International Journal of Computational Geometry & Applications
Vol. 9, Nos. 4 & 5 (1999) 371–417
© World Scientific Publishing Company

COMPLEXITY MEASURES FOR ASSEMBLY SEQUENCES*

MICHAEL H. GOLDWASSER†
Department of Computer Science
Stanford University
Stanford, CA 94305-9045, USA

and

RAJEEV MOTWANI‡
Department of Computer Science
Stanford University
Stanford, CA 94305-9045, USA

Received 3 January 1997
Revised 24 October 1997
Communicated by P. Agarwal, Guest Editor

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. 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. To begin, we define, "virtual assembly sequencing," 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. We formally prove the hardness of finding even near-optimal sequences for most cost measures in our generalized framework. As a special case, we prove equally strong inapproximability results for the problem of scheduling with AND/OR precedence constraints. Finally, 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, for example using an assembly consisting solely of unit disks in the plane.

Keywords: Assembly sequencing, approximability, cost measures, AND/OR scheduling, non-directional blocking graphs

*Preliminary versions of the results of this paper have appeared in the *Proc. of the IEEE International Conference on Robotics and Automation*, 1996,[26] and in the *Proc. of the Workshop on Algorithms and Data Structures*, 1997.[27] A more complete treatement of this material is given in the doctoral thesis of M. Goldwasser.[25]

†wass@cs.stanford.edu

‡rajeev@cs.stanford.edu

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 that guarantee to find a valid assembly sequence when one exists. 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. The IEEE Technical Committee on Assembly and Task Planning[28] summarized the state of assembly sequencing in 1994 by explaining, "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, as the assembly sequence will be used in mass quantities during manufacturing. Many attributes of a sequence will have a direct effect on the cost of manufacturing, for example, the number of operations required, the number of different directions of motion used, and so on. 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. Because modern products are being designed with hundreds or thousands of parts or more, the efficiency of algorithms used for sequencing is also critical.

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 finding optimal or even near-optimal cost sequences in a variety of settings. We attempt to classify the difficulty of finding near-optimal solutions for 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 cost measures includes many of those 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). 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 collection of such graphs has been computed for a representative set of motions, it can be analyzed to efficiently compute a feasible assembly sequence, when one exists. Through the introduction of the *non-directional blocking graph*,[68,70] this technique has resulted in much of the success in finding feasible 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.

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. Our strongest results show that it is hard to find any sequence, for many of the VAS variants, with a cost which can be bounded to within a $2^{\log^{1-\gamma} n}$-factor[a] 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[b]. We prove similar inapproximability results for this scheduling problem. 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 indeed 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 disk is hard for any $\gamma > 0$. A more complete summary of our exact results is given later, in Tables 2–5.

The paper proceeds as follows. In Section 2, we discuss previous work in the areas of assembly sequencing, approximation theory, and computational geometry, which relate to our work. In Section 3, we review the definition of the assembly sequencing problem and specifically examine the development of the non-directional blocking graph in Section 3.1. We introduce our *virtual assembly sequencing* problem in Section 4, as we formalize a list of possible tasks, restrictions, and cost measures for assembly sequencing. In Section 5, we define the AND/OR scheduling problem. Next, we approach the issue of how well the choice of sequences can be optimized over such cost measures. Even in cases where finding the exact optimal solution is difficult, it is desirable to find approximate solutions which are near-optimal. In Section 6, we give reductions relating the many different variants of the problems to each other. Then in Sections 7 and 8, we prove that finding even an approximate solution for most variants is quite hard in our generalized framework. Finally, we re-introduce the geometry in Section 9, proving the inapproximability for several cost measures, even in the original geometric settings. Section 10 discusses some particularly interesting complexity issues relating to AND/OR scheduling, and finally Section 11 concludes with a discussion of open directions for continued research.

[a]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.

[b]Note, this model is in no way related to the AND/OR tree used by Homem de Mello and Sanderson[35,37] for representing all feasible assembly sequences

2. Previous Work

2.1. Assembly Sequencing

The use of automation in assembly sequencing has increased rapidly over the years.[18,34,36,37,52,68,72] 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.[37,71] The problem of finding a valid assembly sequence was later shown to be intractable in many general settings.[38,45,46,58,69,72] 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, and *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 developed to find an assembly sequence when one exists.[29,32,68,70] A good deal of this success can be achieved within the framework of *non-directional blocking graphs*.[31,68,70] As our work is intricately related to this approach, in Section 3.1 we review the concept of non-directional blocking graphs and the subsequent results. Other techniques allow for the enumeration of all possible assembly sequences,[18] however for most products there will be exponentially many such sequences. There are also several important tasks, such as removing a given part for service, which arise as natural *disassembly* problems. For this reason, some researchers have focused specifically on models for disassembly.[50,63,66]

With the ability to find feasible sequences efficiently, several researchers have focused on the importance of using automated tools to choose *cost-effective* sequences. There are many possible ways to define the cost of a sequence, depending on how these sequences will be used in a manufacturing system. Based on a great deal of work with industrial applications, Boothroyd *et al.*[9,10] suggest several empirical measures that effect the cost of assembly for a product; more formal complexity measures have been defined by Wolter[72] and by Wilson and Latombe[70]; and a collection of many cost measures for assembly planning has been gathered by Jones and Wilson.[41] Once a cost measure has been chosen, the question becomes how to find a low cost assembly sequence efficiently. Several papers have considered methods for logically grouping parts of an assembly,[12,56] thereby reducing the effective number of parts in an assembly and thus reducing the search time required for finding an optimal sequence. Although these techniques are quite practical, they simply delay the eventual need for better automated reasoning to overcome the exponential computation required for increasingly large data sets. Researchers have also considered the use of simulated annealing[55] and petri nets[11] in finding low-cost assembly sequences. 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,[20,73] however the required input property does not have a clear definition. For the general setting, our results in Section 9.3 will prove not only the difficulty of finding an optimal sequence by this cost measure, but even a near-optimal solution. Finally, several software systems offer the user the option of optimizing the sequence over a choice of complexity measures,[44,62,72] however these systems must rely on current techniques and thus either require possibly exponential search techniques to find the true optimum, or else polynomial heuristics with no performance guarantees on the cost of the resulting sequence.

2.2. Approximation Theory

For most of our variants, finding the optimal cost assembly sequence is 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 solution. Unless P=NP, there is little hope of efficiently finding the true optimal solution, however this does not rule out the possibility of finding approximate solutions efficiently. 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 well received.

Research in the theory of approximability has consider exactly this issue for other NP-hard optimization problems.[4,22,40,57] 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 similar optimization problems can be quite different, ranging from those which can be approximated arbitrarily closely to the optimum, to those for which getting even a very rough approximation is already NP-hard.

Many researches have worked towards classifying the approximability of different NP-hard problems.[4,5,14,59] We will consider four broad classes defined by Arora and Lund[4] and shown in Table 1, which group problems based on the strength of the inapproximability results which have been proven. Class I includes all prob-

Table 1. The four classes and their representative problems (Arora and Lund[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

lems 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[59] problems, for example VERTEX COVER, METRIC TSP, MAX CUT, and others. Class II groups those problems for which it is quasi-NP-hard[c] 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 $\ln n(1+o(1))$.[21] For problems in Class III, it is quasi-NP-hard to achieve a $2^{\log^{1-\gamma} n}$ factor approximation for any $\gamma > 0$. LABEL COVER[3] is the canonical problem in this class, although the class contains several other natural problems such as LONGEST PATH[43] 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[33] and COLORING.[53]

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 sub-collection 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[3,4] 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$_{\text{min}}$ is to give a labeling that covers all edges, while minimizing the total number of labels, counting multiplicities, assigned to nodes of U.

Finally, we need to define the notion of *approximation-preserving reductions*.[57,59] 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 approximate solutions, hence 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 optimal solution of one problem can be used to find the optimal solution 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 $c \geq 0$. Notice that this allows us

[c]Whereas a problem is NP-hard if solving it would imply NP $\subseteq$ DTIME$(n^{O(1)})$, a problem is quasi-NP-hard if solving it would imply that 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]

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 for some constant $c \geq 0$. 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 change 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 we show 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[d]. 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 polynomial 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.

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 has been well studied in the geometric community. For example a classic result of Guibas and Yao[30] concerns a collection of convex parts in two dimensions. The result stats 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. Such a sequence, removing objects one at a time using translations in a common direction, is known as a *depth order*. Similar separability issues are studied by Toussaint[67] for more general classes of shapes in two dimensions, such as monotone or star-shaped polygons. For a collection of balls in $\mathcal{R}^d$, Dawson[15] shows that there exists at least $d+1$ balls, each of which can be translated to infinity in some direction. In contract, Snoeyink and Stolfi[64] demonstrate a set of convex parts in three dimensions which cannot be disassembled using two hands. A study for constructing a sequence for the individual removal of polygons from a collection in two dimensions introduced a structure termed the *movability wheel*,[19] which can be thought of as a precursor to

[d]Technically, to place a problem into Class I through this type of reduction, it is also necessary to pay attention to the value of the constant c.

the non-directional blocking graph. More recent work has looked at the efficiency of computing and verifying depth orders.[13,16,17]

A surprising element of our results is that some of our general lower bounds given in Section 8, are realized geometrically in Section 9. More often than not, an optimization problem becomes significantly easier when its input is restricted to a low-dimensional, geometric setting. For example, there exists some $c > 0$ for which achieving a $(1+c)$-approximation for the METRIC TSP problem is NP-hard,[60] 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,[33] 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.[6,47] There exists a $(1+o(1))\ln n$ lower bound for approximating the SET COVER problem,[21] however the RECTANGLE COVER problem, covering a set of axis-aligned rectangles with minimum number of points, has no such inapproximability results.[57] 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.

3. 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 from infinity. The classic goal is to produce a sequence of operations resulting in the construction of the product from its individual parts. Each operations combines a set of subassemblies, using a motion 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 will assume that the product is made of *rigid* parts, we assume that each operation is *two-handed*, 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 are often consistent with the technical capabilities of manufacturing systems.

Under these conditions, we may think of devising an assembly sequence by constructing a *disassembly* sequence and then reversing the entire sequence. If considering non-rigid parts, stability, fixturing, and insertion forces, assembly and disassembly sequences are no longer symmetric.[51] 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 eliminated more quickly in this way. Additionally, there are several jobs related to maintenance or recycling of products which require a partial disassembly of a complete product. With this in mind, the goal of a two-handed, 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. The structure of this decomposition can be represented naturally as a binary *assembly tree.* Figure 1, provided by Wilson and Latombe,[70] gives an example of such an assembly tree for a simple two-dimensional product. The root of the tree represents the fully

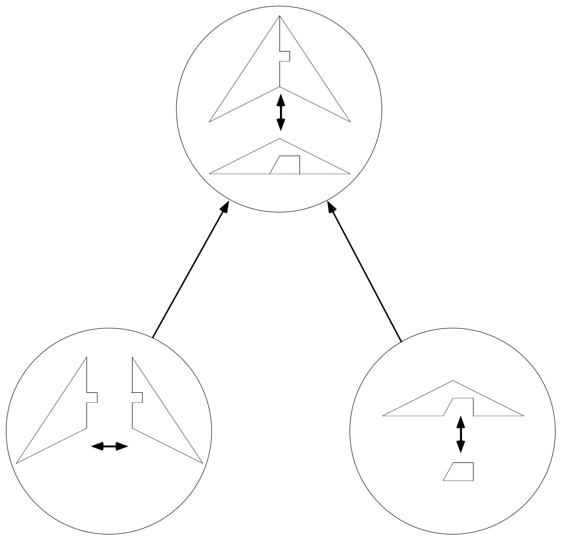


Fig. 1. Assembly tree for a simple product

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, not the desired sequence in which the operations are performed.

3.1. 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 $A \to B$, if part A collides with part B when that motion is applied to A while B remains stationary. Figure 2 again shows the two-dimensional product of Wilson and Latombe,[70] with two particular DBG's for infinitesimal translation. The blocking graph for a given motion provides a compact representation of all collision-free partitions for that motion, as each directed cut between some subsets S and $\overline{S}$ in a DBG corresponds to a collision-free action separating S from $\overline{S}$. Since a DBG represents

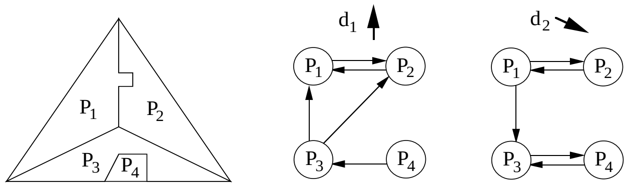


Fig. 2. A simple assembly and two DBG's for infinitesimal translation

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*[68,70] (NDBG). During the construction of an NDBG, the space of motions is 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 of Ref. [68], it is shown 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,[70] extended translations (i.e., to infinity),[70] multiple step translations in the plane,[32] and infinitesimal generalized motions (i.e., rigid body motions).[29,70] 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[31] presents a unified framework for understanding the collection of work surrounding the non-directional blocking graphs.

For each of these families of motions, the NDBG framework immediately provides a polynomial-time algorithm for constructing a feasible (dis)assembly sequence, if one exists. After constructing a polynomial set of DBG's, an arbitrary disassembly 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 polynomial time, is quite simple, and has been implemented in assembly sequencing systems for many of the above motion classes.[29,44,62,68] As we will see, searching for a "good" sequence in this way is not quite so simple.

4. Virtual Assembly Sequencing (VAS)

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 two-handed, monotone assembly sequence that optimizes some cost measure. 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.

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.

INPUT:	A set $\mathcal{P}$ of n items.
	A family $\mathcal{F}$ of directed graphs on n labeled nodes.
OUTPUT:	A (dis)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 decompose $\mathcal{P}'$ into sets S and $\mathcal{P}' - S$ if the graph d has no edges directed from a part in S to a part in $\mathcal{P}' - S$. In graph-theoretic terms, an operation is legal if the partition provides a *directed cut* on the subgraph of d induced by the set $\mathcal{P}'$.

It is important to understand that VAS is truly more general than the original problem, 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 has been efficiently computed. 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 additional structure. It is conceivable that this structure could 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 Section 9.

4.1. Possible Goals

In Section 3, we said that the goal of a two-handed, monotone assembly sequencer, viewed in light of assembly-by-disassembly, is to produce a sequence of actions 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, discussed in Section 3.

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 key parts need not be completely isolated at their leaves, they must simply be 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.

4.2. Possible Restrictions

Manufacturing systems often impose additional constraints on assembly sequences other than simply geometric feasibility. We consider two such restricted versions of the assembly sequencing problem.

R1 **Linear Sequence.**[62,70]
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]
Manufacturing systems may sometimes be constrained to use only a small set of pre-defined motions, due to the existing robotics systems, or other considerations such as fixturing and stability. For instance, a system may be constrained to using only axis-aligned translations. When constrained to a small number of motions, it may be possible that there exist better sequencing algorithms than for the more general problem.

Therefore we consider the restricted problem for which the number of input graphs in $\mathcal{F}$ is bounded by some constant, k.

4.3. 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 might 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.**[62,70,72]
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.**[44,51,72]
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-orienting 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 offers no savings unless the product is still in that orientation.

C3 **Fewest Number of Non-Linear Steps.**[70]
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.**[73]
The cost of an assembly sequence is equal to the total number of operations used. Notice that for goal G1, full disassembly, every possible *two-handed* 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.**[70]
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.

4.4. Immediate Observations

At this point, we consider variants of VAS which can be solved through fairly immediate observations from the definitions in this section. 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 immediate polynomial-time algorithms for exact or approximate minimum 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 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.

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. If we ever reach a subassembly for which there is no legal operation, we are assured that those parts could not have been separated by any sequence.

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 (R1) 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.

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

Observation 7 A *stack* assembly is defined as a product that can be completely (dis)assembled using translations along a single direction.[70] 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/C1). This is true with or without the linear restriction.

The proof of this relies on the fact that with a constant number of graphs, there are a constant number of possible subsets of directions. We may simply try each possible subset 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 For any of the five goals, when minimizing the number of re-orientations when restricted to a constant number of graphs (R2/C2), we are able to find the optimal solution in polynomial time when $|\mathcal{F}| = 2$, and we are able to find an $(|\mathcal{F}| - 1)$-approximation in polynomial time, when $|\mathcal{F}| \geq 3$. This is true with or without the restriction to linear operations.

These results can be achieved using a set of universal sequences, similar to the techniques used for approximating the Shortest Common Supersequence problem.[7] For more discussion on the relation between re-orientations and supersequences, see the proofs of Theorems 14 and 16.

Observation 10 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 (cost C4).

This proof relies on the result of Observation 9, combined with a construction which translates between counting re-orientations versus counting steps with a factor of two increase.

5. AND/OR Scheduling

As a special case, we consider the goal of removing a key part, when restricted to linear steps, while minimizing the number of steps (G2/R1/C4). In this situation, 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 which can be scheduled, and the linear disassembly sequence is simply a schedule for the order of removal. The goal of the scheduling problem is to successfully schedule a key task and the cost is equal to the total number of scheduled tasks[e].

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 problem from more traditional scheduling is the form of the precedence constraints. Commonly, a task t 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 t; see Ref. [22]. 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

[e]We could also consider the problem of removing a set of parts in this way (G3/R1/C4).

same part in some other direction, in which case a different set of parts may block the removal. It is worth noting that with classical AND precedence constraints, this problem of minimizing the number of scheduled tasks can be solved exactly, in polynomial time by computing a depth order.

We consider other models 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 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 Section 9.3. Theorem 7, in Section 8, will specifically address the issues of modeling VAS using AND/OR precedence constraints (as opposed to DNF constraints).

We formally define scheduling with AND/OR precedence constraints, as follows, for an instance with 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, can only be scheduled after *all* tasks in P_i. An OR-task, t_j, can only be scheduled 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*[f], with a node for each task, t_i, and a directed edge[g] from t_i to t_j whenever $t_j \in P_i$, is a direct predecessor of t_i 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.

A similar model for scheduling with AND/OR precedence constraints has been studied earlier by Gillies and Liu.[23,24] However, they only consider the case we define as partial-order precedence constraints. With classical AND precedence constraints, it is assumed that there is no cycle in the precedence graph, as a cycle would make the problem infeasible. With AND/OR constraints, this is no longer a necessary condition for the existence of a valid solution, and in fact cycles will often exist as it may be the case that part A blocks B in one direction, B blocks C in another,

[f]Not to be confused with the original blocking graphs.

[g]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.

and C blocks A in a third. For this reason, we have made no a priori assumptions about the structure of the precedence relations. The work of Refs. [23,24] 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 for a single processor. 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.

6. 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 4. We give several approximation preserving reductions which demonstrate that certain variants are at least as hard to approximate as others. A summary of the reductions is given in Figure 3, although without noting the applicable cost measures. The proofs are given later in this section. Because of the sufficiently strong lower bounds we will prove in Sections 7 and 8, 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.

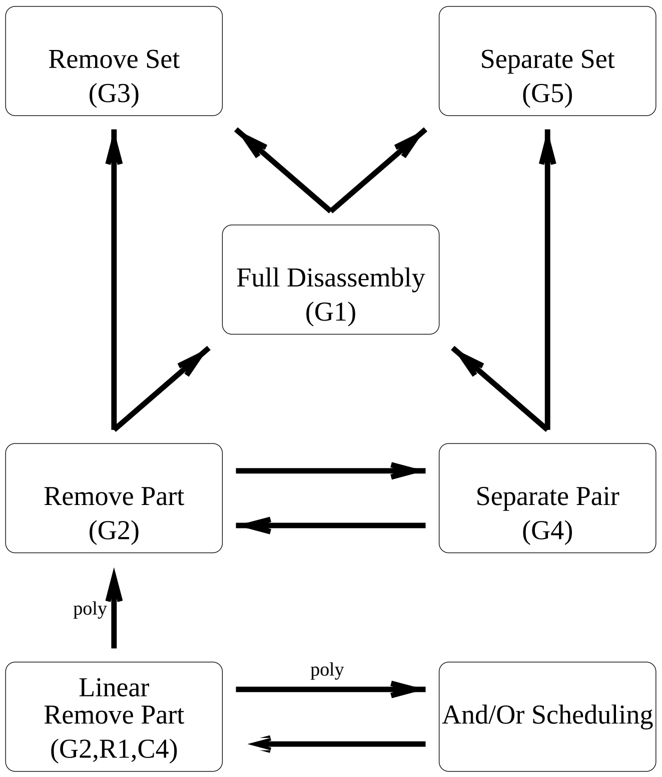


Fig. 3. Reductions between variants of VAS

Theorem 1 [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 2 [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 3 [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 4 [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 5 [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 6 [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 7 [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 8 [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 9 [G2 $\Longrightarrow$ G3, G1 $\Longrightarrow$ G3]
The problem of removing a single key part reduces to the problem of removing a set of parts. Similarly, the problem of full disassembling a product reduces to the problem of removing a set of parts. This is true for all cost measures and all restrictions.

Theorem 10 [G4 $\Longrightarrow$ G5, G1 $\Longrightarrow$ G5]
The problem of separating a pair of parts reduces to the problem of separating a

set of parts from each other. Similarly, the problem of full disassembling a product reduces to the problem of separating a set of parts from each other. This is true for all cost measures and all restrictions.

Proof of Theorem 1. 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.

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 which is the complete graph with the two edges (k, k') and (k', k) removed. 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 assembly, 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, in this case 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 2. 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 4. It is easy to verify that if

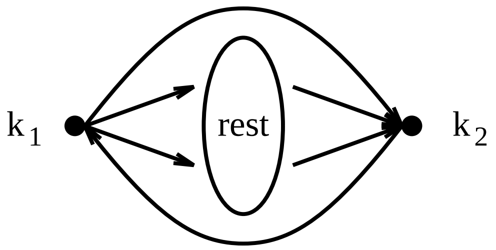


Fig. 4. Separating a pair reduces to removing a part

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. 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 3. 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 with edges (k, a) and (a, k) for all parts $a \neq k$. 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 nonlinear steps[h]. 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 key part 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 4. 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 explicitly require the restriction to linear operations. We will implicitly enforce the linearity

[h]Notice that for Cost C5, there may be a logarithmic increase in the depth. This problem is handled separately in Theorem 5.

by converting each graph into n graphs, each of which only allows for the removal of a specific part.

We keep the same set of n parts, and we create a new family of $n|\mathcal{F}|$ graphs. 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 1. We claim that the only possible action allowed by this graph is to remove the single part p from a subassembly, 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 restricted to a constant size family of graphs (R2). □

Proof of Theorem 5. The proof of this theorem is a simple combination of the techniques from the proofs of Theorems 3 and 4. In the proof of Theorem 3, 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 4, 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 star 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. $\square$

Proof of Theorem 6. 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 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. $\square$

Proof of Theorem 7. 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 which we equate with the statement, "part p has been removed." For each pair $\langle p, d\rangle$, with part $p \in \mathcal{P}$ and direction $d \in \mathcal{F}$, we create task $\hat{t}_{(p,d)}$, which we equate with the statement "I am prepared to remove part p in direction d." We make $\hat{t}_{(p,d)}$ an AND-task with task t_a in its predecessor set for every outgoing[i] edge

[i]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.

$(p, a) \in d$ (we must have already removed all parts blocking p in direction d). We make task t_p an OR-task with task $\hat{t}_{(p,d)}$ in its predecessor set for each direction d (we must have at least one direction which allows for the removal).

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. □

Proof of Theorem 8. 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 difference between these cost measures for a solution is that in one of them, linear steps may be done for free. Our solution for this reduction 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 omit the details of this construction. □

Proof of Theorem 9. Both of these problems are simply special cases of removing a set of parts, as mentioned in Section 4.1. □

Proof of Theorem 10. Both of these problems are simply special cases of separating a set of parts from each other, as mentioned in Section 4.1. □

7. Inapproximability of AND/OR Scheduling

In this section, we prove the inapproximability of minimizing the number of tasks schedule for an instance of scheduling with AND/OR precedence constraints, as defined in Section 5. As a precursor, we prove the inapproximability of minimizing the number of *leaves* scheduled in an instance of AND/OR scheduling with *internal-tree* precedence constraints[j]. We prove this result by showing that the LABEL COVER$_{\text{min}}$ problem is a special case. We then convert this result to prove a similar bound when we further restrict the AND/OR problem to have degree bounded by two. Following this, we convert our bound on minimizing the number of schedule leaves into a bound on the total number of scheduled nodes for the general AND/OR scheduling problem. An overview of the reductions is given in Figure 5.

Theorem 11 *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* $\gamma > 0$. *This remains true even if both the*

[j]The internal-tree defines a monotone, boolean function 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. Therefore, our results also prove the inapproximability of this problem on *monotone* boolean formulae. We are unaware of any previous results for this approximation problem. Minimizing the number of ones in satisfying a 3CNF formula is known to be $n^{0.5-\epsilon}$-hard to approximate,[42] and related minimization problems are studied in Ref. [48].

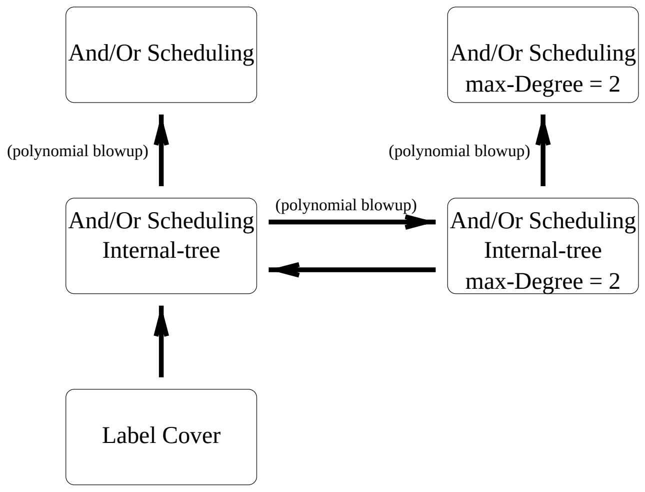

And/Or Scheduling
And/Or Scheduling max-Degree = 2
(polynomial blowup)
(polynomial blowup)
And/Or Scheduling Internal-tree
(polynomial blowup)
And/Or Scheduling Internal-tree max-Degree = 2
Label Cover

Fig. 5. Reductions between variants of AND/OR scheduling

AND-*degree and* OR-*degree are bounded by two.*

Theorem 12 *For the problem of minimizing the number of tasks scheduled in a general instance of* AND/OR *scheduling, it is quasi-NP-hard to achieve an approximation ratio of* $2^{\log^{1-\gamma} n}$ *for any* $\gamma > 0$*. This lower bound remains valid if both the* AND-*degree and* OR-*degree are bounded by two.*

Proof of Theorem 11. Given an instance of LABEL COVER$_{\mathrm{min}}$, as defined in Section 2, we express it as an instance of AND/OR scheduling with internal-tree precedence constraints, as shown in Figure 6.

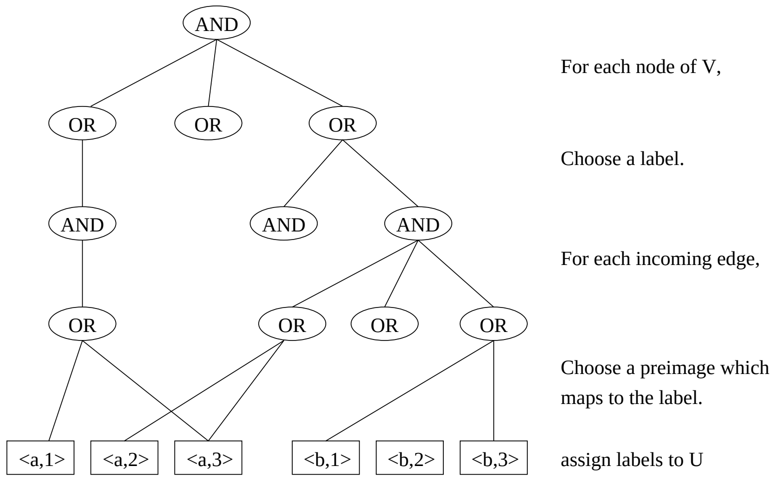


Fig. 6. LABEL COVER as AND/OR scheduling with internal-tree precedence

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 nodes 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 can only be covered if 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.

For the case of unbounded degree, this completes the construction. It can be seen that there is a one-to-one correspondence between valid labelings in the LABEL COVER$_{\text{min}}$ instance and valid solutions to the AND/OR 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 we have given an approximation-preserving reduction with polynomial blowup. Combining this with a previous[4] lower bound for the approximability of LABEL COVER,, we get that it is quasi-NP-hard to achieve an $2^{\log^{1-\gamma} n}$ approximation for any $\gamma > 0$.

To prove the bound with the maximum degree equal to two, we can replace each internal node in the obvious way, 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 must 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 $I(I + L)$ internal nodes, which is still polynomially bounded in L, and thus our reduction runs in polynomial time. □

Proof of Theorem 12. 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 11, 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.

With the appropriate choice of α, the additive error can be made arbitrarily small, giving us an approximation-preserving reduction with polynomial blowup in the input size □

8. Inapproximability of Virtual Assembly Sequencing

We present our core results regarding the VAS model in this section. We gave reductions, in Section 6, which related various versions of VAS to each other, in terms of the level of their approximability. Now we will rely on the hardness of some other problems to prove the difficulty of finding optimal or even near-optimal solutions for most variants of VAS. Our first set of results will be a direct consequence of the hardness of AND/OR scheduling shown in Section 7. Our second set of results will show weaker inapproximability results for a case not covered by our other reductions, namely 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. As there are numerous combinations of goals, restrictions, and cost measures, we present our complete list of results for VAS in Tables 2–4.

Theorem 13 *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 (G2/R1/C4). This result applies even when $|\mathcal{F}| = 2$ (G2/R1R2/C4).*

Proof. This is a direct result of Theorem 12, combined with Theorem 6. □

Corollary 1 *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 2–4.*

Proof. These results are a combination of Theorem 13, together with the many reductions given in Section 6. See Tables 2–4 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 14 *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. Our result is based on a reduction to a related problem, introduced as the Loading Time Scheduling Problem[7] (LTSP). In this scheduling model, certain tasks must be completed on certain machines, however the dominant cost is not

Table 2. 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 4 from G2/R1/C4	$2^{\log^{1-\gamma} n}$-hard Theorem 4 from G2/R1/C4	solvable Observation 8	solvable Observation 8
Re-orientations C2	$2^{\log^{1-\gamma} n}$-hard Theorem 4 from G2/R1/C4	$2^{\log^{1-\gamma} n}$-hard Theorem 4 from G2/R1/C4	$\|\mathcal{F}\| = 2$: solvable $(\|\mathcal{F}\| - 1)$-approximable Observation 9 $\|\mathcal{F}\| = 3$: NP-complete $\|\mathcal{F}\| \geq 4$: $\|\mathcal{F}\|^{\alpha}$-hard Theorem 14 from LTSP	$\|\mathcal{F}\| = 2$: solvable $(\|\mathcal{F}\| - 1)$-approximable Observation 9 $\|\mathcal{F}\| = 3$: NP-complete $\|\mathcal{F}\| \geq 4$: $\|\mathcal{F}\|^{\alpha}$-hard Theorem 14 from LTSP
Non-Linear Steps C3	$2^{\log^{1-\gamma} n}$-hard Theorem 8 from G2/C4	n.a.	$\|\mathcal{F}\| = 3$: NP-complete $\|\mathcal{F}\| \geq 4$: $\|\mathcal{F}\|^{\alpha}$-hard Theorem 8 from G2/R2/C4	n.a.

Steps C4 or C5	$2^{\log^{1-\gamma} n}$-hard
	Theorem 4 from G2/R1/C4
		$2^{\log^{1-\gamma} n}$-hard
		Theorem 13 from AND/OR
			$2(\|\mathcal{F}\| - 1)$-approx
			Observation 10
			$\|\mathcal{F}\| = 3$: NP-complete
			$\|\mathcal{F}\| \geq 4$: $\|\mathcal{F}\|^{\alpha}$-hard
			Theorem 14
			from LTSP
				$\|\mathcal{F}\| \geq 2$: $2^{\log^{1-\gamma} n}$-hard
				Theorem 13
				from AND/OR with
				maxdeg = 2

Table 3. 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 1 from G2/C1	$2^{\log^{1-\gamma} n}$-hard Theorem 1 from G2/R1/C1	solvable Observation 8	solvable Observation 8
Re-orientations C2	$2^{\log^{1-\gamma} n}$-hard Theorem 1 from G2/C2	$2^{\log^{1-\gamma} n}$-hard Theorem 1 from G2/R1/C2	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert - 1)$-approximable Observation 9 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 14 from LTSP	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert - 1)$-approximable Observation 9 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 14 from LTSP
Non-Linear Steps C3	$2^{\log^{1-\gamma} n}$-hard Theorem 8 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 8 from G4/R2/C4	n.a.
Steps C4 or C5	$2^{\log^{1-\gamma} n}$-hard Theorem 1 from G2/C4	$2^{\log^{1-\gamma} n}$-hard Theorem 1 from G2/R1/C4	$2(\lvert\mathcal{F}\rvert - 1)$-approx Observation 10 $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $\lvert\mathcal{F}\rvert^{\alpha}$-hard Theorem 14 from LTSP	$\lvert\mathcal{F}\rvert \geq 4$: $2^{\log^{1-\gamma} n}$-hard Theorem 1 from G2/R1R2/C4

Table 4. 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 3 from G2/C1	$2^{\log^{1-\gamma} n}$-hard Theorem 3 from G2/R1/C1	solvable Observation 8	solvable Observation 8
Re-orientations C2	$2^{\log^{1-\gamma} n}$-hard Theorem 3 from G2/C2	$2^{\log^{1-\gamma} n}$-hard Theorem 3 from G2/R1/C2	$\|\mathcal{F}\| = 2$: solvable $(\|\mathcal{F}\| - 1)$-approximable Observation 9 $\|\mathcal{F}\| = 3$: NP-complete $\|\mathcal{F}\| \geq 4$: $\|\mathcal{F}\|^{\alpha}$-hard Theorem 14 from LTSP	$\|\mathcal{F}\| = 2$: solvable $(\|\mathcal{F}\| - 1)$-approximable Observation 9 $\|\mathcal{F}\| = 3$: NP-complete $\|\mathcal{F}\| \geq 4$: $\|\mathcal{F}\|^{\alpha}$-hard Theorem 14 from LTSP
Non-Linear Steps C3	$2^{\log^{1-\gamma} n}$-hard Theorem 3 from G2/C3	n.a.	$\|\mathcal{F}\| = 3$: NP-complete $\|\mathcal{F}\| \geq 4$: $\|\mathcal{F}\|^{\alpha}$-hard Theorem 3 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 5 from G2/R1/C5	n.a.	$\frac{n}{\lceil \log_2 n \rceil}$-approximable Observation 6	n.a.

actually the time required to run the tasks, rather the expensive context switches in going from using one machine to another. This model relates quite naturally to our original motivation for considering the re-orientation cost measure, namely that orienting a robot to perform insertions in a given directions is quite expensive, but once it is oriented, inserting several more parts in negligible.

The exact definition for LTSP is given as follows. There is a set of n jobs, and ρ machines, and each job, j, can only be performed by some subset of the machines, $M(j)$. The jobs have (standard) precedence constraints, represented by a directed acyclic graph G. Each machine m_i has a loading time $l(m_i)$ which must be paid in order to prepare a machine, however once that machine is loaded, it may perform any available operations at no additional cost. The overall cost for a schedule is the sum of the machine loading times.

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[k], 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 7.

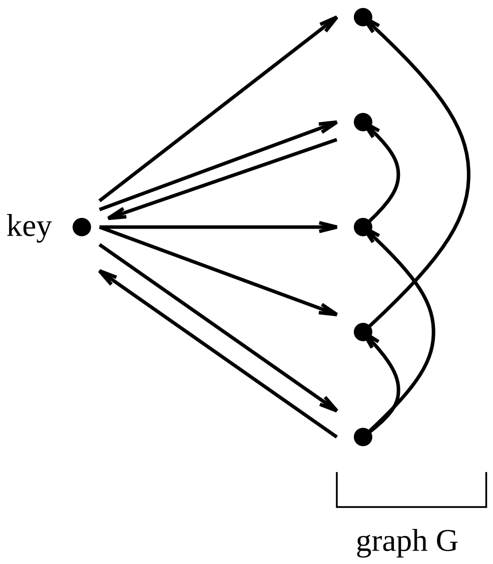


Fig. 7. Reduction from Loading Time Scheduling Problem

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

[k]actually, we reversal all edges of G, as Ref. [7] defines an edge from x to y as signifying that y cannot be run until after x.

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, then 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 Ref. [7] combined with a result of Ref. [54], 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 1. 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. When considering the number of steps, we note that all of the progress of each re-orientation can be made using a single step. In this way, we get similar lower bounds for variant R2/C4, when applied to all goals except full disassembly. □

9. Geometric Lower Bounds

While the inapproximability results of Section 8 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.

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, with the proofs to follow. The results for all goals in a geometric setting are summarized in Table 5.

Theorem 15 *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 five goals. This is valid with or without the linear restriction, R1.*

Theorem 16 *We consider minimizing the number of re-orientations used when restricted to linear steps (R1/C2), for a polygonal 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 17 *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.*

9.1. 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 8, modified from Ref. [62]. Now we consider the minimum number of directions which must be used to remove all the pegs from the base. As pointed out in Ref. [72], this instance looks very much like a SET COVER problem, in that we must choose a minimum number of directions,

Table 5. 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 15	3D-polygons (G1, G2, G3, G4, G5) NP-hard Theorem 15	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 16	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert - 1)$-approximable Observation 9	$\lvert\mathcal{F}\rvert = 2$: solvable $(\lvert\mathcal{F}\rvert - 1)$-approximable Observation 9 2D-polygons (G1, G2, G3, G4, G5) $\lvert\mathcal{F}\rvert = 3$: NP-complete $\lvert\mathcal{F}\rvert \geq 4$: $(1+c)$-hard Theorem 16
Steps C4		DISKS setting (G2, G3, G4, G5) $2^{\log^{1-\gamma} n}$-hard Theorem 17	$2(\lvert\mathcal{F}\rvert - 1)$-approx Observation 10	DISKS setting (G2, G3, G4, G5) $\lvert\mathcal{F}\rvert \geq 2$: $2^{\log^{1-\gamma} n}$-hard Theorem 17

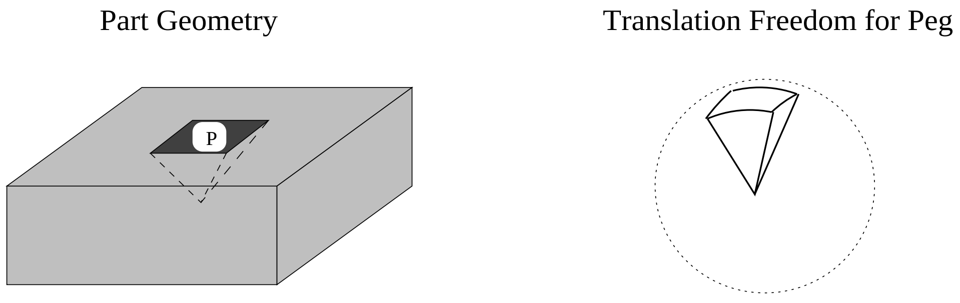


Fig. 8. A peg in a base

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:	$\lvert\mathcal{P}\rvert$, the number of points.

A similar, even more restricted problem, RECTANGLE COVER is defined in Ref. [57], where all polygons are axis-aligned rectangles. It is not known whether RECTANGLE COVER is NP-hard.

Lemma 1 *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.[22] 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. □

Lemma 2 *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.*

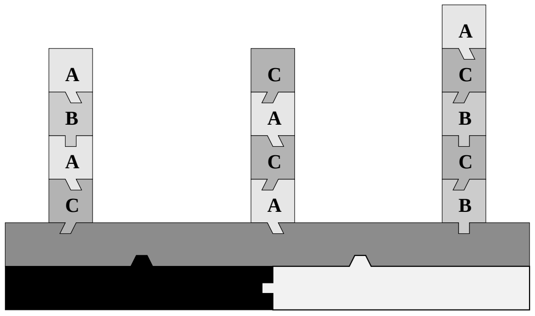
Proof. Given a set of polygons in the plane, we will consider the corresponding homogeneous coordinates[65] to project them onto the upper hemisphere. Given a single such projection, we can design a peg which can be removed from the base using exactly those directions represented by the polygon. We simply 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. □

Proof of Theorem 15. For the goal of removing a key part, this theorem is a result of Lemmas 1 and 2. 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 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 and G5, are generalization of the others, and thus the hardness results also apply. □

9.2. Finding a Common Supersequence

Proof of Theorem 16. When constrained to using linear moves, we give a construction which reduces the problem of finding a common supersequence, 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 full disassembly. The base is a long, flat rectangle, and each sequence will be represented as a tower of 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. An example is given in Figure 9, with two additional pieces added to the base, for reasons described later.

If we assume that none of the input sequences contain 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



S = {ABAC, CACA, ACBCB}

Fig. 9. Example construction for SCS reduction

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,[22] and more recently it was shown to be Max-SNP-hard, even over a binary alphabet.[8] Therefore, by doubling the alphabet as above, we get that our disassembly problem is Max-SNP-hard when $|\mathcal{F}| \geq 4$. When $|\mathcal{F}|$ is not restricted to be constant, there exists a constant $\delta > 0$, such that approximating the shortest common supersequence to within a factor of $log^{\delta}n$ is quasi-NP-hard.[39] Finally, even if strings have consecutive occurrences of the same symbol, finding a common supersequence with the minimum number of *runs* is NP-complete for an alphabet size $|\Sigma| = 3$.[54] 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 9, 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 operations, 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 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. □

9.3. The Disks *Problem*

Proof of Theorem 17. 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 Theorem 11, 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 10.

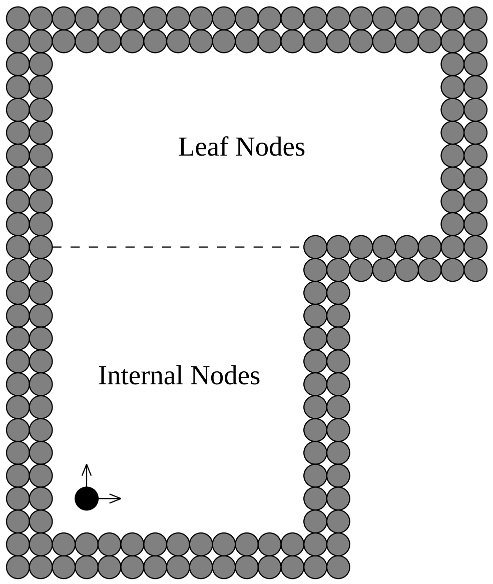


Fig. 10. Overview of DISKS construction

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 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. See Figure 11.

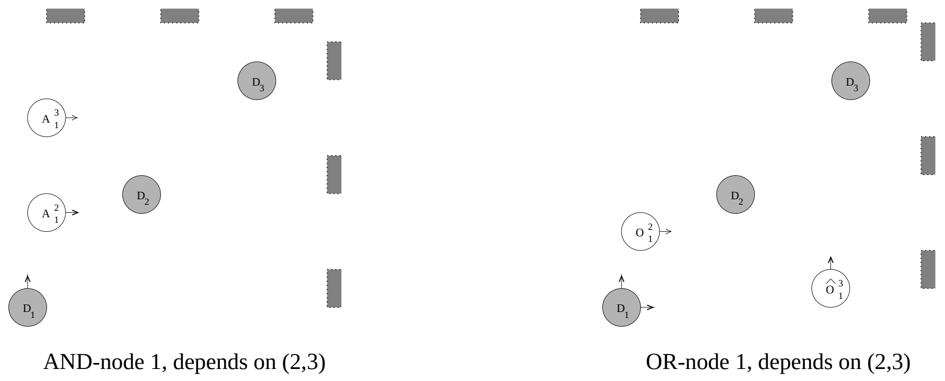


Fig. 11. Internal node mechanisms

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 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 $(6(I+1), 6(I+1)+2Ia)$. 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 horizontal strip may escape. Now, for every internal node T_i which depends on leaf L_a, we create a disk L^i_a, located at $(6i+1, 6(I+1)+2Ia+2i)$, which is forced East by the walls. Figure 12 shows an example of a leaf mechanism.

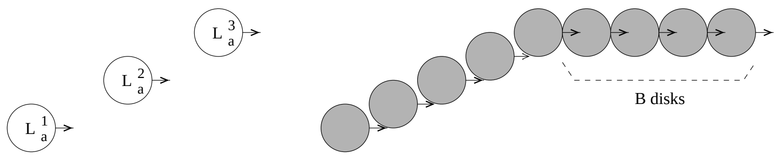


Fig. 12. Leaf node mechanism: Internal nodes $1, 2, 3$ depend on Leaf a.

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 13.

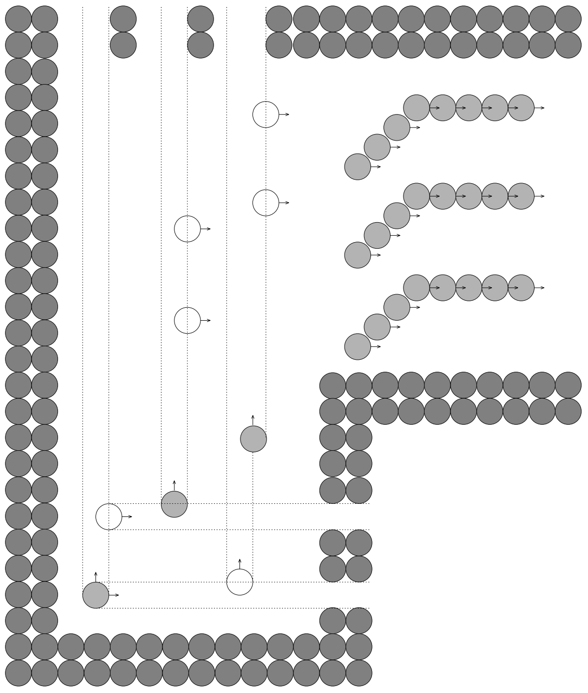


Fig. 13. A complete DISKS construction

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. $\square$

10. On the Hardness of AND/OR Scheduling

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 standard scheduling problem, yet the effect of this change on the difficulty of the problem is quite dramatic. We pose a series of open directions of research related to where this problem fits into the theory of approximability.

In Section 7, we considered 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$_{\min}$ 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.

We examined a very structured class of instances of AND/OR scheduling which had what we termed *internal-tree* precedence constraints, and we looked at the problem of minimizing the number of *leaves* that are scheduled. Without loss of generality, we can assume that the root of our tree is an AND-node. Without 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 levels of 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 the 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 11, we can already express LABEL COVER$_{\min}$. However it is not at all clear that this problem is equivalent to LABEL COVER$_{\min}$as we do not know whether an instance of this restricted AND/OR scheduling can be translated into a LABEL COVER instance. 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 these versions? 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[48] which considers the the problem of minimizing the number of ones required to satisfy a collection of constraints on boolean variables. The 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; see Refs. [49,61,74]. 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.

11. 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 this 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 identifying 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.

Finally, although we have tried to compile a relatively complete list of cost measures and restrictions which have been considered by previous research, there are countless other natural variants which could be added to our model of virtual assembly sequencing. As new settings are motivated by industry, we hope to see the continuation of this analysis in studying exactly what can and cannot be done by polynomial-time assembly sequencers. Hopefully, our work can set a standard for the analysis of optimization in assembly sequencing.

Acknowledgments

The authors wish to thank the members of the Stanford University Assembly Planning group, namely Jean-Claude Latombe, Leo Guibas, Dan Halperin, Cyprien Godard, G. Ramkumar, and Bruce Romney, for their involvement in the initial formulation of these problems. Additional thanks go to Randy Wilson, for many helpful discussions and suggestions about our model, and to Dan Halperin for his suggestion that we study the (easy) setting involving unit disks in the plane, and subsequent discussions. Our appreciation goes to Chandra Chekuri and Sanjeev Khanna for their participation in the study of scheduling with AND/OR precedence constraints. Finally, we appreciate the many comments and suggestions provided by the anonymous referees.

M. Goldwasser was supported 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. R. Motwani was supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership Award, an ARO MURI Grant DAAH04-96-1-0007, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

References

1. P. K. Agarwal, M. de Berg, D. Halperin, and M. Sharir, "Efficient generation of k-directional assembly sequences," in *Proc. Seventh Annual ACM-SIAM Symp. on Discrete Algorithms*, Atlanta, Georgia, Jan. 1996, pp. 122–131.
2. S. Arora, "Polynomial-time approximation schemes for Euclidean TSP and other geometric problems," in *Proc. 37th Symp. on Foundations of Computer Science*,

Burlington, Vermont, Oct. 1996, pp. 1–11.

3. S. Arora, L. Babai, J. Stern, and Z. Sweedyk, "The hardness of approximate optima in lattices, codes and linear systems," *J. Comput. Syst. Sci.* **54** (1997) 317–331.
4. S. Arora and C. Lund, "Hardness of approximations," in *Approximation Algorithms for NP-Hard Problems*, ed. D. Hochbaum (PWS Publishing, Boston, 1996) pp. 399–446.
5. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and the hardness of approximation problems," *J. ACM* **45** (1998) 501–555.
6. B. Baker, "Approximation algorithms for NP-complete problems on planar graphs," *J. ACM* **41** (1994) 153–180.
7. R. Bhatia, S. Khuller, and J. Naor, "The loading time scheduling problem," in *Proc. 36th Symp. on Foundations of Computer Science*, Milwaukee, Wisconsin, Oct. 1995, pp. 72–81.
8. P. Bonizzoni, M. Duella, and G. Mauri, "Approximation complexity of longest common subsequence and shortest common supersequence over fixed alphabet," Technical Report 117/94, Università degli Studi di Milano, 1994.
9. G. Boothroyd, *Assembly Automation and Product Design* (Marcel Dekker, New York, 1991).
10. G. Boothroyd, P. Dewhurst, and W. Knight, *Product Design for Manufacture and Assembly* (Marcel Dekker, New York, 1994).
11. S. Caselli and F. Zanichelli, "On assembly sequence planning using petri nets," in *Proc. IEEE Int. Symp. on Assembly and Task Planning*, Pittsburgh, Pennsylvania, Aug. 1995, pp. 239–244.
12. S. Chakrabarty and J. Wolter, "A hierarchical approach to assembly planning," in *Proc. IEEE Int. Conf. on Robotics and Automation*, San Diego, California, May 1994, pp. 258–263.
13. 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** (1992) 305–323.
14. 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.
15. R. Dawson, "On removing a ball without disturbing the others," *Mathematics Magazine* **57** (1984) 27–30.
16. M. de Berg, *Ray Shooting, Depth Orders and Hidden Surface Removal* (Springer-Verlag, Berlin, 1993).
17. M. de Berg, M. Overmars, and O. Schwarzkopf, "Computing and verifying depth orders," *SIAM J. Comput.* **23** (1994) 432–446.
18. T. De Fazio and D. Whitney, "Simplified generation of all mechanical assembly sequences," *IEEE Trans. on Robotics and Automation* **3** (1987) 640–658.
19. F. Dehne and J.-R. Sack, "Translation separability of polygons," *Visual Computer* **3** (1987) 227–235.
20. D. Dutta and T. Woo, "Algorithm for multiple disassembly and parallel assemblies," *J. Engineering for Industry* **117** (1995) 102–109.
21. U. Feige, "A threshold of $\ln n$ for approximating set cover," in *Proc. 28th ACM Symp. on Theory of Computing*, Philadelphia, Pennsylvania, May 1996, pp. 314–318.
22. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of*

NP-Completeness (W. H. Freeman, New York, 1979).

23. D. Gillies, "Algorithms to schedule tasks with AND/OR precedence constraints," Ph.D. thesis, University of Illinois, Urbana, IL, 1993.
24. D. Gillies and J. Liu, "Scheduling tasks with AND/OR precedence constraints," *SIAM J. Comput.* **24** (1995) 797–810.
25. M. Goldwasser, "Complexity measures for assembly sequencing," Ph.D. thesis, Dept. Comp. Sci., Stanford Univ., Stanford, CA, 1997. Stanford Technical Report STAN-CS-TR-97-1590.
26. M. Goldwasser, J.-C. Latombe, and R. Motwani, "Complexity measures for assembly sequences," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Minneapolis, Minnesota, Apr. 1996, pp. 1581–1587.
27. M. Goldwasser and R. Motwani, "Intractability of assembly sequencing: Unit disks in the plane," in *Proc. Fifth Workshop on Algorithms and Data Structures*, Lecture Notes in Computer Science **1272** (Springer-Verlag, Berlin, 1997) pp. 307–320.
28. S. Gottschlich, C. Ramos, and D. Lyons, "Assembly and task planning: A taxonomy," *IEEE Robotics and Automation Magazine* **1** (1994) 4–12.
29. L. J. Guibas, D. Halperin, H. Hirukawa, J.-C. Latombe, and R. H. Wilson, "Polyhedral assembly partitioning using maximally covered cells in arrangements of convex polytopes," *Int. J. Computational Geometry and Applications* **8** (1998) 179–199.
30. L. J. Guibas and F. F. Yao, "On translating a set of rectangles," in *Computational Geometry*, Advances in Computing Research, ed. F. Preparata (JAI Press, 1983) pp. 61–77.
31. D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," in *Proc. 14th Annual Symp. on Computational Geometry*, Minneapolis, Minnesota, June 1998. To appear in *Algorithmica*.
32. D. Halperin and R. Wilson, "Assembly partitioning along simple paths: the case of multiple translations," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Nagoya, Japan, May 1995, pp. 1585–1592.
33. J. Håstad, "Clique is hard to approximate within $n^{1-\epsilon}$," in *Proc. 37th Symp. on Foundations of Computer Science*, Burlington, Vermont, Oct. 1996, pp. 627–636.
34. R. Hoffman, "A common sense approach to assembly sequence planning," in *Computer-Aided Mechanical Assembly Planning* (Kluwer Academic, Boston, 1991) pp. 289–314.
35. L. Homem de Mello and A. Sanderson, "AND/OR graph representation of assembly plans," *IEEE Trans. on Robotics and Automation* **6** (1990) 188–199.
36. L. Homem de Mello and A. Sanderson, *Computer-Aided Mechanical Assembly Planning* (Kluwer Academic, Boston, 1991).
37. 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** (1991) 228–240.
38. 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** (1984) 76–88.
39. T. Jiang and M. Li, "On the approximation of shortest common supersequences and longest common subsequences," *SIAM J. Comput.* **24** (1995) 1122–1139.
40. D. Johnson, "Approximation algorithms for combinatorial problems," *J. Comput. Syst. Sci.* **9** (1974) 256–278.

41. R. Jones and R. Wilson, "A survey of constraints in automated assembly planning," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Minneapolis, Minnesota, Apr. 1996, pp. 1525–1532.
42. V. Kann, "Polynomially bounded minimization problems which are hard to approximate," in *Proc. 20th Int. Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **700** (Springer-Verlag, Berlin, 1993) pp. 52–63.
43. D. Karger, R. Motwani, and G. Ramkumar, "On approximating the longest path in a graph," in *Proc. Third Workshop on Algorithms and Data Structures*, Lecture Notes in Computer Science **709** (Springer-Verlag, Berlin, 1993) pp. 421–432.
44. 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*, Minneapolis, Minnesota, Apr. 1996, pp. 3361–3368.
45. L. Kavraki and M. Kolountzakis, "Partitioning a planar assembly into two connected parts is NP-complete," *Inf. Process. Lett.* **55** (1995) 159–165.
46. L. Kavraki, J.-C. Latombe, and R. Wilson, "Complexity of partitioning an assembly," in *Proc. Fifth Canadian Conf. on Computational Geometry*, Waterloo, Canada, 1993, pp. 12–17.
47. S. Khanna and R. Motwani, "Towards a syntactic characterization of PTAS," in *Proc. 28th ACM Symp. on Theory of Computing*, Philadelphia, Pennsylvania, May 1996, pp. 329–337.
48. S. Khanna, M. Sudan, and L. Trevisan, "Constraint satisfaction: The approximability of minimization problems," in *Proc. 12th IEEE Computational Complexity Conf.*, Ulm, Germany, June 1997.
49. M. Klawe, W. J. Paul, N. Pippenger, and M. Yannakakis, "On monotone formulae with restricted depth," in *Proc. 16th ACM Symp. on Theory of Computing*, Washington, D.C., 1984, pp. 480–487.
50. K. Lee and R. Gadh, "Computer aided design for disassembly: a destructive approach," in *Proc. 1996 IEEE Int. Symp. on Electronics and the Environment*, Dallas, Texas, May 1996, pp. 173–178.
51. S. Lee, "Backward assembly planning with assembly cost analysis," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Nice, France, May 1992, pp. 2382–2391.
52. S. Lee and Y. Shin, "Assembly planning based on geometric reasoning," *Computers and Graphics* **14** (1990) 237–250.
53. C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems," *J. ACM* **41** (1994) 960–981.
54. M. Middendorf, "Supersequences, runs, and CD grammar systems," in *Developments in Theoretical Computer Science*, Topics in Computer Science **6**, eds. J. Dassow and A. Kelemenova (Gordon and Breach, Amsterdam, 1994) pp. 101–114.
55. 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*, San Diego, California, May 1994, pp. 2058–2063.
56. H. Moradi, K. Goldberg, S. Lee, and R. Wilson, "Geometry-based part grouping for assembly planning," Manuscript, 1997.
57. R. Motwani, "Approximation algorithms," Technical Report STAN-CS-92-1435, Stanford University, 1992.
58. B. Natarajan, "On planning assemblies," in *Proc. Fourth Annual Symp. on Computational Geometry*, Urbana-Champaign, Illinois, June 1988, pp. 299–308.

59. C. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *J. Comput. Syst. Sci.* **43** (1991) 425–440.
60. C. Papadimitriou and M. Yannakakis, "The traveling salesman problem with distances one and two," *Mathematics of Operations Research* **18** (1993) 1–11.
61. R. Raz and A. Wigderson, "Monotone circuits for matching require linear depth," *J. ACM* **39** (1992) 736–744.
62. B. Romney, C. Godard, M. Goldwasser, and G. Ramkumar, "An efficient system for geometric assembly sequence generation and evaluation," in *Proc. 1995 ASME Int. Computers in Engineering Conf.*, Boston, Massachusetts, Sept. 1995, pp. 699–712.
63. N. Shyamsundar and R. Gadh, "Selective disassembly of virtual prototypes," in *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, Beijing, China, Oct. 1996, pp. 3159–3164.
64. J. Snoeyink and J. Stolfi, "Objects that cannot be taken apart with two hands," in *Proc. Ninth Annual Symp. on Computational Geometry*, San Diego, California, May 1993, pp. 247–256.
65. J. Stolfi, *Oriented Projective Geometry: A Framework for Geometric Computations* (Academic Press, Boston, 1991).
66. A. Subramani, "Development of a design for service methodology," Ph.D. thesis, Dept. Industrial and Manufacting Eng., U. Rhode Island, Kingston, RI, 1992.
67. G. Toussaint, "Movable separability of sets," in *Computational Geometry*, ed. G. Toussaint (North-Holland, Amsterdam, 1985) pp. 335–375.
68. R. Wilson, "On geometric assembly planning," Ph.D. thesis, Dept. Comput. Sci., Stanford Univ., Stanford, CA, 1992. Stanford Technical Report STAN-CS-92-1416.
69. R. Wilson, L. Kavraki, J.-C. Latombe, and T. Lozano-Pérez, "Two-handed assembly sequencing," *Int. J. Robotics Research* **14** (1995) 335–350.
70. R. Wilson and J.-C. Latombe, "Geometric reasoning about mechanical assembly," *Artificial Intelligence* **71** (1994) 371–396.
71. R. Wilson and J. Rit, "Maintaining geometric dependencies in an assembly planner," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Cincinnati, Ohio, May 1990, pp. 890–895.
72. J. Wolter, "On the automatic generation of plans for mechanical assembly," Ph.D. thesis, University of Michigan, 1988.
73. T. Woo and D. Dutta, "Automatic disassembly and total ordering in three dimensions," *J. Engineering for Industry* **113** (1991) 207–213.
74. A. Yao, "A lower bound for the monotone depth of connectivity," in *Proc. 35th Symp. on Foundations of Computer Science*, Sante Fe, New Mexico, 1994, pp. 302–308.