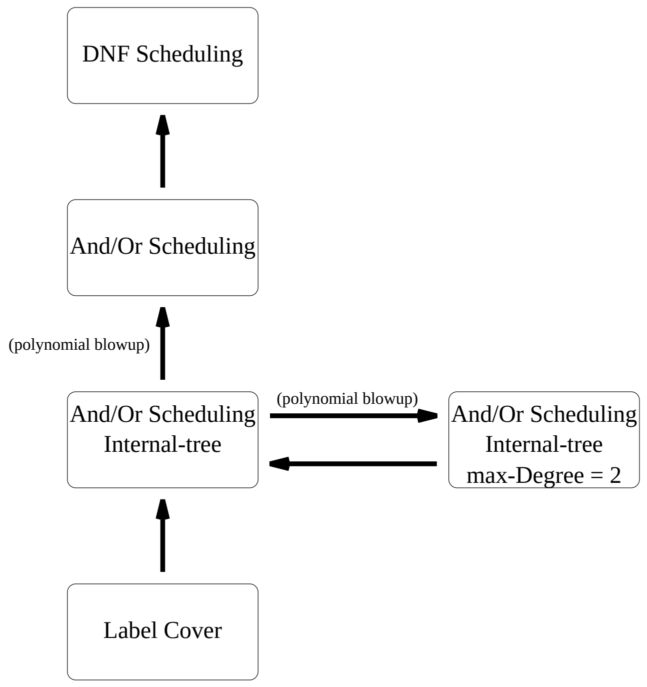


Figure 5.1: Reductions between variants of AND/OR scheduling

solution, for any $\gamma > 0$.

- Minimize the number of scheduled *leaves*, for an instance of AND/OR scheduling with internal-tree precedence constraints.
- Minimize the number of scheduled *leaves*, for an instance of AND/OR scheduling with internal-tree precedence constraints, and max-degree bounded by two.
- Minimize the number of scheduled *tasks*, for an instance of AND/OR scheduling.
- Minimize the number of scheduled *tasks*, for an instance of AND/OR scheduling with max-degree bounded by two.
- Minimize the number of scheduled *tasks*, for an instance of DNF scheduling.

5.4.2 Proofs

We begin by considering the AND/OR scheduling problem when restricted to *internal-tree* precedence constraints, as defined in Section 5.2. We will look at the problem where we only charge an algorithm for the *leaves* that it schedules. We show the inapproximability of this problem by showing that the LABEL COVER$_{\text{min}}$ problem ([3, 4]) is a special case. Following this, we show that the lower bound applies even when we further restrict the AND/OR problem to have degree bounded by two. Finally, we convert this bound on the number of leaves scheduled into a bound on the total number of scheduled nodes for the general AND/OR scheduling problem. This also proves the hardness of DNF scheduling, as AND/OR scheduling is simply a special case. An overview of the reductions is given in Figure 5.1.

Theorem 13 *The* LABEL COVER$_{min}$ *problem is simply a special case of minimizing the number of* leaves *scheduled in an instance of* AND/OR *scheduling with* internal-tree *precedence constraints.*

Proof: Given an instance of LABEL COVER$_{\text{min}}$, defined in Section 2.2, we express it as an instance of AND/OR scheduling with internal-tree precedence constraints, as shown in Figure 5.2. The AND/OR instance has five levels, which alternate between AND-nodes and OR-nodes. The highest level contains solely the root of the internal-tree, and the lowest level contains exactly the leaves. The tasks at the five levels are as follows:

- The first level has a single AND-node, which is the root of the internal-tree. This task enforces that every node in V must have a non-empty set of labels.
- The second level has an OR-node for each vertex in V. This node requires that for a given node v to have a non-empty label set, at least one label must be assigned to it.
- The third level has an AND-node for each pair $\langle v, l' \rangle$, where $v \in V$, and $l' \in \{1, \ldots, N\}$. This node signifies that for label l' to be assigned to vertex v, it must be the case that for each edge $e = (u, v)$ incident to v, the mapping Π_e on that edge, must respect the labeling.
- The fourth level has an OR-node for each pair $\langle e, l' \rangle$, where $e = (u, v)$ is an edge, and l' is a label. If l' is to be assigned to v, then edge e cannot be covered unless one of the pre-images of l' from mapping Π_e is assigned to u.
- The fifth level has a leaf for each pair $\langle u, l \rangle$, and corresponds to label l being assigned to vertex u.

This completes the construction. It can be seen that there is a one-to-one correspondence between valid labeling in the LABEL COVER$_{\text{min}}$ instance and valid solutions to the AND/OR

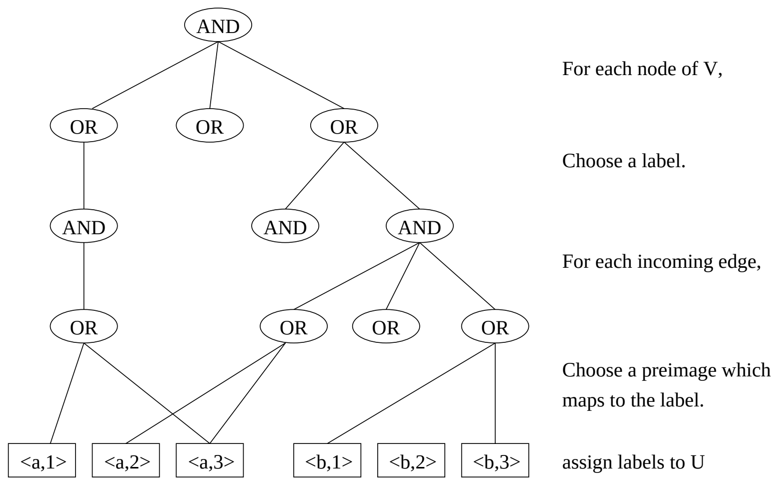


Figure 5.2: Label Cover as AND/OR scheduling with internal-tree precedence

scheduling instance, where the number of labels used is exactly equal to the number of leaves scheduled. It is easy to verify that the AND/OR instance has internal-tree precedence constraints. Notice that the number of non-leaf tasks in this construction is polynomially bounded in the size of the Label Cover$_{\text{min}}$ instance (namely, in $|U|$, $|V|$ and N), and thus any polynomial time approximation algorithm for this AND/OR scheduling problem, also serves as a polynomial time algorithm for Label Cover$_{\text{min}}$. ▮

Corollary 14 *It is quasi-NP-hard to approximate the number of* leaves *scheduled in an instance of* AND/OR *scheduling with* internal-tree *precedence constraints, to within a factor of* $2^{\log^{1-\gamma} n}$ *for any* $\gamma > 0$.

Proof: Approximating the Label Cover$_{\text{min}}$ problem within a factor of $2^{\log^{1-\gamma} n}$ is quasi-NP-hard for any $\gamma > 0$ [4]. Combining this lower bound with Theorem 13 proves our result. ▮

Theorem 15 *Even if both the* AND*-degree and* OR*-degree are bounded by two, it is quasi-NP-hard to approximate the number of* leaves *scheduled in an instance of* AND/OR *scheduling with* internal-tree *precedence constraints, to within a factor of* $2^{\log^{1-\gamma} n}$ *for any* $\gamma > 0$.

Proof: We give an approximation-preserving reduction from the problem of AND/OR scheduling with internal-tree precedence constraints but unbounded degree, to the same problem with max-degree bounded by two. Combining this reduction with Corollary 14 proves our result.

Given an instance with unbounded degree, we can bound the degree in the obvious way, by replacing each internal node with a tree of bounded degree nodes. Assume there were originally I internal nodes and L leaves, and that I is polynomially bounded in L. The maximum degree for any node is at most $(I + L)$, and thus that node can be replaced by a tree of at most $(I + L)$ nodes, each with degree two. Notice that the cost of the solutions has remained unchanged, as we are only charged for the number of *leaves* that are scheduled. The new instance has at most $I(I + L)$ internal nodes, which is still polynomially-bounded in L, and thus our reduction runs in polynomial time. ▮

Lemma 16 *If there exists an algorithm that achieves an $f(n)$-approximation to the number of* tasks *scheduled in an instance of* AND/OR *scheduling, then there exists an algorithm that achieves a $(1 + \epsilon)f(poly(n))$-approximation for any $\epsilon > 0$ for the number of* leaves *scheduled in an instance of* AND/OR *scheduling with internal-tree precedence constraints. This remains true even if both the* AND*-degree and* OR*-degree are bounded by two for each problem.*

Proof: Assume we are given an algorithm, A, for the general AND/OR scheduling problem that achieves an approximation ratio of $f(n)$. We show how to use this algorithm to approximately solve an instance of minimizing the number of scheduled *leaves* for AND/OR scheduling with internal-tree precedence constraints. The difficulty of attempting to minimize the number of scheduled leaves, using an algorithm that minimizes the number of scheduled nodes is that the overhead of the internal nodes may have a significant cost, changing the quality of the approximation. This can be remedied quite easily, by artificially increasing the cost of scheduling a leaf.

Assume we have a hard instance of internal-tree scheduling from Theorem 15, with I internal nodes and L leaves. We convert this to a general instance of AND/OR scheduling by replacing each leaf with a chain of αI new nodes, for some constant α. Notice that both the AND-degree and the OR-degree remain unchanged, although this new instance no longer has internal-tree precedence constraints. This new instance has a total of $I + \alpha IL$ nodes. Relying on the fact that I is polynomially bounded by L, we see that the new size is also polynomially bounded by L.

We rely on two useful facts in our construction. First, if the optimal solution of the original problem scheduled a total of $\mathrm{OPT}_{\mathrm{orig}}$ leaves, we are assured that

$$\mathrm{OPT}_{\mathrm{new}} \leq \alpha \cdot I \cdot \mathrm{OPT}_{\mathrm{orig}} + I.$$

Secondly, given any solution to the new problem that schedules t total nodes, we can create a solution to the original problem such that its cost is at most $\frac{t-I}{\alpha I}$. This is done by simply choosing to schedule those leaves for which the associated chain was completely scheduled. Combining these facts, we now analyze the approximation ratio of running algorithm A on our newly constructed instance.

Let A_{new} be the number of nodes scheduled by algorithm A for this new instance, and let A_{orig} be the number of leaves scheduled by our solution to the original problem that we can construct from the solution to the new problem.

$$\begin{aligned}
A_{\text{orig}} &\leq \frac{A_{\text{new}} - I}{\alpha I} \leq \frac{A_{\text{new}}}{\alpha I} \leq \frac{f(\text{poly}(L)) \cdot \text{OPT}_{\text{new}}}{\alpha I} \\
&\leq \frac{f(\text{poly}(L)) \cdot \left[\alpha \cdot I \cdot \text{OPT}_{\text{orig}} + I\right]}{\alpha I} \\
&= f(\text{poly}(L)) \cdot \text{OPT}_{\text{orig}} + \frac{f(\text{poly}(L))}{\alpha} \\
&= \text{OPT}_{\text{orig}} \left(f(\text{poly}(L))(1 + \frac{1}{\alpha \cdot \text{OPT}_{\text{orig}}}) \right) \\
&\leq \text{OPT}_{\text{orig}} \left(f(\text{poly}(L))(1 + \frac{1}{\alpha}) \right)
\end{aligned}$$

By choosing α appropriately, we see that we have constructed an algorithm for the original problem that achieves a $(1+\epsilon)f(\text{poly}(n))$-approximation. Since the new problem has size that is polynomial in the original problem, our new algorithm runs in polynomial time. ▮

Theorem 17 *It is quasi-NP-hard to approximate the number of* tasks *scheduled in an instance of* AND/OR *scheduling, to within a factor of* $2^{\log^{1-\gamma} n}$ *for any* $\gamma > 0$. *This remains true even if both the* AND*-degree and* OR*-degree are bounded by two.*

Proof: We combine the reduction of Lemma 16, with the result of Theorem 15. If there did exist a $2^{\log^{1-\gamma} n}$-approximation algorithm for this problem for some $\gamma > 0$, then by Lemma 16, there would exist an approximation algorithm for the case of internal-tree precedence constraints that achieves a factor of $(1+\epsilon)2^{\log^{1-\gamma} \text{poly}(n)}$. However, this provides a $2^{\log^{1-\gamma'} n}$-approximation for any $0 < \gamma' < \gamma$, which is quasi-NP-hard by Theorem 15. ▮

Corollary 18 *It is quasi-NP-hard to approximate the number of* tasks *scheduled in an instance of* DNF *scheduling, to within a factor of* $2^{\log^{1-\gamma} n}$ *for any* $\gamma > 0$.

Proof: Since general AND/OR scheduling is a special case of DNF scheduling, this corollary follows directly from Theorem 17. ▮

5.5 Open Problems

5.5.1 Alternating AND/OR Levels

In Section 5.4, many of our results involve the special case of AND/OR scheduling with internal-tree precedence constraints. We looked at the problem of minimizing the number of leaves that are scheduled in order to successfully complete the root of such a tree. As we pointed out in Section 5.2, this problem is equivalent to minimizing the number of ones that are used in satisfying a monotone, boolean formula. An interesting series of open questions involves the structure of this problem as we vary the depth of the tree.

Without loss of generality, we can assume that the root of our tree is an AND-node, and if we do not have a bound on the out-degree of the internal nodes, we can always collapse the internal nodes into alternating levels of AND-nodes followed by OR-nodes, eventually followed by a single level of leaves. Now we can consider the complexity of the problem based on the number of alternating levels.

If we consider one full alternation, an AND-node at the root, followed by a level of OR-nodes, followed by a level of leaves, this problem is *exactly* equivalent to the SET COVER problem. Every instance of SET COVER can be written as a suitable scheduling instance and vice versa. To see the connection, we equate each leaf with a subset, and each OR-node with an item in the universe. The precedence constraint for each OR-node enforces that one of the subsets containing the associated element must be chosen.

If we consider two full alternations, as we saw in Theorem 13, we can already express LABEL COVER$_{\mathrm{min}}$. However it is not at all clear whether or not this problem is equivalent to LABEL COVER$_{\mathrm{min}}$, and we conjecture that it is strictly more expressive. If we attempt to use our intuition of AND/OR scheduling from Figure 5.2, in creating an instance of LABEL COVER, we run across two obstacles. The first obstacle may be the more technical issue, namely that the bipartite graph used in LABEL COVER must be regular. If we attempt to create a bipartite graph based directly on the precedence constraints, there is no guarantee that all of the tasks' degrees in a given level will be the same. It may be possible to pad an instance to artificially enforce the regularity condition, however the second obstacle may be a more serious issue. If we are to think about the leaves as corresponding to (node,label) pairs, then there will be a natural structure partitioning the leaves into groups based on the corresponding node of the pair. That is, by the "natural" interpretation, no node will ever depend on leaves from more than one group of this underlying partition, since that node is enforcing the covering of a specific edge. We leave it as an open question to decide whether or not two alternating levels is strictly more expressive than LABEL COVER$_{\text{min}}$.

Furthermore, what happens when we go to three full alternations, or to an arbitrary depth internal-tree? Does this hierarchy collapse at some point, and if so when? Can the inapproximability bounds be strengthened for any of these depths? What if we consider the general problem without internal-tree precedence constraints?

There are several areas of research that may prove beneficial in answering some of these questions. The first is a study of constraint satisfaction problems, in which they consider the the problem of minimizing the number of ones required to satisfy a collection of constraints on boolean variables [50]. Their main result is that there are a finite number of distinct levels of approximability for minimizing the number of ones needed in satisfying such constraint systems. These results, however do not apply to this problem as their constraint systems must be expressible using constraints that are bounded arity functions on the final variables. However, their work may relate to our problem when both the depth and max degree are bounded by a constant, a case we have not considered.

Secondly, there is a great deal of previous work related to the size and depth of boolean circuits, including those for monotone boolean formulae [51, 66, 80]. It is clear than an arbitrarily complex formula on n leaves can be collapsed into an AND/OR tree with a single alternating level, where the top choice is of picking one of the satisfying assignments, and for each satisfying assignment, you must schedule all of the leaves that correspond to variables set to one. The problem here is that the number of internal nodes in this representation is no longer polynomial in the number of leaves, and this condition was necessary for our reductions. It may be possible to use some of the previous work in circuit complexity to strengthen some of our inapproximability results for AND/OR scheduling.

5.5.2 Summary of Open Problems

In Section 5.4, we consider several versions of this scheduling problem, giving reductions from one to another, and then proved a lower bound of $2^{\log^{1-\gamma} n}$ against the approximability of all of these problems by showing that the easiest of these versions captures the LABEL COVER$_{\text{min}}$ problem as a special case. It is open to determine a separation between any of the steps of the series of reductions. That is, the LABEL COVER$_{\min}$ results provide our strongest results even for the most general AND/OR scheduling problem, yet there is reason to believe this may be an even more difficult problem. It is already conjectured that LABEL COVER is truly n^ϵ-hard to approximate for some $\epsilon > 0$ [4], a result that would carry over through all of our reductions. However it may be possible to strengthen the lower bounds for AND/OR scheduling without necessarily settling the LABEL COVER conjecture.

Secondly, we offer no non-trivial (i.e. $o(n)$) approximation algorithms for any of these problems. It so happens that there is no known, non-trivial approximation algorithm for LABEL COVER the "easiest" of our problems. However our reductions are a bit misleading in this respect, because of the polynomial blowup in the input size. It may be possible to achieve a non-trivial upper bound for many of our variants of AND/OR scheduling, without providing such an upper bound on LABEL COVER. For example, the existence of an n^α-approximation for some $\alpha < 1$ for minimizing the number of scheduled tasks in a general instance of AND/OR scheduling problem, would guarantee through our reductions, an n^β-approximation for LABEL COVER$_{\min}$, however it may be that $\beta > 1$, in which case the approximation would be trivial.

In summary, for each of the following variants, it is open to present an n^α-approximation algorithm for some $\alpha < 1$. Also it is open to prove that approximating any of these problems is n^ϵ-hard for some $\epsilon > 0$.

- The number of scheduled tasks for an instance of AND/OR scheduling
- The number of scheduled tasks for an instance of AND/OR scheduling with bounded depth
- The number of scheduled leaves for an instance of AND/OR scheduling with internal-tree precedence constraints
- The number of scheduled leaves for an instance of AND/OR scheduling with internal-tree precedence constraints and bounded degree

Chapter 6

Inapproximability of Virtual Assembly Sequencing

6.1 Introduction

In this chapter, we present our core results regarding the VAS model, as we prove the difficulty of finding optimal or even near-optimal solutions for the many variants of VAS defined in Chapter 3. These problems can be shown to encompass the combined difficulty of many previously studied sequencing and covering problems.

In many cases, these problems can be shown to be NP-complete, proving that it is unlikely that an efficient algorithm will be able to find the best assembly sequencing for a given input. However, as we discussed in Section 2.2, we are also interested in how well an algorithm is able to find a solution that can be shown to have a cost near to the optimal solution's cost. A company using the result of an assembly sequence in practice may not care about the mathematical problem of whether they have the absolute best sequence, so long as they can be assured that they have found a sequence, more quickly, whose cost is almost the same. Unfortunately, we show that for most variants of VAS, finding an efficient algorithm that can guarantee to produce even a near-optimal sequence is difficult. Throughout this chapter, we consider the approximability of the problems that we study.

This chapter proceeds in two phases. First, we attempt to relate the many different variants of VAS to each other, so that we can determine whether certain problems are easier or more difficult than others, thereby pointing the direction of research towards attacking the easier problems. Section 6.4 presents this series of reductions which relates variants of VAS to each other. Secondly, we establish inapproximability results, even for many of the "easiest" of these problems, by showing that they capture several previously studied, difficult problems from the literature. These hardness results are proven in Section 6.5. Before demonstrating the full reductions, we present an overview of our inapproximability results in Section 6.5, and we attempt to give some intuition in Section 6.3, with hands on practice for working with examples of the VAS problem and presenting several useful gadgets to be used later in the chapter.

6.2 Our Results

The following three tables give a summary of the lower bounds we prove in this chapter against the approximability of VAS in various settings. Table 6.1 summarizes the hardness of the problem of removing a key part from the rest of the assembly. Table 6.2 summarizes the hardness of the problem of separating a key pair of parts from each other. Table 6.3 summarizes the hardness of the full disassembly problem.

6.3 Useful Gadgets

In this section, we introduce several artificial graphs which provide useful gadgets for designing constructions involving VAS later in this chapter. As a side benefit, examining these graphs gives us our first, true practice at interpreting graphs as directional blocking graphs and understanding their effects on the separability of the parts. We will see how these gadgets can be used to create various behaviors such as releasing a part from the rest of an assembly, or in locking two parts together. Also, we will see that the operations allowed by such graphs change when restricted to certain subassemblies.

As a second benefit, this chapter provides us with our first intuition regarding the difference between virtual assembly sequence and true, geometric assembly sequencing. Later in this chapter, we prove many very strong inapproximability results, largely because we are always able to add a specially designed graph into the family of motions, giving us a desired behavior. When we consider geometric settings in Chapter 7, inserting a particular graph into the family of motions is not always possible. These graphs represent the blocking relationships between the parts, and are determined by the geometric description of all the parts. In certain situations, there are graphs which quite simply, are not realizable geometrically. In

Table 6.1: Inapproximability of removing a key part

	G2 No restrictions	G2/R1 Linear	G2/R2 Constant Family	G2/R1R2 Constant Family & Linear
Directions C1	$2^{\log^{1-\gamma} n}$-hard Theorem 22 from G2/R1/C4	$2^{\log^{1-\gamma} n}$-hard Theorem 22 from G2/R1/C4	solvable Observation 8	solvable Observation 8
Re-orientations C2	$2^{\log^{1-\gamma} n}$-hard Theorem 22 from G2/R1/C4	$2^{\log^{1-\gamma} n}$-hard Theorem 22 from G2/R1/C4	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert-1)$-approximable Observations 9 & 10 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 31 from LTSP	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert-1)$-approximable Observations 9 & 10 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 31 from LTSP
Non-Linear Steps C3	$2^{\log^{1-\gamma} n}$-hard Theorem 26 from G2/C4	n.a.	$\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 26 from G2/R2/C4	n.a.
Steps C4 or C5	$2^{\log^{1-\gamma} n}$-hard Theorem 22 from G2/R1/C4	$2^{\log^{1-\gamma} n}$-hard Theorem 24 from AND/OR	$2(\lvert\mathcal{F}\rvert - 1)$-approx Observation 12 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 31 from LTSP	$\lvert\mathcal{F}\rvert \geq 2$: $2^{\log^{1-\gamma} n}$-hard Theorem 24 from AND/OR with maxdeg = 2

Table 6.2: Inapproximability of separating two parts from each other

	G4 No restrictions	G4/R1 Linear	G4/R2 Constant Family	G4/R1R2 Constant Family & Linear
Directions C1	$2^{\log^{1-\gamma} n}$-hard Theorem 19 from G2/C1	$2^{\log^{1-\gamma} n}$-hard Theorem 19 from G2/R1/C1	solvable Observation 8	solvable Observation 8
Re-orientations C2	$2^{\log^{1-\gamma} n}$-hard Theorem 19 from G2/C2	$2^{\log^{1-\gamma} n}$-hard Theorem 19 from G2/R1/C2	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert-1)$-approximable Observations 9 & 10 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 31 from LTSP	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert-1)$-approximable Observations 9 & 10 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 31 from LTSP
Non-Linear Steps C3	$2^{\log^{1-\gamma} n}$-hard Theorem 26 from G4/C4	n.a.	$\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 26 from G4/R2/C4	n.a.
Steps C4 or C5	$2^{\log^{1-\gamma} n}$-hard Theorem 19 from G2/C4	$2^{\log^{1-\gamma} n}$-hard Theorem 19 from G2/R1/C4	$2(\lvert\mathcal{F}\rvert - 1)$-approx Observation 12 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 31 from LTSP	$\lvert\mathcal{F}\rvert \geq 4$: $2^{\log^{1-\gamma} n}$-hard Theorem 19 from G2/R1R2/C4

Table 6.3: Inapproximability of complete disassembly

	G1 No restrictions	G1/R1 Linear	G1/R2 Constant Family	G1/R1R2 Constant Family & Linear
Directions C1	$2^{\log^{1-\gamma} n}$-hard Theorem 21 from G2/C1	$2^{\log^{1-\gamma} n}$-hard Theorem 21 from G2/R1/C1	solvable Observation 8	solvable Observation 8
Re-orientations C2	$2^{\log^{1-\gamma} n}$-hard Theorem 21 from G2/C2	$2^{\log^{1-\gamma} n}$-hard Theorem 21 from G2/R1/C2	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert - 1)$-approximable Observations 9 & 10 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 31 from LTSP	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert - 1)$-approximable Observations 9 & 10 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 31 from LTSP
Non-Linear Steps C3	$2^{\log^{1-\gamma} n}$-hard Theorem 21 from G2/C3	n.a.	$\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 21 from G2/R2/C3	n.a.
Depth C5	$\frac{n}{\lceil \log_2 n \rceil}$-approximable Observation 6 $2^{\log^{1-\gamma} n}$-hard Theorem 23 from G2/R1/C5	n.a.	$\frac{n}{\lceil \log_2 n \rceil}$-approximable Observation 6	n.a.

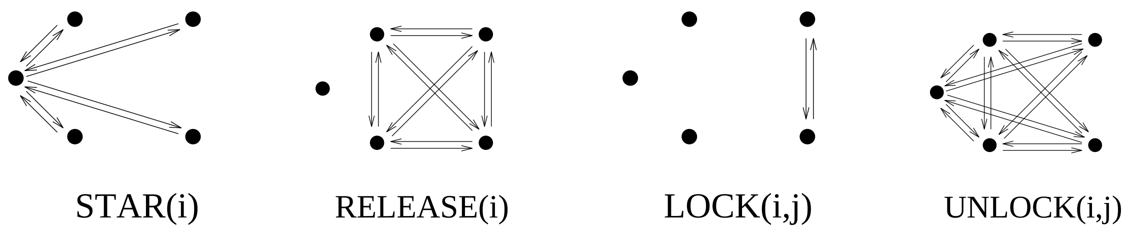


Figure 6.1: Useful Gadgets

other situations, even though a graph may be realizable, altering the parts in order to create this graph will change graphs for many other motions as a side effect.

Now we examine the four graphs pictured in Figure 6.1. The first graph, defined for every part i, we have named Star(i). This graph on n nodes, includes the edges (i, a) and (a, i) for all nodes $a \neq i$. It is clear that this graph is of no use on the fully assembled product, as it is strongly connected, and thus allows no legal operations. In fact the graph remains strongly connected for any subassembly which happens to contain the part i. However, for any subassembly which does not contain part i, we see that this graph will become quite useful, allowing all possible operations.

The second graph, also defined for every part i, we call Release(i). This graph is exactly the complement of Star(i), and includes edges (a, b), for all $i \neq a \neq b \neq i$. This graph has a very clear behavior as it is a clique on $(n-1)$ nodes. For any subassembly which includes the part i, this graph allows a single linear operation, namely to remove part i from the rest of the subassembly. For any subassembly without part i, this graph will be strongly connected. Therefore, this graph will either be ignored or used exactly once, for any legal assembly sequence.

A third graph, Lock(i, j), we define for any pair of nodes i and j, to consist simply of the two edges (i, j) and (j, i). This graph will allow all actions, other than those actions which separate i from j. The complement of this graph, Unlock(i, j), has quite a different behavior. On just about all subassemblies, this graph will be strongly connected, and thus of no use. In fact, the only assembly for which this graph will be helpful is if parts i and j are together in a subassembly with *no* other parts. In this specific two-part subassembly, this graph will allow the two parts to be separated.

6.4 Reductions Between Variants of VAS

In this section, we examine the relationships between many of the possible goals, restrictions, and cost measures given in Section 3. We give several approximation-preserving reductions which demonstrate that certain variants are at least as hard to approximate as others. Because of the sufficiently strong lower bounds we will prove in Section 6.5, we allow our reductions to have an additive error in the approximation factor, and where noted, we will allow a polynomial blowup of the input size. For a review of the reduction definitions, see Section 2.2.

An overview of the reductions is given in Figure 6.2, albeit without reference to the applicable cost measures and restrictions for each reduction.

6.4.1 Our Results

Theorem 19 [G2 $\Longrightarrow$ G4] (optionally C1, C2, C3, C4, C5, R1, R2, R1R2)
For all five cost measures, the problem of removing a key part from the rest of the assembly can be reduced to the problem of separating a key pair of parts from each other. This reduction also holds when restricted to linear steps, to a constant size family, or to both.

Theorem 20 [G4 $\Longrightarrow$ G2] (optionally C1, C2, C3, C4, C5, R1, R2, R1R2)
For all five cost measures, the problem of separating a key pair of parts from each other can be reduced to the problem of removing a key part from the rest of the assembly. This reduction also holds when restricted to linear steps, to a constant size family, or to both.

Theorem 21 [G2 $\Longrightarrow$ G1] (optionally C1, C2, C3, R1, R2, R1R2)
For minimizing the number of directions, re-orientations, or non-linear steps, the problem of removing a key part can be reduced to the problem of fully disassembling a product. This reduction also holds when restricted to linear steps, to a constant size family, or to both.

Theorem 22 [G2/R1/C4 $\overset{\text{poly}}{\Longrightarrow}$ G2] (optionally C1, C2, C4, C5, R1)
Minimizing the number of steps for removing a key part when restricted to linear moves can be reduced, with polynomial blowup, to minimizing either the number of directions, number of re-orientations, depth, or number of steps for the problem of removing a key part, with or without a restriction to linear steps.

Theorem 23 [G2/R1/C5 $\overset{\text{poly}}{\Longrightarrow}$ G1/C5]
Minimizing the number of steps for removing a key part when restricted to linear moves can

be reduced, with polynomial blowup, to minimizing the depth for full disassembly without the linear restriction.

Theorem 24 [AND/OR $\Longrightarrow$ G2/R1/C4] (optionally R2)
An instance of the AND/OR *scheduling problem can be written directly as a special case of the problem of minimizing the number of steps while removing a key part when restricted to linear moves. The number of graphs is exactly equal to the max* OR*-degree of the scheduling problem.*

Theorem 25 [G2/R1/C4 $\overset{\text{poly}}{\Longrightarrow}$ AND/OR]
Minimizing the number of steps while removing a key part, when restricted to linear moves, can be reduced with a polynomial blowup to the problem of AND/OR *scheduling. The number of tasks in the scheduling problem is bounded by* $(n+1)|\mathcal{F}|$*, and the max* OR*-degree is equal to* $|\mathcal{F}|$*.*

Theorem 26 [C4 $\Longrightarrow$ C3] (optionally G2, G3, G4, G5, R1, R2, R1R2)
Minimizing the total number of steps can be reduced to minimizing the number of non-linear steps. This is true for all applicable cost measures and all restrictions.

Theorem 27 [G3 $\Longrightarrow$ G2, G3 $\Longrightarrow$ G1]
The problem of removing a set of parts reduces to the problem of removing a single key part. Similarly, the problem of removing a set of parts reduces to the problem of fully disassembling a product. This is true for all cost measures and all restrictions.

Theorem 28 [G5 $\Longrightarrow$ G4, G5 $\Longrightarrow$ G1]
The problem of separating a set of parts from each other reduces to the problem of separating a pair of parts. Similarly, the problem of separating a set of parts from each other reduces to the problem of fully disassembling a product. This is true for all cost measures and all restrictions.

6.4.2 Proofs

Proof of Theorem 19:
The intuition for this reduction is simple. We take an instance of the problem of removing a key part k, and construct an instance of the problem of separating two parts by introducing a part k' which is "glued" to k unless all other parts are separated from k.

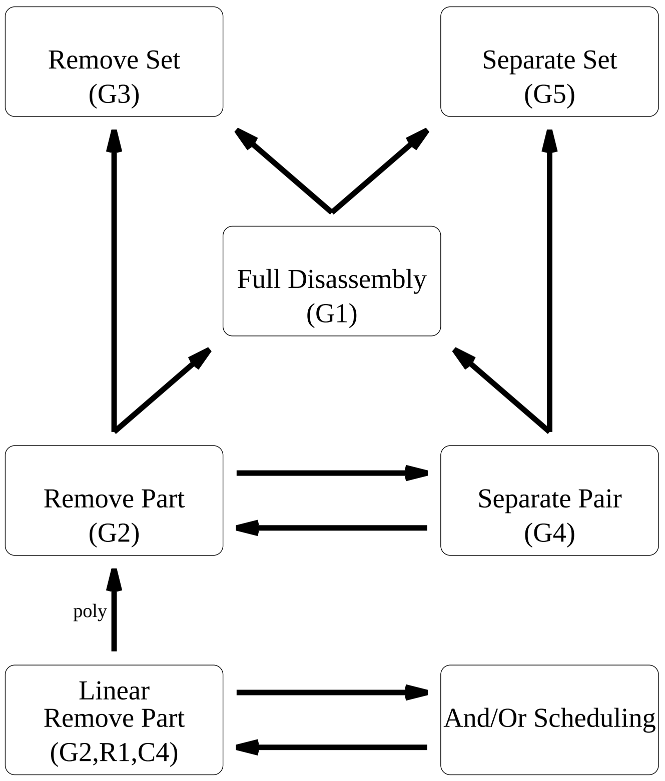


Figure 6.2: Reductions between variants of VAS

To implement this idea, we modify each graph in $\mathcal{F}$ by introducing part k' and adding edges (k, k') and (k', k). Therefore, no legal operation using one of these graphs can separate k and k' into different subassemblies. Finally, we add one new graph, Unlock(k, k'). If k and k' are the only parts in a subassembly, then this new graph will allow for their separation, however, it is useless for any other operations.

If we are not restricted to linear moves, we claim that there is a one-to-one correspondence between solutions of the two instances. Any solution for removing part k in the original problem, can be mimicked in the new problem to separate k and k' from the rest of the problem, and then one final operation using the extra graph can separate k and k'. Similarly, any solution to separate k from k' must end with such a move, and thus the rest of the sequence can be mimicked for the first problem. The input size for the reduction is increased by one part and one graph, and for all cost measures, the costs of the corresponding solutions differ by at most an additive error of one.

If we are restricted to using linear steps, however, there is a technical difficulty. In the original problem, the very last step in isolating part k will always be a linear step in which one of the subassemblies is the single part k. By the construction given above, k' would be glued to k, and we would replace the removal of k by the removal of the two-part subassembly, k and k'.

Therefore, if we are restricted to linear steps, we alter our reduction as follows. We replace each of the original graphs by two new graphs on $(n+1)$ nodes. The first is exactly the same as the original reduction, where we add edges (k, k') and (k', k). In the second new graph, we add all edges (a, b) for $a \neq k \neq b$, thereby creating a clique on the n nodes other than k, along with the original edges that connected k to other nodes. We claim that this graph will allow part k to be removed by a linear step, exactly when part k could have been removed by a linear step in the original graph, and that no other legal steps will exist. The only possible directed cut on this graph would be between k and the rest of a subassembly, and we claim this happens only if all of the outgoing (incoming) neighbors for part k are gone. This is exactly the case in which k could have originally been removed with a linear step, and this results in the separations of k from k'.

For this new reduction, the input size has again increased by one part, and the number of graphs has doubled. Again, we claim that for all cost measures and combinations of restrictions, the costs of corresponding solutions for the original and new problems differ by at most an additive error of one. This completes the reduction. ▮

Proof of Theorem 20:

We take an instance of the problem of separating parts k_1 and k_2, and we construct an instance of simply removing part k_1. We will create a new graph which allows k_1 to be separated from everything, so long as it had previously been separated from k_2.

In this reduction, we take each graph in $\mathcal{F}$ without modification, and we create one new graph with edges (k_1, k_2) and (k_2, k_1) along with edges (k_1, a) and (a, k_2) for all other parts a. This graph is shown in Figure 6.3. It is easy to verify that if both k_1 and k_2 are in a subassembly, then this graph is strongly connected and so no legal operations can be done. However, if k_1 has been previously separated from k_2, than this graph will allow k_1 to be separated from everything else in a single, linear step.

Again, any solution to one of these problems can be translated into a solution to the other with error of at most one for any of the cost measures, and the input has increased by one graph. ▮

Proof of Theorem 21:

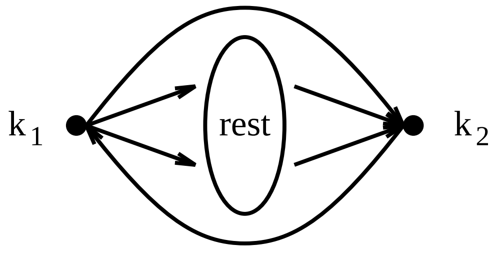


Figure 6.3: Separating a pair reduces to removing a part

Again, we give a very simple modification to translate an instance of the problem of removing a key part k, into an instance of fully disassembling a product. For this reduction, we simply create one additional graph which allows the entire product to fall apart if the key part is missing.

Specifically, we take each graph in $\mathcal{F}$ without modification, and insert one new graph, Star(k). For all subassemblies not containing k, this graph will allow complete disassembly with additional cost one, in terms of the number of directions, number of re-orientations or number of non-linear steps[1]. Furthermore, this graph is strongly connected, and hence of no use, for any subassembly which contains k.

This gives us an approximation-preserving reduction for these cost measures. Clearly, any solution to the new full disassembly instance can be translated to a solution to the original keypart problem with at least as low of a cost. Furthermore, the optimal solution for the full disassembly problem has cost at most one more than the optimal solution for removing the key part, namely using the new graph to finish the disassembly with additive cost one. ∎

Proof of Theorem 22:

Assume we are given an instance of minimizing the number of steps for removing a key part when restricted to linear operations. We construct a new instance of removing a key part, where we no longer require the restriction to linear operations.

We keep the same set of n parts, and we create a new family of $n|\mathcal{F}|$ graphs. Since $|\mathcal{F}| = \text{poly}(n)$, so is $n|\mathcal{F}|$. For each pair $\langle p, d\rangle$, with part $p \in \mathcal{P}$ and direction $d \in \mathcal{F}$, we create a new graph which is a clique on the $n-1$ parts $(\mathcal{P} - p)$, and which has edges (p, a) or (a, p) for each part a whenever the respective edge exists in d. This graph is identical to the graph we introduced in the second part of the proof of Theorem 19. We claim that the only possible action allowed by this graph is to remove the single part p from a subassembly,

[1]Notice that for Cost C5, there may be a logarithmic increase in the depth. This problem is handled separately in Theorem 23.

and that this one action is possible if and only if there is a linear operation which removes part p from the same subassembly using direction d. That is, the set of outgoing (incoming) edges of d must be broken in order to remove p in our new graph.

With this claim, we immediately get our result, as there is a one-to-one correspondence between solutions of the original problem and solutions of the new problem with identical costs. Every linear move in the original problem has a unique graph in the new problem that allows the identical part to be removed, and vice versa. Therefore, the number of steps in a solution to the original problem is exactly equal to the number of steps in a solution to the new problem. Furthermore, since each graph is useful for at most one linear move in our new instance, then the number of steps in the original solution is also equal to the number of steps, number of directions, or number of re-orientations used in the new instance.

Since all valid moves in our new instance happen to be linear, it makes no difference in the result whether the new instance is restricted to linear moves or not. Note that because our construction increased the number of graphs by a factor of n, we cannot make any such claim for problems with a constant size family of graphs (R2). ▮

Proof of Theorem 23:

The proof of this theorem is a simply combination of the techniques from the proofs of Theorems 21 and 22. In the proof of Theorem 21, we reduced the problem of removing a part to the full disassembly problem, by adding in one additional graph which was a star on the key part. In this way, once the key part is removed, this new graph can be used to complete the rest of the disassembly. The problem when considering the minimum depth cost measure was that the completion may still require an additional logarithmic increase in the depth.

We will remedy this by artificially increasing the cost of the original moves, so as to make this final logarithmic additive cost inconsequential. To do so, we must go back and reconsider the problem of removing a key part while restricted to linear operations, and so we assume we are given such an instance. We construct a new instance of minimizing the depth for full disassembly as follows.

We start by replacing every part a with n new parts $\{a_1, \ldots a_n\}$. In each graph, for any original edge (a, b), we introduce all possible edges (a_i, b_j). Additionally, we introduce all edges (a_j, a_i) such that $i < j$. If we are still restricted to linear moves, this has the effect that a_2 cannot possibly be removed until after a_1, and so on. However, if it was originally the case for a subassembly that part a would be immediately removable, in this current instance, it would be the case that a_1 could be immediately removed, followed in turn by a_2 and similarly the entire sequence of a's. Therefore, the overall effect of this replacement is simply to increase the cost of each original linear operation from 1 step to a sequence of depth n.

At this point, we continue in our construction as we did in Theorem 22, by relaxing the restriction to linear steps while artificially assuring that no non-linear steps will be possible. This results in an instance of the general problem of removing a key part, where the cost of our steps is still artificially replaced by a sequence of depth n. Now we can again add in one final new graph which is a start graph on part k_n, where k was the original key part for removing. Therefore, once k_n has been removed, this new graph allows for the complete disassembly of the remaining parts in depth logarithmic in the total number of parts. Our total number of parts has become n^2, and so the additive cost on the depth for this final group of steps is at most $\log n^2 = 2 \log n$. However, all of our steps for the original problem have been replaced by a sequence of n steps, making the effect of this final step insignificant, and thus we have an approximation preserving approximation, with a polynomial blowup in the size of the problem. ▮

Proof of Theorem 24:

Given an instance of AND/OR scheduling, we realize it as an instance of removing a given part, restricted to linear moves, while minimizing the number of steps. We create one part for each task in the scheduling problem. The number of graphs in our family of motions is exactly equal to the max OR-degree of the scheduling problem. By default, each of the graphs is complete, however we will delete the following edges. For an AND-task, t_i, we will modify the first graph by deleting edges (t_i, a) for all $a \notin P_i$. In this way, part t_i can be removed using this graph, if and only if all of its corresponding predecessors have been previously removed. For an OR-task, t_j, with degree Δ, we will modify the first Δ graphs as follows. For each $a \in P_j$, we will modify one of the graphs by deleting all edges (t_j, b) for all $b \neq a$. In this way, part t_j can be removed using this graph so long as part a is priorly removed. Therefore, if any one of the predecessors has been removed, then there will be some graph which allows for the removal of t_j with a linear move. This VAS instance is exactly the original AND/OR scheduling instance, where the number of (linear) steps required is equal to the number of scheduled tasks. ▮

Proof of Theorem 25:

Given an instance of removing a key part with linear steps, we create an instance of AND/OR scheduling as follows. Our intuition is to equate the removal of a part with the scheduling of *two* tasks, namely one task which says "I am prepared to remove part p in direction d"

and a second task which says "part p has been removed." We implement this as follows. For each part p we create task t_p, and for each pair (p, d), with part $p \in \mathcal{P}$ and direction $d \in \mathcal{F}$, we create task $\hat{t}_{(p,d)}$. We equate task t_p with the statement, "part p has been removed" and so we make it an OR-task with task $\hat{t}_{(p,d)}$ in its predecessor set for each direction d (we must first be prepared to remove the part in at least one direction). We equate task $\hat{t}_{(p,d)}$ with the statement "I am prepared to remove part p in direction d," and thus we make it an AND-task with task t_a in its predecessor set for every outgoing[2] edge $(p, a) \in d$.

For this construction, any solution to the scheduling problem can be translated to a solution of the VAS problem with the number of removed parts at most half of the number of scheduled tasks. Similarly every solution to the VAS problem can be translated directly to a solution to the scheduling problem with the number of scheduled tasks exactly twice the number of removed parts. Therefore, our absolute approximation ratio is preserved. However, our construction requires a polynomial blowup in the problem size, therefore what we have shown is that an $f(n)$-approximation algorithm for AND/OR scheduling gives us an $f(\text{poly}(n))$-approximation algorithm for this VAS variant[3]. ∎

Proof of Theorem 26:

Given an instance of VAS where the goal is to minimize the total number of steps, we wish to transform this instance into a new problem where we charge for the number of *non-linear* steps. The only difficulty is that when charging only for non-linear steps, any linear steps may be done for free. Our solution to this is quite simple. If we turn every part into two parts which are glued together, then an original linear step is no longer linear, and thus will be charged accordingly.

We implement this as follows. For every part a we create a new part a'. For every graph in $\mathcal{F}$ we add edges (a, a') and (a', a) for all parts, thereby gluing the new part to the original part. Finally we add one additional graph to the family of motions, defined as follows. This graph is a bi-directional clique on the entire set of nodes, with the edges (a, a') and (a', a) deleted for all parts a. For any subassembly which contains two original parts, such as a and b, we claim that this graph will be strongly connected, and thus of no use. However, for any two part subassembly such as a and a', this graph will allow the two parts to be separated.

[2]Actually, this only accounts for the possibility of removing part p away from the rest of the subassembly in this direction. As we can consider the "reverse" operation of removing the rest of the subassembly away from p, we should really construct two parts for each such pair, where the second checks for all incoming edges.

[3]If the family of graphs has constant size, then this construction simply requires a linear increase in size.

Notice that this separation can be done with a linear move, and thus in our new setting, we will not be charged for such an operation.

For any of the goals, we claim that there is a one-to-one correspondence between solution of the original setting, and solutions of the new setting which specifically use the new graph only at the end. Furthermore, we note that the costs of the corresponding solutions in their respective settings will be exactly the same, as all steps in the original setting, both linear and non-linear, have been turned into non-linear steps, and any steps using the newest graph are linear, and thus do not affect the cost. ▮

Proof of Theorem 27:
Both of these problems are simply special cases of removing a set of parts, as mentioned in Section 3.2.1. ▮

Proof of Theorem 28:
Both of these problems are simply special cases of separating a set of parts from each other, as mentioned in Section 3.2.1. ▮

6.5 Hardness Results

In the previous section, we gave reductions which related various versions of VAS to each other, in terms of the level of their inapproximability. Now we will rely on the hardness of some other problems to establish true inapproximability results for most variants of VAS. Our first set of results will be a direct consequence of the hardness of AND/OR scheduling shown in Chapter 5. Our second set of results will show weaker inapproximability results for a case not covered by our other reductions, namely minimizing the number of re-orientation when the family of motions is restricted to be of constant size. These results come from a natural reduction from the Loading Time Scheduling Problem [8].

Theorem 29 *It is quasi-NP-hard to achieve a $2^{\log^{1-\gamma} n}$-approximation for any $\gamma > 0$, for the problem of minimizing the number of steps while removing a key part when restricted to linear operations. This result applies even when $|\mathcal{F}| = 2$. (For one graph, this problem is polynomially solvable.)*

Proof: This is a direct result of Theorem 17, combined with Theorem 24. ▮

Corollary 30 *It is quasi-NP-hard to achieve a $2^{\log^{1-\gamma} n}$-approximation for any $\gamma > 0$, for minimizing the cost of many other variants of* VAS. *A summary of the results are given in Tables 6.1–6.3.*

Proof: These results are a combination of Theorem 29, together with the many reductions given in Section 6.4. See Tables 6.1–6.3 for the full results. Each table entry additionally refers to the relevant problem that is used to establish the lower bound as well as the theorem containing that reduction. ▮

Theorem 31 *We consider minimizing the number of re-orientations, when the family of motions is restricted to be of constant size, (R2/C2). For any of the five goals, we prove the following two results, (i) for $|\mathcal{F}| = 3$, this problem is NP-complete; (ii) for $|\mathcal{F}| \geq 4$, there exists an $\alpha > 0$, such that achieving an $|\mathcal{F}|^{\alpha}$-approximation is NP-hard. Both of these facts hold with or without the linear restriction, R1. Furthermore these same lower bounds can be shown for minimizing the number of steps (R2/C4) for all applicable goals.*

Proof: We begin by proving this result for the goal of removing a key part from the assembly. We will give a reduction from the Loading Time Scheduling Problem, defined as follows [8]. There is a set of n jobs, and ρ machines, and each job, j, can only be performed by some subset of the machines, $M(j)$. An algorithm pays for loading a machine, but once that machine is loaded, it may perform any available operations at no additional cost. Finally, the jobs have (standard) precedence constraints, represented by a directed acyclic graph G. The overall cost is the sum of the machine loading times for the sequence. They consider a weighted version where each machine m_i has loading time $l(m_i)$.

We give a reduction from the LTSP problem when all loading times are equal to 1, to the VAS problem of removing a key part (with or without the linear restriction), while minimizing the number of re-orientations. Given an instance of LTSP we create an instance of VAS with a part for each job in the LTSP instance, and one additional part, *key*, whose removal will be our goal. For each machine m, we create a graph $G_m \in \mathcal{F}$. The graph will be a superset of the precedence graph, G, given in the LTSP instance[4], augmented with the edge (key, i) for all jobs i, as well as the edge (j, key) for any job, j, such that $m \notin M(j)$. An example of such a graph is given in Figure 6.4.

[4]actually, we reversal all edges of G, as [8] defines an edge from x to y as signifying that y cannot be run until after x.

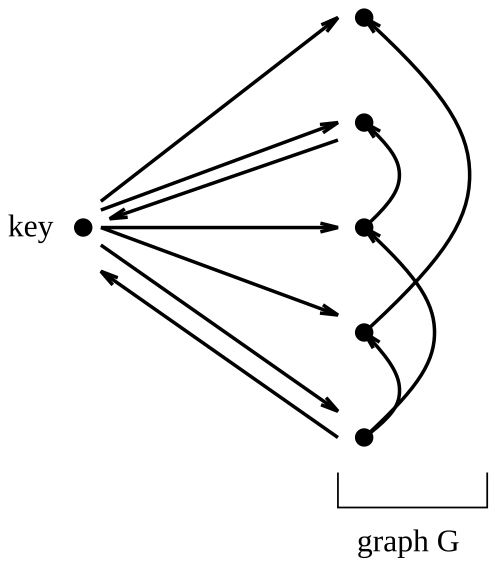


Figure 6.4: Reduction from Loading Time Scheduling Problem

We begin by considering the problem when restricted to linear moves, and we associated the removal of a part with the scheduling of a job in the LTSP instance. We claim that our graph G_m allows for the immediate removal of part j by a linear step, if and only if job j can be immediately scheduled on machine m. Assume that job j can currently be scheduled on machine m. In this case it must be that $m \in M(j)$ and that all predecessors of j have already been scheduled. But in this case we claim that vertex j has no outgoing edges in graph G_m, and thus can legally be removed using that graph. Since $m \in M(j)$, then vertex j does not have an edge to *key*, and since all of the predecessors of job j have been previously scheduled, then vertex j does not have any outgoing edges remaining from the original graph G. Similarly, if job j cannot be immediately scheduled, then it must be either because $m \notin M(j)$ or else one of the predecessors of j has not yet been scheduled. If $m \notin M(j)$, then both edges (key, j) and (j, key) exist and hence j and *key* cannot be separated. Instead, if one of the predecessors of j, call it b, has not yet been scheduled, than the edges (key, j), and (j, b) will exists, and thus node j has both an incoming and outgoing edge, and thus cannot be removed with a linear operation. For this reason, we claim that any solution to the LTSP instance can be translated to a solution with equal cost for removing the key part, and vice versa, and thus we have an approximation preserving reduction. At this point, we rely on results shown in [8] combined with a result of [59], to prove our claims, where the number of machines for LTSP corresponds to $|\mathcal{F}|$.

Notice that this exact construction, proves the result for the full disassembly problem, as our product will be fully disassembled exactly when all parts have been removed from the key. For the problem of separating a pair, we can use a trick similar to Theorem 19, by creating one new part, k', which is to be separated from k. In each of our existing graphs, we add the edge (k, k'), and edges (k', a) for all nodes $a \neq k$. In this way, part k' behaves as would a job that could be scheduled on any machine, but which has precedence constraints that require that all other jobs are scheduled before it. Therefore, the identical results hold for the problem of separating two parts.

This concludes the proof for all the cases when restricted to linear moves. If we allow non-linear moves, this construction still holds for the number of re-orientations. The fundamental observation is that if a single graph in our construction allows for a set of parts to be removed through a sequence of linear operations, then that graph also allows for the removal of all of those parts at once in a single operation. This is because there cannot be an edge going from one of these parts to one of the parts in the remaining subassembly, as if this were the case than the part with such an outgoing edge could not possibly have been removed with a linear step. This proves our result for re-orientations without the restriction to linear steps. Because all of the progress of each re-orientation can be made using a single step, the optimal number of steps is exactly equal to the optimal number of re-orientations. In this way, we get similar lower bounds for variant R2/C4, when applied to all goals except full disassembly. ∎

6.6 Open Problems

The foremost open problems for this chapter are to improve any of the lower bounds given in Tables 6.1–6.3. We will discuss several of the variants specifically.

- The most dramatic improvement would be to extend the current lower bounds of $2^{\log^{1-\gamma} n}$ to instead achieve n^{ϵ} lower bounds for the hardest variants.

- Is it possible to produce lower bounds for minimizing the number of non-linear steps or the overall number of steps, when restricted to a constant size family of graphs (R2/C3 and R2/C4)? The difficulty is the following. Our initial hardness results relied on the problem G2/R1R2/C4 where the problem was restricted to linear steps, and two graphs. In the proof of Theorem 22, we artificially enforced the restriction to linear moves, so that we were able to lift the restriction from the problem definition. However, our reduction required that we replaced each graph with linearly many new graphs, and therefore we were no longer able to restrict our problem to a constant size family of graphs. For this reason, we were not able to give any lower bounds for these variants.

- For problem G4/R1R2/C4 our reductions required that $|\mathcal{F}| \geq 4$ for the goal of separating a pair, even though the results for the corresponding problem when removing a key part applied for $|\mathcal{F}| = 2$. What can be said about G4/R1R2/C4 when $|\mathcal{F}| = 2$ or $|\mathcal{F}| = 3$?

- Can any lower bounds be proven for minimizing the depth for the full disassembly problem with a constant size family of graphs? Is the problem NP-hard?

Chapter 7

Inapproximability of Geometric Settings

7.1 Introduction

While the inapproximability results of Chapter 6 give strong evidence that minimizing the cost of assembly sequences is quite difficult, it does not prove so conclusively. The reductions which we gave in our general VAS model do not automatically apply to the original geometric assembly sequencing problems. This includes the inapproximability proofs and similarly all of the reductions relating the hardness of different variants of our problem to each other. The reason these results do not apply is that a hard instance of the general problem may not be realizable using geometric input. It is possible that, by generalizing the original problem, we may have made it much more difficult. In fact, there are many general problems in the literature which have proven significantly easier to approximate, once restricted to a geometric setting (see Section 2.3).

For these reasons, we consider the difficulty of these same assembly sequencing problems, in this chapter, when restricted to various geometric settings. We prove lower bounds against the approximability of three different complexity measures, using three different geometric settings. We begin by studying the minimum number of directions needed for any of the five goals, in a setting of three-dimensional polyhedral assemblies, when the allowable motions are either infinitesimal or infinite translations. Secondly, we consider minimizing the number of re-orientations for any of the five problem goals, when restricted to linear moves. We give a lower bound construction using a two-dimensional polygonal assembly, showing the inapproximability when restricted to removing one part at a time. Finally, for minimizing the number of steps needed in removing a key part or separating a key pair, we give our strongest inapproximability results. We prove a $2^{\log^{1-\gamma} n}$ lower bound for the approximability, using an assembly consisting entirely of unit disks in the plane, where disks are removed using translations to infinity. This result, surprisingly, matches our strongest known lower bounds for the identical problem variants in the general VAS framework.

We present the following three theorems in this chapter, with the proofs given in the following sections. The results for all goals are summarized in Table 7.1.

Theorem 32 *We consider minimizing the number of directions used (C1), for a polyhedral assembly in three-dimensions, restricted to either infinitesimal or infinite translations. In this setting, it is NP-hard to minimize the number of distinct directions for any of the goals (G1, G2, G3, G4, G5). This is valid with or without the linear restriction, R1.*

Theorem 33 *We consider minimizing the number of re-orientations used when restricted to linear steps (R1/C2), for a polyhedral assembly in two-dimensions, using either infinitesimal or infinite translations. For all five goals, we prove,*

(i) When $|\mathcal{F}| = 3$, minimizing the number of re-orientations is NP-complete.

(ii) When $|\mathcal{F}| = 4$, achieving a $(1 + c)$-approximation is NP-hard for some $c > 0$.

(iii) In general, achieving a $\log^{\delta} n$-approximation is quasi-NP-hard, for some $\delta > 0$.

Theorem 34 *We consider an assembly consisting solely of disks of unit radius, whose centers lie on a polynomial-sized grid in the plane. Our goal is to remove a key disk, and we allow disks to be removed individual by translations to infinity. For this setting, it is quasi-NP-hard to approximate the minimum number of steps within a factor of $2^{\log^{1-\gamma} n}$ for any $\gamma > 0$. This bound also applies if we consider only translations along the positive X-axis and Y-axis. Additionally, this construction generalizes to axis-aligned unit squares, and to higher dimensions.*

7.2 Graph Properties in Geometric Settings

Once restricted to a geometric setting, there should be a great deal more enthusiasm for our ability to find low cost assembly sequences in various settings. Instances of VAS drawn

Table 7.1: Inapproximability in geometric settings

	No restrictions	R1 Linear	R2 Constant Family	R1R2 Constant Family & Linear
Directions C1	3D-polygons (G1, G2, G3, G4, G5) NP-hard Theorem 32	3D-polygons (G1, G2, G3, G4, G5) NP-hard Theorem 32	solvable by computing DBG's	solvable by computing DBG's
Re-orientations C2		2D-polygons (G1, G2, G3, G4, G5) $\log^{\delta} n$-hard Theorem 33	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert - 1)$-approximable Observations 9 & 10	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert - 1)$-approximable Observations 9 & 10 2D-polygons (G1, G2, G3, G4, G5) $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $(1 + c)$-hard Theorem 33
Steps C4		DISKS setting (G2, G3, G4, G5) $2^{\log^{1-\gamma} n}$-hard Theorem 34	$2(\lvert\mathcal{F}\rvert - 1)$-approx Observation 12	DISKS setting (G2, G3, G4, G5) $\lvert\mathcal{F}\rvert \geq 2$: $2^{\log^{1-\gamma} n}$-hard Theorem 34

from a geometric setting should have a great deal more structure, of which we can take advantage. In order to have additional success for a particular setting, there are two issues which must be handled. The first is to determine what additional properties exist in the family of blocking graphs due to geometry. The following issue is how those properties can be utilized algorithmically to better approximate the optimal cost of assembly sequencing.

We consider some of the related work in computational geometry discussed in Section 2.3. For a collection of convex parts in two dimensions, Guibas and Yao show that for any given direction, there will always exist some part which can be translated to infinity in the direction without disturbing the other parts [31]. This fact has an immediate impact for assembly sequencing. For any given direction, this implies that the blocking graph will not contain any directed cycles. This is an example of identifying additional structure on blocking graphs from a geometric setting. Furthermore, this property can be readily used in finding good assembly sequences. Since each graph is acyclic, there exists a stack assembly sequence in any direction, based on Observation 7. Therefore, if we wish to minimize the number of directions for disassembling such a collection, we can be assured that a single direction is sufficient, and thus we have the optimal cost solution. Similarly, this provides the optimal solution for minimizing the number of re-orientations, and we can be sure to find the optimal number of steps required which will be either one or two, based on Observation 11.

For convex parts in three dimensions, there certainly may exist cycles in the associated blocking graphs. However, for instance, with balls in d dimensions, the result of Dawson states that at least $\min(n, d+1)$ balls can be translated individually to infinity. This fact alone, does not necessarily tell us anything for a specific blocking graph, rather it gives us a property for the family of graphs, namely that there must be at least $\min(n, d+1)$ distinct parts, which have no outgoing edges in at least one blocking graph.

Similar results on the separability of objects may provide us with additional properties, however they do not provide any immediate algorithmic results. For example, Toussaint shows that for any two star-shaped polygons in two dimensions, there must exist some direction of translations which separates the two [73]. In terms of our graphs, this tells us that there must be at least one blocking graph where the corresponding edge between two parts is missing. Whether this property is helpful for assembly sequencing is not clear. Similarly, he shows a very interesting property, namely that for a collection of general polygons in two dimensions, if a given direction does not offer a depth order, then it must be the case that there exist two polygons which are interlocked (cannot be separated from each other) in that

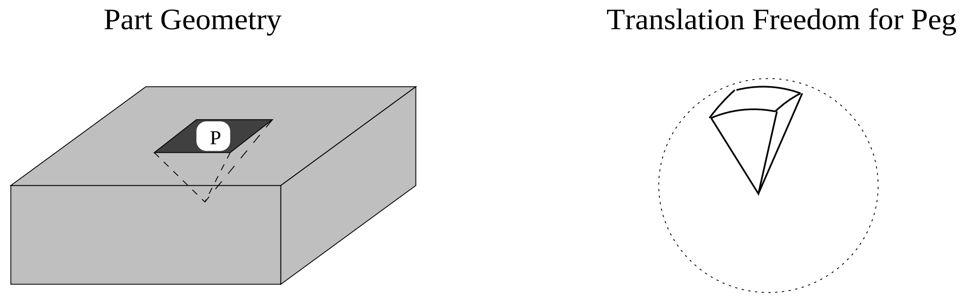


Figure 7.1: A peg in a base

direction. As a graph property, this tells us that for any cycle in our blocking graphs, there must actually exist two nodes which have a two-cycle. This too seems like a very relevant property for assembly sequencing, but whether it can be used algorithmically is not clear.

7.3 A Special Case of Set Cover

Consider the following scenario in three-dimensions. Imagine a large, flat rectangle as the base of an assembly, with the remainder of the parts as polygonal pegs which are embedded into the base. Each peg will naturally have some specific region of directions, by which it can be translated away from the base. An example of a single such peg is shown in Figure 7.1, modified from [67]. Now we consider the minimum number of directions which must be used to remove all the pegs from the base. As pointed out in [78], this instance looks very much like a Set Cover problem, in that we must choose a minimum number of directions, where each direction allows for the removal of some set of pegs. Unfortunately, it does not seem possible to realize an arbitrary instance of Set Cover in this way, so the lower bounds for Set Cover do not apply. Instead, we examine a special case of Set Cover which we call Convex Polygon Cover, which we are able to realize geometrically. We define the Convex Polygon Cover problem as follows,

Input:	A collection, $\mathcal{R}$, of (possibly degenerate) convex polygons in the plane.
Output:	A set of points $\mathcal{P}$, such that every polygon of $\mathcal{R}$ contains at least one point of $\mathcal{P}$.
Cost:	The number of points, $|\mathcal{P}|$.

A similar, even more restricted problem, Rectangle Cover is defined in [62], where all polygons are axis-aligned rectangles. It is not known whether Rectangle Cover is NP-hard.

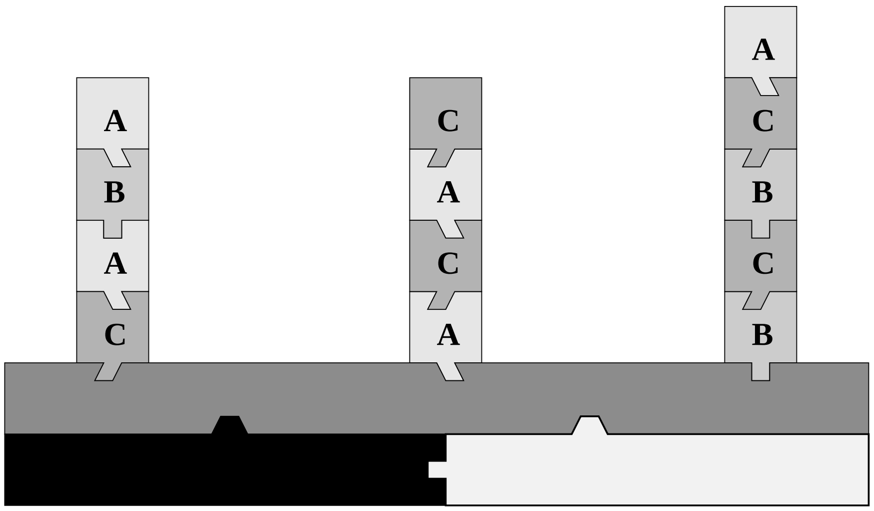
Lemma 35 *We are able to realize any instance of* Convex Polygon Cover *using a three-dimensional, polygonal assembly, where the goal is to remove a key part using as few directions of translation as possible.*

Proof: Given a set of polygons in the plane, we will consider the corresponding homogeneous coordinates to project them onto the upper hemisphere [71]. Given a single such projection, we can design a peg which can be removed from the base using exactly those directions represented by the polygon. We simply create a peg which is embedded into the base, with the shape of the polyhedral cone defining the projected polygon (for example, if the polygon were a square centered around the origin, our corresponding peg would be a four-sided pyramid embedded upside down with its tip in the base). For degenerate polygons, we may use parallel planes with an arbitrarily small separation to define our pegs. Each peg can be made arbitrarily small, and so we may lay out many such pegs in the base, far enough apart so that they will not interfere with each other's removal. This completes the construction. To remove any given peg, we must at some point use a direction of translation which lies in the corresponding polygonal region. Therefore, any assembly sequence which removes all the pegs will provide a solution to the Convex Polygon Cover problem, where the number of directions used is equal to the cost of the solution. ▮

Lemma 36 *The* Convex Polygon Cover *problem is NP-hard.*

Proof: We will base this result on a reduction from the problem of Planar Vertex Cover, which is known to be NP-complete [23]. Given an instance of Planar Vertex Cover, we can simply let each edge of the graph be represented by a degenerate polygon, and we consider this input to the Convex Polygon Cover problem. Without loss of generality, there is no need to pick any point that is not at a vertex of the graph, in covering the polygons. Therefore, there is a one-to-one correspondence between such solutions to the Convex Polygon Cover instance and solution to the Planar Vertex Cover instance. This completes the reduction, and thereby proves that Convex Polygon Cover is NP-hard. ▮

Proof of Theorem 32: For the goal of removing a keypart, this theorem is a result of Lemmata 35 & 36. Using this exact construction, the product is fully disassembled exactly when all pegs have been removed from the base, and so this proves the hardness for goal G1. For the goal of separating two parts, we can split the base into two pieces, cutting it parallel to



S = {ABAC, CACA, ACBCB}

Figure 7.2: Example construction for SCS reduction

its top surface, so that all the pegs completely penetrate the first piece, and are embedded into the second. The two parts of the base will be stuck to each other until all of the remaining pegs share a common direction for removal, and hence separating the two key parts will require the same number of directions as the original problem, proving the result for goal G4. The goals G3 & G5, are generalization of the others, and thus the hardness results also apply. ∎

7.4 Finding a Common Supersequence

Proof of Theorem 33: When constrained to using linear moves, we give a construction which reduces the problem of finding a common supersequence [23], to the problem of fully disassembling an assembly consisting of polygons in two-dimensions. A string T is a supersequence of a string S, if S can be obtained by erasing zero or more symbols of T. Given a finite set of strings over alphabet Σ, a common supersequence is a string T which is a supersequence for each string in the set. Given a set of s strings, with combined length n, over an alphabet of size $|\Sigma| = k$, we build the following instance of removing a part from an assembly. The base is a long, flat rectangle, and each sequence will be represented as a tower of "square" blocks stacked on the base, with the towers spaced sufficiently far away from each other. Each block will represent a symbol in a string, and will be attached to the piece below it with a small "peg" inserted so as to restrict the separation to a single direction of motion. Each alphabet symbol will be assigned a unique angle of separation for the associated pegs. All of the directions will be chosen to lie in a sufficiently small cone so as to prevent the individual towers from interfering with each other. A (modified) example is given in Figure 7.2.

If we assume that none of the input sequences contains consecutive occurrences of the same character, then we claim that any solution for fully disassembling the product provides us with a supersequence whose length is equal to the number of re-orientations, and vice versa. If the supersequence input does have consecutive occurrences of the same character, we can remedy this by doubling the size of the alphabet, replacing each occurrence of character a by the sequence a_1a_2.

The problem of finding the shortest common supersequence is known to be NP-hard [23], and more recently it was shown to be Max-SNP-hard, even over a binary alphabet [9]. Therefore, by doubling the alphabet as above, we get that our problem of removing a part is Max-SNP-hard, when $|\mathcal{F}| \geq 4$. Also, it was shown in [40] that there exists a constant $\delta > 0$, such that it is quasi-NP-hard to approximate the shortest common supersequence to within a factor of $log^{\delta}n$. Finally, even if strings have consecutive occurrences of the same symbol, it was shown that for an alphabet of size $|\Sigma| = 3$, that finding a common supersequence with the minimum number of *runs* is NP-complete [59]. A *run* is defined as a group of consecutive occurrences of the same symbol, and hence the number of runs is exactly equal to the number of re-orientation in our problem. For this reason, minimizing the number of re-orientations is NP-complete when $|\mathcal{F}| = 3$. This proves our theorem for the full disassembly goal.

We can extend this construction to the case of removing a key part or separating a key pair as follows. We introduce two extra parts, as shown in Figure 7.2, which are interlocked, and which are attached to the bottom of the base with pegs which span the range of angles used by the alphabet symbols. Because we are restricted to using linear operation, if we now request for the base to be removed as the key part, this will require exactly the same number of re-orientations as the original construction. Furthermore, if we request the separation of the original base from one of the two new parts, this too requires the same number of re-orientations. In this way, our reduction remains valid for all five of the possible goals. ∎

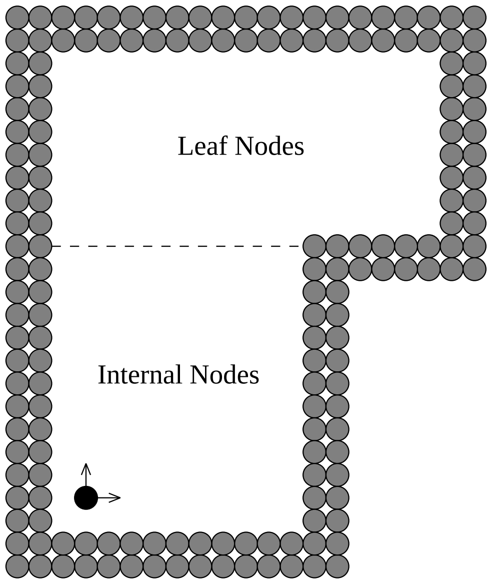


Figure 7.3: Overview of DISKS construction

7.5 The DISKS Problem

Proof of Theorem 34: Our proof is based on a reduction, with polynomial blowup, from AND/OR scheduling with internal-tree precedence constraints, and with OR-degree bounded by two. (We do not require such a bound on the AND-degree.) Given a hard instance from Corollary 13, we construct an instance of the DISKS problem. We assume, without loss of generality, that OR-nodes rely only on internal nodes.

Our scene consists entirely of disks with radius one, whose centers lie on a polynomially-sized, integer grid. We prove this result directly for the case where only two directions of translations are allowed, namely North and East. We place a wall of width $2W$ around the perimeter of our working area which we consider immovable. We will place some holes in the wall, described later, which allow a clear path out for some disks. We consider our main working area as two sections, one for the mechanisms involving the interior nodes, and the second section for the leaf node mechanisms. The overview of the construction is given in Figure 7.3.

First we describe the mechanism involving the internal nodes. Since the internal-tree defines a partial order on these nodes, we can number the internal nodes, $T_1, \ldots, T_I$ so that if an internal node depends on another internal node, it will have a higher index. For each internal node, T_i, we create a disk, D_i, centered at $(6i, 6i)$. We define the wall to the North by

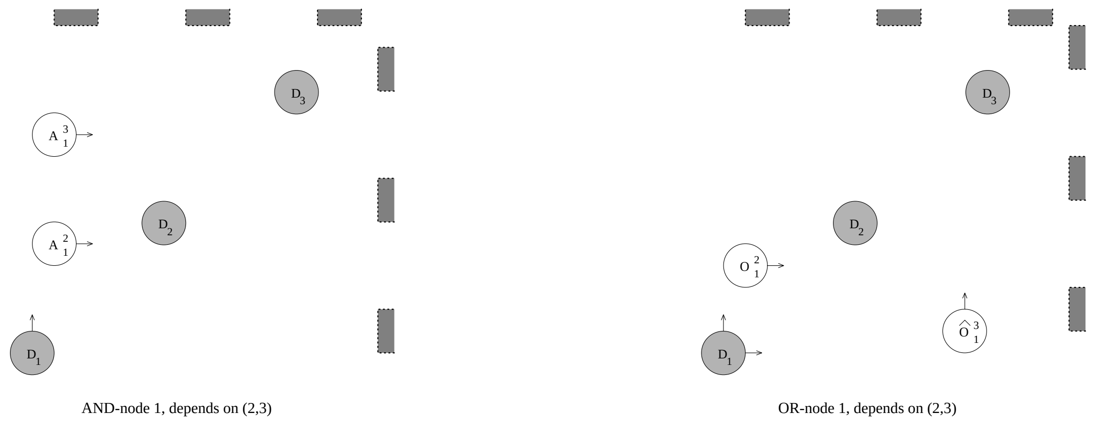


Figure 7.4: Internal node mechanisms

placing a column of W disks with x-coordinates centered at $6i+2$ for each disk D_i, assuring us that the disk itself has an "escape route" to the North. For the East wall, we place a row of disks centered at y-coordinate $6i+2$ in the case that disk D_i is an OR-disk, or else at $6i+1$ in the case that disk D_i is an AND-disk. In this way, we assure an additional escape passage to the East for an OR-disk, but not for an AND-disk.

Next, we add in additional disks to enforce the precedence constraints. For AND-node, T_i, blocked by node $T_k \in P_i$ (and thus $i < k$), we add a disk A_i^k centered at $(6i+1, 6k-1)$, which will be forced to the East by our previous placement of the walls. For an OR-node, T_i, which depends on 2 nodes, T_k and T_l, we create two new disks, O_i^k located at $(6i+1, 6k-1)$, which will be forced East by our walls, and $\hat{O}_i^l$ located at $(6l-1, 6i+1)$, which will be forced North. The entire internal node mechanisms are contained in a $(6I+1) \times (6I+1)$ square. Examples are given in Figure 7.4.

The section for the leaf mechanisms begins at height $6(I+1)$ so as to be higher than the internal mechanisms. We can number the leaf nodes in any order, and we create a separate mechanism for each leaf in a strip of height $2I$. For a given leaf, L_a, we create what we term a *blockade*, to the right of this strip. The blockade consists first of a diagonal chain of to the Northeast of height $2I$, followed by a horizontal chain of B disks to the East of the end of the first chain (where B is determined later). The disk beginning the blockade is centered at $(6(I+1), 6(I+1)+Ia)$. The wall to the East of the blockade is removed, allowing the disks of the blockade an escape. For any disk located in the horizontal strip associated with L_a, escaping to the East will require an additional cost of at least B to break through the blockade. However this cost is only charged once per blockade, after which any disks in the

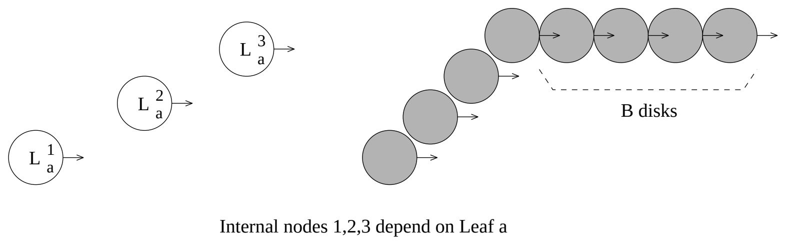


Figure 7.5: Leaf node mechanism

horizontal strip may escape. Now, for every internal node T_i which depends on leaf L_a, we create a disk L_a^i, located at $(6i+1, 6(I+1)+Ia+2i)$, which is forced East by the walls. Figure 7.5 shows an example of a leaf mechanism.

To complete the construction, we set the blockade value, $B = 4I(L+I)$, to be greater than the total number of disks in the remainder of the internal and leaf mechanisms combined. In this way, the number of blockades removed dominates any additive costs in the rest of the construction. Finally, we assign $W = B(L+1)$, so that the cost of removing all non-wall disks is less than the cost of digging a single new hole through any part of the wall. For this reason, we may assume without loss of generality that any solution to this DISKS instance has cost at most W. Finally, we note that the wall has perimeter which is $O(BL)$, and hence the total number of disks in our construction is polynomially bounded. An example of the final construction is given in Figure 7.6.

It is not hard to verify that for this DISKS instance, a solution for removing the root disk with cost at most kB can be translated to an AND/OR solution of cost at most k. Similarly, an AND/OR solution of cost k can be translated to a DISKS solution with cost less than $(k+1)B$. Therefore, approximating the DISKS problem to within a factor of $2^{\log^{1-\gamma} n}$ for any $\gamma > 0$ is quasi-NP-hard, as the additive error and the polynomial increase of the input size disappear by adjusting γ.

Our proof shows the hardness of the DISKS problem when translations are limited to the North and East. If we allow translations in arbitrary directions, the theorem holds using this same construction. Furthermore, even if we are not restricted to linear moves, we could prove the same lower bound for minimizing the number of disks removed.

It is also easy to see that the disks can be replaced by axis-aligned, 2×2 squares and the construction still holds. For higher dimensions, the walls can be extended to block any useful motions in other dimensions, while still using polynomially many disks. ▮

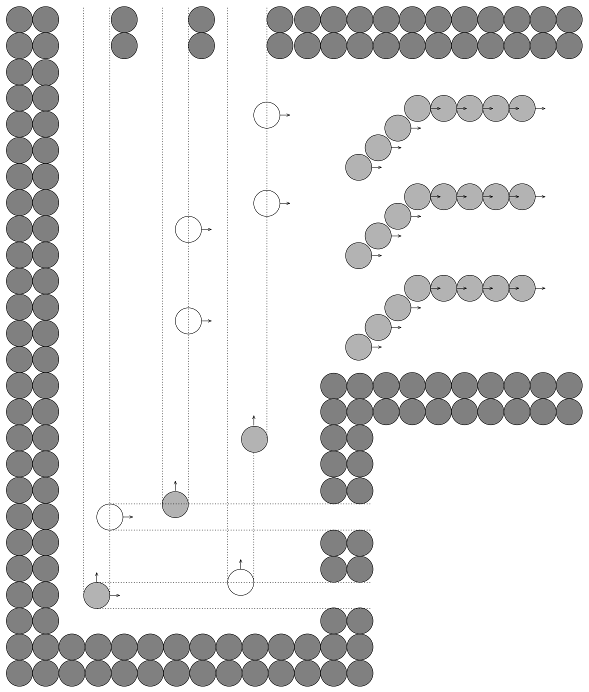


Figure 7.6: A complete DISKS construction

7.6 Open Problems

The overwhelming open problem is to better understand the approximability of various cost measures in different geometric settings. Although we have given several lower bounds for geometric settings, in many cases these bounds are far weaker than the corresponding bounds for VAS. However, even despite this, we have not been able to give any new non-trivial upper bounds which take advantage of the geometric settings, and so the gaps between the upper and lower bounds are enormous. Besides those discussed in this chapter, there are many other interesting geometric settings to study. We leave these studies for future research.

Finally, we list several open questions which are more closely related to the settings of this chapter.

- Can the NP-completeness of CONVEX POLYGON COVER be strengthened to achieve stronger inapproximability results for the minimum number of directions?
- In regard to the minimum number of re-orientations, can the two-dimensional construction from SHORTEST COMMON SUPERSEQUENCE be modified to handle the case

without the linear restriction? (As is, without the restriction, one would be able to remove the towers from the base without taking them apart).

- If three dimensions are used, can the construction from SHORTEST COMMON SUPERSEQUENCE be modified to handle the case without the linear restriction?
- Is it possible to develop an n^{ϵ} approximation algorithm for removing an axis-aligned unit disk with linear operations, while minimizing the number of steps? What about if restricted to positive X and Y axes? What if we consider unit squares instead?
- Does there exists a collection of unit-squares which requires $\omega(\sqrt{n})$ steps to remove a given square, when all directions of infinite translations are allowed?

Chapter 8

Conclusions and Open Problems

We explain the lack of progress in finding optimal or near-optimal assembly sequences by formally proving the inapproximability for minimizing the cost of an assembly sequence for a variety of desired cost measures. We look at several variants of the problem based on either full or partial (dis)assembly, and we classify the approximability of the problems based on the desired cost measure and additional restrictions placed on the allowed sequences.

For a graph-theoretic generalization of these problems, we show that achieving an approximate solution within a factor of $2^{\log^{1-\gamma} n}$ of optimal, for any $\gamma > 0$, is difficult for many of the cost measures we consider. As a special case, we prove similar hardness results for the problem of scheduling with AND/OR precedence constraints. Finally, as our graph-theoretic problem is a generalization, we prove hardness results for several complexity measures in simple geometric settings. For minimizing the number of parts which must be removed to access a key part, we match our strongest inapproximability results, even for a setting consisting entirely of unit disks in the plane, while using simple translations to infinity to remove parts. For minimizing the number of directions used or the number of re-orientations, our geometric lower bounds are far weaker than their graph-theoretic counterparts.

Our hope is that our work can be used to better understand the source of the difficulties, possibly leading the way to successful approximation algorithms, or else in redirecting future efforts into identify other structure or properties of industrial assembly sequencing instances which would allow for better approximations.

The overwhelming open problem which remains is to develop non-trivial approximation algorithms for any of the settings which we study. The importance of our graph-theoretic model is that it captures techniques that are currently used for finding feasible sequences for a great deal of geometric settings. Achieving any positive results in this model would immediately apply to all of these geometric settings. Although our lower bounds show that success in this model is limited, achieving something such as a $\sqrt{n}$-approximation would still be of great practical value. Automated assembly sequencers are beginning to have more impact in industrial use, and for a manufacturer, it is of no comfort to simply say that a problem is difficult. The product is going to have to be manufactured one way or another, and so any improvement to the cost is quite valuable.

Alternatively, it may be the case that by studying different geometric settings individually, much better approximations can be achieved by taking advantage of additional structure in the problem. Although we have shown that in some cases, the geometric problem is indeed quite hard, many of our geometric lower bounds are far weaker than the general bounds. These geometric problems are the true motivation for this work and so future research should either provide approximation algorithms for these settings, or else improve the geometric lower bounds to justify the lack of progress. Specific open questions, closely related to our work, have been listed at the end of relevant chapters, in Sections 5.5, 6.6 and 7.6.

Finally, there are many additional variations which can be added to our original model of virtual assembly sequencing, defined in Chapter 3. For example, although we tried to compile a relatively complete list of cost measures which have been considered by previous research, there are many other natural measures which may be interesting in practice. For example many of our cost measures can be re-defined as weighted version. For instance, rather than charge a sequence for the number of steps, it may be useful to assign weights to the different members of the family of motions, so that more complicated motion types can be allowed, but with higher cost.

Other restrictions can also be added to our graph-theoretic model quite naturally. One interesting constraint that has been considered quite often in assembly sequencing is to require that all subassemblies which are used during the construction of a product are connected subassemblies [32, 47, 74]. Generally, unconnected subassemblies require additional expensive fixturing, and are more troublesome in terms of stability and maneuverability. The additional connectedness restriction can easily be modeled as a part of VAS. We include one additional, undirected graph, the *connection graph* G_C, which has a node for each part in the assembly, and a directed edge between two parts if they are in contact in the final assembly. With this additional restriction, even the question of finding a feasible assembly sequence is an issue, as the standard algorithm is no longer correct. For finding a feasible sequence for the original problem, Observation 2 assured us that the removal of a part could not possibly disallow a future possible operation and thus any valid partitioning could be used with the subassemblies handled recursively, however this observation is not valid anymore with connectedness required. As a special case, when all graphs in the family $\mathcal{F}$ are subsets of the connection graph, finding a feasible assembly sequence can be done in polynomial time, however it remains an open question for the general case, whether a feasible sequence can be find in polynomial time with this connectedness requirement [32]. Other additional restrictions, common to assembly planning, are surveyed in [42, 43], where they show that several of the constraints can be naturally integrated into the NDBG approach.

Appendix A

Tables of Experimental Results

These tables contain the results of the experiments described in Section 4.5.

A.1 Number of Steps for Removing a Part with Linear Restriction

bell9-linear (9 parts, 5 graphs)

Part	lb1	lb2	lb3	lb4	OPT-lin	rbest	ravg
1	2	2	2	2	2	2	3.4
4	3	3	4	4	4	4	6.9
6	2	2	2	2	2	2	4.9
TIME	10	203	3440	14780	303		23

bell17-linear (17 parts, 5 graphs)

Part	lb1	lb2	lb3	lb4	OPT-lin	rbest	ravg
1	2	2	2	2	2	2	5.2
6	2	2	2	2	2	2	7.4
7	2	2	2	2	2	2	7.5
9	2	3	3	3	3	3	9.9
10	3	3	4	4	4	4	11.7
11	2	2	2	2	2	2	5.3
14	2	2	2	2	2	2	7.9
TIME	10	193	3716	13849	1816		36

bell22-linear (22 parts, 5 graphs)

Part	lb1	lb2	lb3	lb4	OPT-lin	rbest	ravg
1	2	2	2	2	2	2	5.4
6	2	2	2	2	2	2	7.9
7	2	2	2	2	2	2	8.2
9	2	3	3	3	3	3	11.4
10	3	3	4	4	4	4	15.7
11	2	2	2	2	2	2	5.7
14	2	2	2	2	2	2	9.2
16	3	3	4	5	5	7	17.7
17	2	2	2	2	2	2	8.0
19	2	2	2	2	2	2	8.2
20	2	2	3	3	3	3	14.2
21	2	2	2	2	2	2	8.4
TIME	10	182	3907	15853	6447		46

eng12-linear (12 parts, 5 graphs)

Part	lb1	lb2	lb3	lb4	OPT-lin	rbest	ravg
0	3	3	4	4	4	4	7.8
2	2	2	3	3	3	3	7.0
3	2	2	2	2	2	2	4.8
4	2	3	3	3	3	3	7.6
5	2	3	3	3	3	3	7.6
7	2	2	2	2	2	2	4.2
8	2	2	2	2	2	2	3.2
TIME	10	201	4164	14863	536		27

eng23-linear (23 parts, 12 graphs)

Part	lb1	lb2	lb3	lb4	OPT-lin	rbest	ravg
0	2	2	3	3	3	3	14.0
1	2	2	2	2	2	2	8.9
2	2	3	3	3	3	3	14.0
3	2	3	3	3	3	3	14.0
5	3	3	5	5	5	7	18.2
7	2	2	2	2	2	2	7.8
8	2	2	2	2	2	2	5.9
10	4	5	6	6	7	9	19.1
11	3	4	4	4	4	4	15.8
12	2	2	2	2	2	2	9.1
15	2	2	2	2	2	2	9.5
TIME	24	1545	103669	585807	196269		139

eng30-linear (30 parts, 13 graphs)

Part	lb1	lb2	lb3	lb4	OPT-lin	rbest	ravg
0	4	4	5	5	5	6	21.2
1	2	2	2	2	2	2	11.1
2	2	3	3	3	3	3	18.2
3	2	3	3	3	3	3	18.2
5	3	5	7	7	7	14	25.4
7	2	2	2	2	2	2	10.0
8	2	2	2	2	2	2	7.3
10	6	7	9	9	10	17	25.6
11	3	4	4	4	4	4	20.6
12	2	2	2	2	2	2	11.8
15	2	2	2	2	2	2	12.0
17	2	2	2	2	2	2	12.0
18	2	2	2	2	2	2	11.7
25	2	2	2	2	2	2	9.3
TIME	26	2009	164994	1004465	234225		191

eng42-linear (42 parts, 13 graphs)

Part	lb1	lb2	lb3	lb4	OPT-lin	rbest	ravg
0	8	8	9	9	9	17	35.3
1	2	4	6	6	6	9	32.2
2	2	3	4	5	7	16	35.1
3	2	3	4	5	7	15	35.2
5	4	5	5	5	5	6	26.9
14	6	10	13	15	-	31	38.3
16	2	2	2	2	2	2	14.5
17	2	2	2	2	2	2	10.6
19	6	10	11	12	-	36	36.5
22	3	4	4	4	4	7	27.6
23	2	2	2	2	2	2	16.9
26	2	2	2	2	2	2	16.9
28	2	2	2	2	2	2	16.5
29	2	2	2	2	2	2	16.7
36	2	2	2	2	2	2	13.4
41	2	6	10	10	10	25	37.9
TIME	26	2865	271154	2652067	–		324

sno29-linear (29 parts, 1250 graphs)

Part	lb1	lb2	lb3	lb4	OPT-lin	rbest	ravg
0	3	4			5	6	11.7
2	3	4			9	10	16.4
3	2	4			9	11	17.9
6	3	4			9	10	17.5
7	3	5			10	11	17.4
8	2	3			6	7	13.7
9	2	3			8	11	18.2
10	3	5			8	9	16.8
11	2	3			6	6	12.8
12	2	4			7	7	16.7
13	2	3			4	4	8.4
14	3	4			4	4	6.7
15	3	4			7	9	13.5
16	2	3			3	3	5.8
17	2	2			2	2	3.5
18	4	4			6	6	12.0
19	3	4			8	10	17.4
20	2	3			3	3	5.8
21	3	4			9	11	18.1
22	2	2			2	2	3.6
23	3	4			7	7	15.1
24	3	4			8	10	17.7
25	3	4			6	6	12.4
26	2	3			4	4	8.3
27	3	4			5	5	8.7
TIME	2500	29004200			207372		14999

A.2 Number of Re-orientations for Removing a Part with No Restrictions

bell9 (9 parts, 5 graphs)

Part	OPT	size1	size2	edges1	edges2	rbest	rand1
4	3	3	3	3	3	3	3
6	2	3	2	3	2	2	2.6
TIME	20.0	15.0	17.5	15.0	17.5		13.9

bell17 (17 parts, 5 graphs)

Part	OPT	size1	size2	edges1	edges2	rbest	rand1
10	3	3	3	3	3	3	3
14	2	3	2	3	2	2	2.6
TIME	20.0	15.0	17.5	15.0	17.5		13.9

bell22 (22 parts, 5 graphs)

Part	OPT	size1	size2	edges1	edges2	rbest	rand1
10	3	3	3	3	3	3	3
14	2	2	2	2	2	2	2.4
16	2	2	2	2	2	2	2
17	2	2	2	2	2	2	2
TIME	15.0	11.3	13.8	11.3	13.8		11.8

eng12 (12 parts, 5 graphs)

Part	OPT	size1	size2	edges1	edges2	rbest	rand1
0	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
4	2	3	2	2	2	2	2.8
5	2	3	2	2	2	2	2.8
TIME	15.0	12.5	15.0	12.5	15.0		12.0

eng23 (23 parts, 12 graphs)

Part	OPT	size1	size2	edges1	edges2	rbest	rand1
0	2	2	2	2	2	2	2
2	2	3	2	3	2	2	2.9
3	2	3	2	3	2	2	2.9
5	2	2	2	2	2	2	2
10	3	4	4	4	4	3	3.8
11	2	4	2	4	2	2	3.7
15	2	2	2	2	2	2	2
TIME	44.6	34.3	42.9	34.3	42.9		33.2

eng30 (30 parts, 13 graphs)

Part	OPT	size1	size2	edges1	edges2	rbest	rand1
0	3	3	3	3	3	3	3
2	2	3	2	3	2	2	3.3
3	2	3	2	3	2	2	3.3
5	2	2	2	2	2	2	2.4
10	3	4	4	4	4	3	3.8
11	2	4	2	4	2	2	3.7
18	2	2	2	2	2	2	2
25	2	2	2	2	2	2	2
TIME	58.5	39.0	53.6	37.4	53.6		38.2

eng42 (42 parts, 13 graphs)

Part	OPT	size1	size2	edges1	edges2	rbest	rand1
0	3	3	3	3	3	3	3
2	2	3	2	3	2	2	3.1
3	2	3	2	3	2	2	3.1
14	3	3	3	3	3	3	3.1
19	3	5	4	5	4	3	4.5
22	2	4	2	4	2	2	3.9
29	2	2	2	2	2	2	2
36	2	2	2	2	2	2	2
TIME	63.4	40.6	55.3	40.6	55.3		40.1

sno29 (29 parts, 1250 graphs)

Part	OPT	size1	size2	edges1	edges2	rbest	rand1
0	5	9	10	13	8	5	10.7
2	9	13	13	14	11	9	13.3
3	9	11	12	12	10	10	14.0
6	9	11	12	14	12	10	13.8
7	9	14	13	14	12	10	14.0
8	6	11	10	10	10	7	12.1
9	8	11	14	12	9	9	13.5
10	7	14	14	14	10	7	13.7
11	6	11	9	9	8	6	11.3
12	7	10	14	14	10	8	13.3
13	4	7	6	5	6	4	7.8
14	4	13	4	7	4	4	6.0
15	7	12	12	13	10	8	11.7
16	3	4	4	4	4	3	5.3
17	2	3	2	3	2	2	3.3
18	6	13	10	13	8	6	10.9
19	8	11	14	13	9	9	13.6
20	3	4	4	4	4	3	5.2
21	9	11	12	13	11	10	14.3
22	2	3	2	3	2	2	3.1
23	7	10	10	10	10	7	12.8
24	8	12	12	11	12	10	14.3
25	6	9	10	13	9	6	11.1
26	4	7	6	5	6	4	7.7
27	5	10	6	7	6	5	8.2
TIME	116900	12200	26700	12500	25700		13255.5

Appendix B

Questioning the Claims of [20, 79]

In a sequence of two journal articles, Woo and Dutta present an algorithm for the removal of a part (or multiple parts) from a product made of polyhedral parts in either two or three dimensions [20, 79]. They allow for parts to be removed, one at a time, using translations to infinity. In their work, they claim that for products which have what they term the "total ordering" property, their algorithm is guaranteed to produce the minimal cost assembly sequence, when trying to minimize either the number of steps required or minimize the number of *adjacent* parts removed. For these totally ordered products, this claim ammounts to solving the (G2/C4/R1) variant for which we proved a $2^{\log^{1-\gamma} n}$ lower bound, even in the DISKS setting.

This potential contradiction leads us to reconsider the claims of optimality made in [20, 79], and the assemblies to which the algorithm applies. Unfortunately, we feel that their presentation suffers from being inconsistent and vague, and thus their claims are in question. For starters, the description of their algorithm, the definition of the totally ordered properties, and their claims for optimality are not well specified. Furthermore, in the explanation of these issues, there are significant inconsistencies between the two versions of their papers. We will begin by addressing their definitions and claims which we feel are not sufficiently clear. Following this, we will give an example of a product very similar to an example they give, for which their optimality claim does not hold.

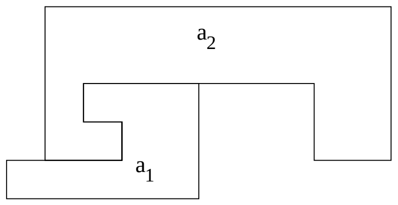


Figure B.1: Freedom for Infinitesimal Translational

B.1 Cost Measure

The first issue for us is to understand what cost measure they claim to be optimizing. In their first paper in 1991, they claim in Section 3.1 that "the DT thus constructed yields a minimum number of removals (motions) in order to access a certain component" [79]. In the 1995 paper, they claim to remove a part while minimizing the number of *adjacent* parts which are removed. In fact, in this second paper, they claim to be able to solve this task, even when asked to remove an arbitrary *set* of parts. In any event, our example in Section B.4, will serve as a counterexample to the optimality of both of these cost measures.

B.2 Motion Class

A second issue is understanding what motion class they allow in their setting. They are very clear in that they consider strictly translational, one-step motions. In terms of whether they wish to consider infinitesimal translations versus translations to infinity, they explicitly express on the first page of [79] that there are several criteria for "clearing" a part from a subassembly, and that they choose to adopt the criterion which requires that "the component can be translated to infinity." However, the subroutine which they present for checking for the removability of a given part clearly checks only for infinitesimal motion. The algorithm `Disassemblable` which they present for computing the freedom of a part from a subassembly is based entirely from examining the contact faces which the part in question shares with adjacent parts [79]. As an example, if their algorithm were run on the two part assembly which they provide in figure 1.3(a) of [79], (our Figure B.1), it would determine that part a_1 can be successfully removed along the positive X-axis, when clearly this is not possible if the part must be cleared to infinity.

Based on their explicit description of the check for removability, we seem to have no choice but to assume that they are considering only the local freedom for infinitesimal translations.

This decision is also consistent with the fact that in their algorithm for disassembly, when trying to find the effects of the removal of one part on the removability of other parts, they only recheck the removal of those parts which are adjacent to the first part removed.

B.3 "Total Ordering"

The real issue in question is understanding the *total ordering* property which they define for a product. Their claims are simply that their algorithms are guaranteed to find the optimal sequence for products which exhibit this special property, and so in order to refute such claims, we must understand this property.

Originally, they define this property in Definition 1.3 of [79] as follows:

> "An assembly is *partially ordered* if the disassembly of a component requires an immediate prior disassembly of k components. If $k < 2$, the assembly is *totally ordered*."

The concept is defined in a similar manner in [20], although the word *immediate* is no longer present when they define total ordering in Section 2.1 as:

> "A sequential assembly $\mathcal{S}$ is *partially ordered* if the removal of a component in the assembly requires the prior disassembly of k (> 1) components. Otherwise, $\mathcal{S}$ is *totally ordered*."

These definitions now hinge on the concept of a part's removal "requiring the prior disassembly" of k other parts. Unfortunately, this concept is never formalized in either paper, although it is used repeatedly. Consider the statement that the removal of some part, call it a requires the prior removal of k parts. We imagine two logical interpretations we may give to this phrase. The first interpretation is that no matter how one tries to remove part a, any valid sequence will always require at least k other parts to be removed before a (however, the set of blocking parts may be different for different possible solutions). The second interpretation is that there are k *specific* parts, which absolutely must be removed before a can be removed, in any possible disassembly sequence.

If we take the first interpretation, simply that any valid sequence must first remove some number of parts, then the notion of a totally ordered product is quite trivial, as it would mean that every part of the product can be removed in either one or two steps. If this is indeed the intended concept of total ordering, then their claims of optimality are quite trivial. If every part requires either one or two steps for removal, it is quite simple to identify those

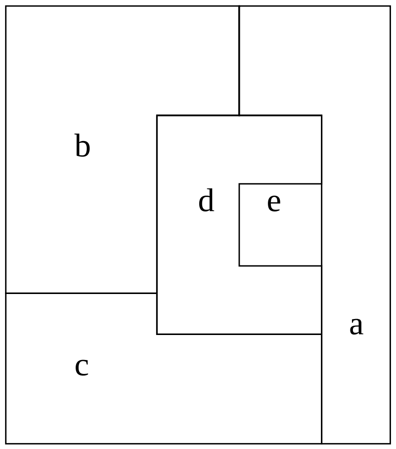


Figure B.2: An example given by Woo and Dutta

parts which can be removed in one step from the full assembly. For all other parts, it must be that the removal of some single other part must create the freedom for removing this part. In fact, at one point, it seems that maybe this is the interpretation which they intend, as they describe, in Section 3.3 of [20], a scenario in which "...the removal of each component is possible after the removal of at most one other component and the assembly is totally ordered. Clearly, then the height of DT is ≤ 2 ..." However, in this case, they still claim to be able to produce the optimal sequence for removing a set of multiple parts minimally, and it cannot possibly be accomplished to remove a large set of components, one at a time, using at most two steps overall. Also, this interpretation would be somewhat surprising, given that the examples they give of their algorithm in action (Figure 3.3 of [79] and Figure 4 of [20]) both result in trees with heights greater than two.

If we take the second interpretation, namely that a parts removal may rely on the prior removal of some *specific* other part, this seems a more natural view for the concepts of defining a partial order or total order. Unfortunately, if this is the interpretation, it is not clear at a glance how to tell whether a given product has this required property or not. Also, in this respect, their use of the phrase is quite puzzling, as they give an example in [20], of an assembly which we show in Figure B.2. For this assembly, they consider removing the multiple set of parts $\{d, e\}$, and they make the following statement in the fifth paragraph of Section 3.2: "By inspection, we can verify that it is necessary and sufficient to remove component a for disassembling component d and e." It is clear that the removal of a is certainly sufficient for allowing the removal of parts d and e, however it is not at all necessary. One could remove

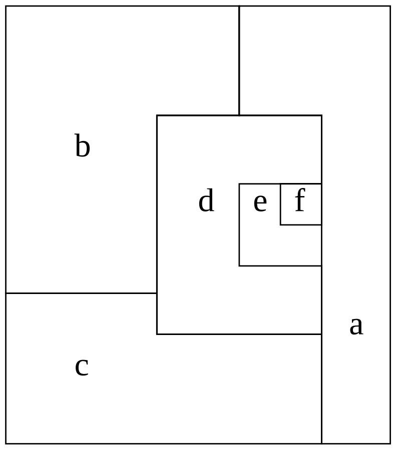


Figure B.3: Our counterexample for optimal disassembly

parts in the following order, b, c, d, e, and thus remove both d and e without removing a.

B.4 Counterexample

In [20], they consider the product which we showed in Figure B.2, and they walk through the execution of their algorithm on this example, and show the resulting sequences which are optimal. Here, we make a minor modification to the identical product, breaking the original part e into two parts e and f. Our new product is given in Figure B.3. If we now use the same reasoning to walk through the execution of the algorithm on this new input, we get the result of the DT pictured in Figure B.4. The algorithm would first determine that parts a, b, or c may be removed at the top most level. The removal of part b or part c alone, does not allow for any additional progress to be made. However, the removal of part a allows for the new removal of part f, which in turn will now allow for the removal of part e, and this in turn frees part d.

If we are interested in the removal of part d, this algorithm results in the sequence of removals, $a \rightarrow f \rightarrow e \rightarrow d$. From their claims, it seems that we should be able to conclude that this sequence is optimal for the removal of d, but clearly this is not the case. If our cost measure is the total number of prior removal to access d, then this sequence has cost 3, and similarly, if the cost measure is the number of *adjacent* parts removed, we find that the cost is still equal to 3, as all parts in this assembly are adjacent to d. Clearly, for either of these cost measures, the optimal sequence is first to remove b, followed by c, and then d, requiring

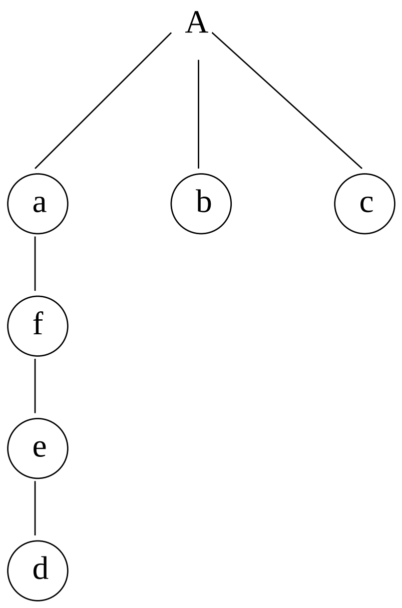


Figure B.4: The resulting DT for our counterexample

cost 2 by either metric.

Bibliography

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

[2] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. 37th Symp. on Found. Comput. Sci.*, pages 1–11, 1996.

[3] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes and linear equations. In *Proc. 34th Symp. on Found. Comput. Sci.*, pages 724–733, 1993.

[4] S. Arora and C. Lund. Hardness of approximations. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, MA, 1996.

[5] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proc. 33rd Symp. on Found. Comput. Sci.*, pages 13–22, 1992.

[6] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[7] D. Baldwin. Algorithmic methods and software tools for the generation of mechanical assembly sequences. M.Sc. thesis, MIT, Cambridge, MA, 1990.

[8] R. Bhatia, S. Khuller, and J. Naor. The loading time scheduling problem. In *Proc. 36th Symp. on Found. Comput. Sci.*, pages 72–81, 1995.

[9] P. Bonizzoni, M. Duella, and G. Mauri. Approximation complexity of longest common subsequence and shortest common supersequence over fixed alphabet. Technical Report 117/94, Universita degli Studi di Milano, 1994.

[10] G. Boothroyd. *Assembly Automation and Product Design.* Marcel Dekker, Inc., New York, NY, 1991.

[11] G. Boothroyd, P. Dewhurst, and W. Knight. *Product Design for Manufacture and Assembly.* Marcel Dekker, Inc., New York, NY, 1994.

[12] S. Caselli and F. Zanichelli. On assembly sequence planning using petri nets. In *Proc IEEE Int. Symp. on Assembly and Task Planning*, pages 239–244, 1995.

[13] S. Chakrabarty and J. Wolter. A hierarchical approach to assembly planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 258–263, 1994.

[14] B. Chazelle, H. Edelsbrunner, L. Guibas, R. Pollack, R. Seidel, M. Sharir, and J. Snoeyink. Counting and cutting cycles of lines and rods in space. *Computational Geometry: Theory and Applications*, 1(6):305–323, 1992.

[15] P. Crescenzi and V. Kahn. A compendium of NP optimization problems. Technical Report SI/RR-95/02, Dipartimento di Scienceze dell'Informazione. Università di Roma "La Sapienza", 1995.

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

[17] M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal.* Springer-Verlag, Berlin, Germany, 1993.

[18] M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. *SIAM J. Comput.*, 23(2):432–446, 1994.

[19] F. Dehne and J.-R. Sack. Translation separability of polygons. *Visual Computer*, 3(4):227–235, 1987.

[20] D. Dutta and T. Woo. Algorithm for multiple disassembly and parallel assemblies. *J. Engineering for Industry*, 117(1):102–109, 1995.

[21] T. De Fazio and D. Whitney. Simplified generation of all mechanical assembly sequences. *IEEE Trans. on Robotics and Automation*, 3(6):640–658, 1987.

[22] U. Feige. A threshold of $\ln n$ for approximating set cover. In *Proc. 28th ACM Symp. Theory Comput.*, pages 314–318, 1996.

[23] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[24] D. Gillies. *Algorithms to Schedule Tasks with AND/OR Precedence Constraints.* Ph.D. thesis, University of Illinois, Urbana, IL, 1993.

[25] D. Gillies and J. Liu. Scheduling tasks with AND/OR precedence constraints. *SIAM J. Comput.*, 24(4):797–810, 1995.

[26] M. Goldwasser, J.-C. Latombe, and R. Motwani. Complexity measures for assembly sequences. In *Proc IEEE Int. Conf. on Robotics and Automation*, pages 1581–1587, 1996.

[27] M. Goldwasser and R. Motwani. Intractability of assembly sequencing: Unit disks in the plane. In *Proc. of the Workshop on Algorithms and Data Structures*, page To appear, 1997.

[28] S. Gottschlich, C. Ramos, and D. Lyons. Assembly and task planning: A taxonomy. *IEEE Robotics and Automation Magazine*, 1(3):4–12, 1994.

[29] R. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:416–426, 1969.

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

[31] L. Guibas and F. Yao. On translating a set of rectangles. In F. Preparata, editor, *Computational Geometry*, Advances in Computing Research, pages 61–77. JAI Press Inc., 1983.

[32] D. Halperin, J.-C. Latombe, and R. Wilson. A general framework for assembly planning: The path space approach. *Manuscript*, 1997.

[33] D. Halperin and R. Wilson. Assembly partitioning along simple paths: the case of multiple translations. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1585–1592, 1995.

[34] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proc. 37th Symp. on Found. Comput. Sci.*, pages 627–636, 1996.

[35] R. Hoffman. A common sense approach to assembly sequence planning. In *Computer-Aided Mechanical Assembly Planning*, pages 289–314. Kluwer Academic Publishers, Boston, 1991.

[36] L. Homem de Mello and A. Sanderson. AND/OR graph representation of assembly plans. *IEEE Trans. on Robotics and Automation*, 6(2):188–199, 1990.

[37] L. Homem de Mello and A. Sanderson. *Computer-Aided Mechanical Assembly Planning.* Kluwer Academic Publishers, Boston, 1991.

[38] L. Homem de Mello and A. Sanderson. A correct and complete algorithms for the generation of mechanical assembly sequences. *IEEE Trans. on Robotics and Automation*, 7(2):228–240, 1991.

[39] J. Hopcroft, J. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: P-space hardness of the "Warehouseman's Problem". *Int. J. Robotics Research*, 3(4):76–88, 1984.

[40] T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. In *Automata, Languages and Programming (Proc. 21st ICALP)*, volume 820 of *Lecture Notes in Computer Science*, pages 191–202. Springer-Verlag, 1994.

[41] D. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Systems Sci.*, 9:256–278, 1974.

[42] R. Jones and R. Wilson. A survey of constraints in automated assembly planning. In *Proc IEEE Int. Conf. on Robotics and Automation*, pages 1525–1532, 1996.

[43] R. Jones, R. Wilson, and T. Calton. Constraint-based interactive assembly planning. In *Proc IEEE Int. Conf. on Robotics and Automation*, page To appear, 1997.

[44] V. Kann. Polynomially bounded minimization problems which are hard to approximate. In *Automata, Languages and Programming (Proc. 20th ICALP)*, volume 700 of *Lecture Notes in Computer Science*, pages 52–63. Springer-Verlag, 1993.

[45] D. Karger, R. Motwani, and G. Ramkumar. On approximating the longest path in a graph. In *Proc. of the Workshop on Algorithms and Data Structures*, volume 709 of *Lecture Notes in Computer Science*, pages 421–432, 1993.

[46] S. Kaufman, R. Wilson, R. Jones, T. Calton, and A. Ames. The Archimedes 2 mechanical assembly planning system. In *Proc IEEE Int. Conf. on Robotics and Automation*, pages 3361–3368, 1996.

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

[48] L. Kavraki, J.-C. Latombe, and R. Wilson. Complexity of partitioning an assembly. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 12–17, Waterloo, Canada, 1993.

[49] S. Khanna and R. Motwani. Towards a syntactic characterization of PTAS. In *Proc. 28th ACM Symp. Theory Comput.*, pages 329–337, 1996.

[50] S. Khanna, M. Sudan, and L. Trevisan. Constraint satisfaction: The approximability of minimization problems. In *Proc. 12th IEEE Comput. Complexity Conference*, 1997.

[51] M. Klawe, W Paul, N. Pippenger, and M. Yannakakis. On monotone formulae with restricted depth. In *Proc. 16th ACM Symp. Theory Comp.*, pages 539–550, 1984.

[52] S. Krishnan and A. Sanderson. Path planning algorithms for assembly sequence planning. In *Proc. Int. Symp. on Intelligent Robotics*, pages 428–439, 1991.

[53] E. Lawler, J. Lenstra, H. Rinnooy Kan, and D. Shmoys. Sequencing and scheduling: algorithms and complexity. In *Handbooks in Operations Research and Management Science, Vol. 4*, pages 445–522. North-Holland, 1993.

[54] K. Lee and R. Gadh. Computer aided design for disassembly: a destructive approach. In *Proc. IEEE Int. Sym. on Electronics and the Environment*, pages 173–178, 1996.

[55] S. Lee. Backward assembly planning with assembly cost analysis. In *Proc IEEE Int. Conf. on Robotics and Automation*, pages 2382–2391, 1992.

[56] S. Lee and Y. Shin. Assembly planning based on geometric reasoning. *Computers and Graphics*, 14(2):237–250, 1990.

[57] A. Lowe and S. Niku. Methodology for design for disassembly. In *ASME Publication #DE*, volume 81, pages 47–53, 1995.

[58] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[59] M. Middendorf. Supersequences, runs, and CD grammar systems. In J. Dassow and A. Kelemenova, editors, *Developments in Theoretical Computer Science*, volume 6 of *Topics in Computer Science*, pages 101–114. 1994.

[60] J. Millner, S. Graves, and D. Whitney. Using simulated annealing to select least-cost assembly sequences. In *Proc IEEE Int. Conf. on Robotics and Automation*, pages 2058–2063, 1994.

[61] H. Moradi, K. Goldberg, S. Lee, and R. Wilson. Geometry-based part grouping for assembly planning. *Manuscript*, 1997.

[62] R. Motwani. Approximation algorithms. Stanford Technical Report STAN-CS-92-1435, 1992.

[63] B. Natarajan. On planning assemblies. In *Proc. 4th ACM Symp. on Computational Geometry*, pages 299–308, 1988.

[64] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Systems Sci.*, 43(3):425–440, 1991.

[65] C. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[66] R. Raz and A. Wigderson. Monotone circuits for matching require linear depth. *J. ACM*, 39(3):736–744, 1992.

[67] B. Romney, C. Godard, M. Goldwasser, and G. Ramkumar. An efficient system for geometric assembly sequence generation and evaluation. In *Proc. ASME Int. Computers in Engineering Conference*, pages 699–712, 1995.

[68] N. Shyamsundar and R. Gadh. Selective disassembly of virtual prototypes. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, volume 4, pages 3159–3164, 1996.

[69] N. Shyamsundar and R. Gadh. Geometric abstractions to support disassembly evaluation. In *Proc. 23rd Design Automation Conference*, page To appear, 1997.

[70] J. Snoeyink and J. Stolfi. Objects that cannot be taken apart with two hands. In *Proc. ACM Symp. on Computational Geometry*, pages 247–256, 1993.

[71] J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations.* Academic Press, 1991.

[72] A. Subramani. *Development of a Design for Service Methodology.* Ph.D. thesis, Dept. Industrial and Manufacting Eng., U. Rhode Island, Kingston, RI, 1992.

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

[74] R. Wilson. *On Geometric Assembly Planning.* Ph.D. thesis, Dept. Comput. Sci., Stanford Univ., Stanford, CA, 1992. Stanford Technical Report STAN-CS-92-1416.

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

[76] R. Wilson and J.-C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(2):371–396, 1994.

[77] R. Wilson and J. Rit. Maintaining geometric dependencies in and assembly planner. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 890–895, 1990.

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

[79] T. Woo and D. Dutta. Automatic disassembly and total ordering in three dimensions. *J. Engineering for Industry*, 113(2):207–213, 1991.

[80] A. Yao. A lower bound for the monotone depth of connectivity. In *Proc. 35th Symp. on Found. Comput. Sci.*, pages 302–308, 1994.