Complexity Measures for Assembly Sequences

Michael Goldwasser[*] Rajeev Motwani [†]

Department of Computer Science
Stanford University
Stanford, CA 94305-9045

Abstract

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

We begin by introducing a formal framework for studying the optimization of several complexity measures. We consider a variety of different settings and natural cost measures for assembly sequences. Following which, we define a graph-theoretic problem which 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. For our *virtual assembly sequencing* problem we are able to use techniques common to the theory of approximability to prove the hardness of finding even near–optimal sequences for most cost measures in our generalized framework. As a special case, we prove strong inapproximability results for the problem of scheduling with AND/OR precedence constraints.

Of course, hardness results in our generalized framework do not immediately carry over to the original geometric problems. We continue by realizing several of these hardness results in rather simple geometric settings, proving the difficulty of some of the original problems. We are able to show strong inapproximability results in a far simpler setting than the domain of most assembly sequencers, for example using an assembly consisting solely of unit disks in the plane. These inapproximability results, to the best of our knowledge, are the strongest hardness results known for a purely combinatorial problem in a geometric setting.

[*]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.

[†]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.

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 which results in the assembly of the product from the individual parts. Efficient algorithms have been developed, for many classes of motions, which are guaranteed to find a valid assembly sequence when one exists. The IEEE Technical Committee on Assembly and Task Planning summarized the current state of assembly sequencing by explaining [21], "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. In manufacturing, the resulting assembly sequence will be performed in mass quantities, and so the cost of the sequence is of great importance. Furthermore, for products where the optimal cost assembly sequence proves to be expensive, this information can be used by engineers in modifying the design of the product.

Unfortunately, there has been little success in optimizing the cost of assembly sequences. Our work explains this lack of progress by formally proving the hardness of approximating the optimal cost sequence in a variety of settings. We attempt to classify the approximability of many variants of assembly sequencing, based on the desired cost measure, the specific goal required, and other restrictions placed on the sequence. Our list of possible cost measures are motivated directly by industry, and include many measures suggested by previous researchers. Examples include minimizing the number of distinct directions of motion used during a sequence, minimizing the number of steps in a sequence, or minimizing the number of re-orientations of the assembly.

We begin by studying a graph-theoretic generalization of assembly sequencing which we term *virtual assembly sequencing* (VAS). Much of the success in finding feasible sequences has been a result of the introduction of the *non-directional blocking graph* [50, 52]. For a given direction of motion, the geometric model of the product can be analyzed to construct a graph which represents the blocking relationships among the parts. Once a set of such graphs has been computed, they can be analyzed to compute a feasible (dis)assembly sequence, when one exists. Our 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 prove that for many of the variants, it is hard to find any sequence whose cost can be bounded to within a $2^{\log^{1-\gamma} n}$-factor[1] of the optimal cost sequence for any $\gamma > 0$. Our reductions show that assembly sequencing simultaneously encompasses difficulties seen in covering, scheduling, and supersequencing problems. 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[2]. We prove similar inapproximability results for this scheduling problem.

Finally, since our virtual assembly sequencing model is a generalization, our lower bounds do not necessarily apply to the original problem as we no longer assume that the set of input graphs are the result of any original geometric setting. We continue by showing that many of our lower bounds can, in fact, be realized geometrically, thereby proving the hardness of the true assembly sequencing problems. As an example, we consider a setting consisting entirely of unit disks in the plane, and we look at the task of removing a given disk from the rest of the assembly. We prove that achieving

[1]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.

[2]N.B.: this is not to be confused with the AND/OR tree representation of assembly sequences[28]

a $2^{\log^{1-\gamma} n}$-approximation to minimizing the total number of disks which must be removed to access the given disks is hard for any $\gamma > 0$.

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 we 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.

Following this, we approach the problem 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 approximate the problem by finding such a near–optimal solution. In Sections 7 and 8, we present our results which prove that in our generalized framework, finding even an approximate solution for most cost measures is quite hard. We re-introduce the geometry in Section 9, to prove the inapproximability of several variants of the original assembly sequencing problems. Section 10 discusses some particularly interesting complexity issues relating to AND/OR scheduling, and finally Section 11 discuss several other open directions for research, and contains concluding remarks.

2 Previous Work

2.1 Assembly Sequencing

The use of automation in assembly sequencing has increased rapidly over the years [7, 16, 26, 27, 28, 37, 39, 50, 51, 53, 54]. 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 inefficient generate-and-test sequencers, operating by generating candidate operations and testing their feasibility [28, 53]. However, if arbitrarily complex motions and paths are allowed, assembly sequencing was shown to be intractable [29, 34, 43, 54]. This led some researchers to consider restricted, but still interesting, versions of the problem (e.g., *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, described by a constant number of degrees of freedom, polynomial algorithms were developed which will find an assembly sequence if one exists [22, 24, 50, 52]. Most of this success can be achieved within the framework of *non-directional blocking graphs* [50, 52], and as our work is intricately related to this approach, we review these approaches in more detail in Section 3.1. It is also possible to enumerate all possible assembly sequences [16], although there may be exponentially many such sequences for a product.

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 eventually in a manufacturing system. Several empirical measures have been suggested [10], and more formal complexity measures are examined in a generalized system [54]. The importance of using automated reasoning to evaluate the complexity of assembly sequences is discussed, along with the introduction of several evaluation measures [52]. A hierarchical approach is used to identify common subassemblies in products, thereby allowing more effort to be used towards finding a "better" assembly sequence [11]. Although practical, this technique simply delays the eventual need for better automated reasoning to overcome increasingly large data sets. For a restricted class of inputs which have a so-called "total ordering" property, a greedy algorithm is given which claims to produce the minimal length sequence to remove any given part [55], however the required input property does not have a clear definition, and as our results will show, this problem is quite difficult.

Class	Factor of Approximation that is hard	Representative Problems
I	$1+\epsilon$	Max-3Sat
II	$O(\log n)$	Set Cover
III	$2^{\log^{1-\gamma} n}$	LabelCover
IV	n^{ϵ}	Clique

The four classes and their representative problems. (Table 10.1 from [4])

Several software systems offer the user the option of optimizing the sequence over a choice of complexity measures [33, 47, 54], however these systems currently must rely on either a brute force search of the entire space, or else branch-and-bound type searches with no performance guarantees for 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, brute-force search of all possibilities. For this reason, there is little hope of efficiently finding the true optimal solution, however this does not rule out the possibility of finding approximate solutions efficiently. 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.

For this reason, we approach these problems using techniques common to the theory of approximability [4, 18, 31, 42]. Since we cannot expect to find the optimal sequence in polynomial time, we look for a polynomial time *approximation algorithm* which 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* between the cost of the solution returned by the algorithm versus the cost of the optimal solution. Although all NP-complete decision problems can be reduced to one another, the approximability of such problems can be quite different, ranging from problems which can be approximated arbitrarily close to optimal, to problems where 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, 12, 44]. We will consider four broad classes defined in [4], which group problems based on the strength of the inapproximability results which have been proven. Class I includes all problems for which approximating the optimal solution to within a factor of $(1+\epsilon)$ is NP-hard for some $\epsilon > 0$. The canonical problem for this class is Max-3Sat, and the class includes all Max-SNP-complete problems [44], for example Vertex Cover, Metric TSP, Max Cut, and others. Class II groups those problems for which it is quasi-NP-hard[3] 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 $\ln n(1 + o(1))$ [17]. For problems in Class III, it is quasi-NP-hard to achieve a $2^{\log^{1-\gamma} n}$ factor approximation for any $\gamma > 0$. LabelCover is the canonical problem in this class[3], although the class contains several other natural problems such as Longest Path [32] and Nearest Lattice Vector [3]. Finally, Class IV consists of the hardest problems, namely those

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

for which it is NP-hard to achieve an n^ϵ approximation factor for some $\epsilon > 0$. This class includes problems such as CLIQUE [25] and COLORING [40].

We give a series of results, proving not only the hardness of finding the exact optimal solutions in this model, but even of finding reasonable approximation algorithms. To do so, we use *approximation-preserving reductions* [42, 44]. 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. Such classical reductions do not guarantee anything about the relation between approximate solutions, and this in part explains the vast difference between the approximability of various NP-complete problems. Therefore, to compare the approximability of difficult problems, it is necessary to use such approximation-preserving reductions which show not only that finding an optimum of one problem can be used to find an optimal of the other, but also that finding an approximate solution can be translated to an approximate solution for the other of similar performance ratio.

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 [13, 14, 15, 23, 48, 49]. In a single direction, a *depth order* of a set of parts is an ordering of the parts which allows for collision-free translations of the individual parts to infinity. A classic result states that given a collection of convex shapes in two dimensions, for any translational direction there exists some ordering, such that the parts can be translated away one at a time [23]. Therefore, if trying to minimize the number of directions needed for an assembly sequence, this result tells us that one direction is always sufficient for an assembly of convex parts in two dimensions. Similar separability issues are studied in two dimensions for more general classes of shapes, such as monotone or star-shaped polygons [49]. For a collection of balls in $\mathcal{R}^d$, there exists at least $d+1$ balls, each of which can be translated to infinity in some direction [13]. However, there exists a set of convex parts in three dimensions which cannot be disassembled using two hands, with only translations to infinity, or even generalized rotations and translations [48].

A surprising element of our problem is that the general lower bounds given in Section 8, are realized in Section 9, for a simple geometric setting consisting of a collection of unit disks in the plane. More often than not, an optimization problem becomes significantly easier when its input is restricted to a geometric setting. For example, there exists some $c > 0$ for which achieving a $(1+c)$-approximation for the METRIC TSP problem is NP-hard [45], 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 [25], 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 [35]. There exists a $(1+o(1))\ln n$ lower bound for approximating the SET COVER problem [17], however the RECTANGLE COVER problem, covering a set of axis-aligned rectangles with minimum number of points, has no such inapproximability results [42].

To the best of our knowledge, the geometric results in Section 9 provide the strongest inapproximability bounds shown 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.

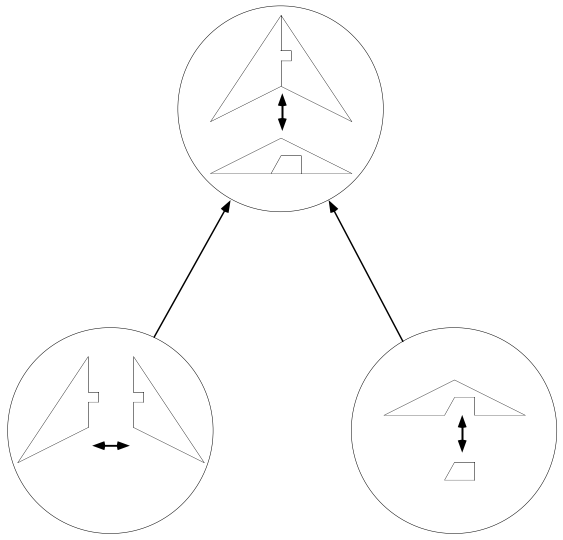


Figure 1: Assembly tree for a simple product

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 to 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 assume that the product is made of *rigid* parts, we assume that each operation is binary, that is, combines exactly two subassemblies, and we consider only monotone assembly sequences, that is, when an operation has placed a part in a subassembly, that part may no longer be moved relative to the subassembly. Although restrictive, these assumptions are common in assembly sequencing and can be applied to a majority of products.

Under these conditions, we may think of devising an assembly sequence by constructing a *disassembly* sequence and then reversing the entire sequence[4]. The advantage of reasoning about disassembly is that the final assembled product is usually much more constrained than the initial configuration of parts, and so infeasible plans can be more quickly eliminated in this way. Additionally, there are several jobs related to maintenance or recycling of products which require a partial disassembly of a complete product.

With this in mind, the goal of a binary, monotone assembly sequencer is to start with the fully assembled product and partition the set of parts into two groups which can be *separated* by a collision-free motion. Once this is done, each of the resulting subassemblies can be disassembled. This decomposition can be represented naturally as a binary *assembly tree*. The root of the tree represents the fully-assembled product, and the children of each internal node represent the two subassemblies which are combined to produce the larger subassembly. For a more detailed discussion,

[4]In general, these tasks are not symmetric, for instance when flexible parts are deformed during assembly [38].

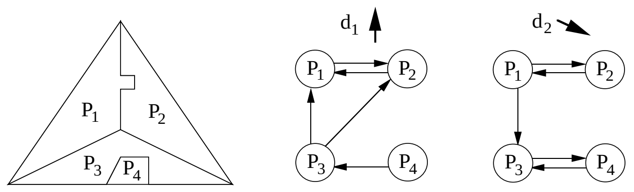


Figure 2: A simple assembly and two DBG's for infinitesimal translation

see [52, 50, 54]. Figure 1 gives an example, taken from [52], of such an assembly tree for a simple two-dimensional product. Note that the assembly tree represents the set of operations used in the decomposition, however it does not represent the exact sequence in which the operations are performed.

3.1 A Review of Non-directional Blocking Graphs

The first key concept in understanding current techniques in assembly sequencing is that of a *directional blocking graph* (DBG). For a specific motion, a DBG can be defined as a directed graph with a node for each part of the assembly, and an edge $A \rightarrow B$, if part A collides with part B when that motion is applied to A, while B remains stationary. Figure 2, also from [52], gives an example of a two-dimensional product as well as two DBGs for infinitesimal translation. The blocking graph for a given motion provides a compact representation of all collision-free operations for that motion, in that a directed cut between subset S and subset T in a DBG exactly represents a collision-free separation. Since a DBG represents all possible operations for a single direction of motion, by constructing a DBG for each possible motion, we can fully represent all possible operations. Unfortunately, the family of motions may have infinite size.

However, it was noticed that many distinct motions may be represented by the identical DBG, since slight changes in a motion may not effect the blocking relationships between any of the parts. The construction of a *non-directional blocking graph* (NDBG) divides the space of motions 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 general, it seems that if the family of motions has a constant number of degrees of freedom, then the number of distinct equivalence classes is polynomially bounded. Techniques from computational geometry allow for the construction of the NDBG for a variety of motion classes, including infinitesimal translations [52], extended translations (i.e., to infinity) [52], multiple step translations [24], and infinitesimal generalized motions (i.e., rigid body motions) [22, 52].

For each of these families of motions, the NDBG framework immediately provides a polynomial time algorithm for constructing a feasible assembly sequence, if one exists. After constructing the set of DBG's, an arbitrary assembly sequence is found by taking any legal separation using any of the directions, and then recursing on the resulting subassemblies. Since the removal of parts can only reduce the blocking relationships, there will be no false dead ends and this procedure will result in either producing an entire assembly tree, or else will reach a subassembly which cannot be partitioned by any of the motions, thereby proving that no assembly sequence exists. This algorithm runs in polynomially time, is quite simple, and has been implemented to construct assembly sequences for many of the above motion classes [33, 47]. As we will see, searching for a "good" sequence in this way is not quite so simple.

4 Virtual Assembly Sequencing (VAS)

Our generalized framework is based directly on the work of the NDBG, as we assume that our problem begins as we are handed the full set of DBG's, and our goal is to choose a binary, monotone assembly sequence. Notice that the NDBG is simply a re-organization of the original input to aid in sequencing. The complete set of directed graphs contains all of the information about the input relevant to assembly sequencing, and so nothing is lost by considering the graphs as the input to the problem. Therefore, we will define the *virtual assembly sequencing* problem as a graph-theoretic problem where the sole input is a set of directed graphs, and thus the VAS problem directly captures the original assembly sequencing problem for any geometric setting in which the NDBG has been constructed in polynomial time.

What is important to understand is that VAS is a generalization of the original problem, and this is because we will not make any assumptions about the structure of the individual graphs, or their interdependence on each other. In reality, when an NDBG is constructed from a geometric description of a product, the resulting set of blocking graphs may have some hidden structure. It is conceivable that this structure would allow for additional success in devising assembly sequences, and therefore VAS may indeed be a strictly harder problem than the original assembly sequencing problem. However to date, no such hidden structure has been used successfully for any geometric setting, and current systems based on the NDBG eventually analyze the set of graphs at face value in finding assembly sequences.

For this reason, we formally define VAS as the following graph-theoretic problem.

INPUT:	A set, $\mathcal{P}$ of n items.
	[We will call each member of this set a "part."]
	A polynomial-sized family, $\mathcal{F}$, of directed graphs on n nodes.
	[We will call each member of this family a "direction."]
OUTPUT:	An assembly sequence for $\mathcal{P}$ using only "legal" operations based on $\mathcal{F}$.

We inherit the definition of "legal" motions from the notion of directed blocking graphs. Given a subset of parts $P' \subseteq P$, a direction $d \in \mathcal{F}$ can be used to partition P' into sets A and B if the graph d has no edges directed from a part in B to a part in A, (i.e., the partition provides a directed cut on the induced subgraph for P').

4.1 Possible Goals

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

G1 **Full disassembly.**
This is the classical problem. In our terms the goal is to find a sequence of operations which begin 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.** [47]
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* which 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 which 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 which need to be treated specially when they are brought together.

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

G5 **Separate a given set.**
Rather than a pair of parts, a set of parts is given here, and the goal is to decompose the assembly so that no two of the key parts are in the same subassembly. When the key set consists of two parts, this is exactly goal G4, and when the set consists of all parts, this is identical to goal G1.

4.2 Possible Restrictions

Often, manufacturing systems impose additional constraints on assembly sequences other than simply geometric feasibility. Although we are ignoring many such important issues, we consider a few such restricted versions of the assembly sequencing problem.

R1 **Linear Sequence.** [47, 52]
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 which 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-sized Family of Motions.** [1]
Originally, our only assumption was that the number of blocking graphs is polynomially bounded in the number of parts. Now we consider instances where the number of graphs in bounded by some constant, k.

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

Once constrained to a small number of input graphs, we are interested in whether or not the smaller input allows for better sequencing algorithms than does the more general problem.

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 will give a different definition of which is better for their 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 studying these basic measures in depth is a necessary first step before examining specialized combinations of complexity measures.

C1 **Fewest Number of Directions.** [47, 52, 54]
The cost of an assembly sequence is equal to the number of directions in $\mathcal{F}$ which 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.** [54]
The cost of an assembly sequence is equal to the number of operations which use a direction which is *different* from the previous 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 assemblies all parts must be physically inserted from above and thus an operation in a different direction is performed by re-oriented the subassembly on the assembly line so that the desired direction is aligned vertically. This is typically slow and might require additional expensive fixtures. In both of these cases, using an orientation that was encountered earlier in the process is of no advantage unless the product is still in that orientation.

C3 **Fewest Number of Non-Linear Steps.** [52]
A step 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 steps. 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 **Minimum Depth of an Assembly Sequence.** [52]
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, as

the minimum depth tree has the quickest "throughput." As a special case, when the goal is to remove a key part, than this cost is equal to the depth of the key leaf, and thus exactly the *number of steps* taken to free the part from the rest of the product.

C5 **Fewest Number of Removed Parts.**
This measure is specific to the partial disassembly problems. We define the exact measure as $(n - M)$ where M is the number of parts in the largest, intact subassembly. We will consider this subassembly as the "main" assembly, and thus all other parts were "removed" at some point from the main assembly.

4.4 Immediate Observations

We take a few moments to make some initial observations about our model, and several immediately results. This also gives the reader a chance to digest the problem.

- 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.

- 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 operations is a mistake in terms of feasibility, as the removal of parts can only decrease the blocking relationships of future moves. At any point, if we find a subassembly for which there is no legal operation, we are assured that those parts could not have been separated by any sequence.

- A graph admits a valid *linear* operation to remove part p, if and only if part p has no outgoing (incoming) edges. In polynomial time, we may check whether a set of graphs admits a feasible *linear* sequence for any of our goals.

- A *stack* assembly is defined in [52], as a product which can be completely (dis)assembled using translations along a single direction. They observe the a product admits a stack assembly, if and only if one of the blocking graphs is *acyclic*, and this can be checked in polynomial time.

- For restriction R2, when the number of graphs is constant, it is polynomially solvable to find the minimum number of directions required for any of the five goals, with or without the linear restriction. There are a constant number of possible subsets of directions, and so we may simply try each possibility, and check the feasibility of the problem with those graphs, in polynomial time.

- For restriction R2, if $|\mathcal{F}| = 2$, minimizing the number of re-orientations for any of our goals can be solved in polynomial time. For $|\mathcal{F}| \geq 3$, we can achieve a $(|\mathcal{F}| - 1)$-approximation for minimizing the number of re-orientations, using a set of universal sequences, similar to the techniques used for approximating the SHORTEST COMMON SUPERSEQUENCE problem [8]. For more discussion on the relation between re-orientations and supersequences, see the proofs of Theorems 11 and 13.

5 AND/OR Scheduling

As a special case, we consider the goal of removing a key part, while minimizing the number of removed parts, and restricting ourselves to linear steps (G2/R1/C5). In this situation, we can view the VAS problem in a more simple manner as a scheduling problem. The removal of each part is thought of 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[5].

A complication in our formulation as a scheduling problem is that the 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 distinquishes this problem from more traditional scheduling is the form of the precedence constraints. Traditionally, a task may only have AND precedence constraints, in that it has an associated set of tasks all of which must be scheduled before that task. Unfortunately this is not the case in our assembly sequencing problem. In a single direction, if a part is to be removed, then indeed there is a clear set of parts which block that operation and thus must be removed earlier. However, we may choose to remove that same part in some other direction, in which case a different set of constraints apply.

Instead, our problem can be viewed as a special case of scheduling with AND/OR precedence constraints, where the OR's allow us to offer a choice of directions. We will consider scheduling, where every task has a set of direct predecessors, and each task is either an AND-task, in which case it cannot be scheduled until after all of its predecessors, or the task is an OR-task, in which case it cannot be scheduled until after at least one of its predecessors. For our setting, we consider a single processor and unit processing time for all tasks. It is worth noting that with classical AND precedence constraints, this problem of minimizing the number of scheduled tasks can be solved exactly, in polynomially time by computing a depth order.

A model for scheduling with AND/OR precedence constraints has been studied earlier [19, 20], but with one key difference. The precedence constraints for an instance can be represented as a directed graph[6]with each node additionally tagged as either an AND-node or an OR-node; in this previous work, they assume that there is no cycle in this precedence graph, as that 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, and C blocks A in a third. For this reason, we make no apriori assumptions about the structure of the precedence relations. Gillies and Lin [19, 20] prove the NP-hardness of many variants of the problem, however they do not consider the approximability of the hard problems.

It is important to note that in context of assembly sequencing, the precedence constraints for this scheduling problem could be more naturally modeled as DNF-scheduling, where each task has a constraint of the form $(A \wedge B) \vee (A \wedge C \wedge D)$. Clearly, AND/OR scheduling is a special case of this where each constraint is either a conjunction or a disjunction, but not both. We choose to consider the AND/OR scheduling problem for several reasons, most notably because we will use this problem to give a geometric realization of the hardness results for assembly sequencing in Section 9. Therefore, in translating an assembly sequence instance into an AND/OR scheduling instance, we may have to introduce polynomially many artificial tasks in order to break up the DNF formulae into the appropriate form (see Theorem 6).

[5]We could also consider the problem of removing a set of parts in this way (G3/R1/C5).

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

5.1 Notation and Definitions

The input contains a set of tasks, $\mathcal{T}$. Each task, $t_i \in \mathcal{T}$, is labeled as either an AND-task or an OR-task, and has an associated set of tasks, P_i. An AND-task, t_i, cannot be scheduled until after *all* tasks in P_i. An OR-task, t_j, cannot be scheduled until after *at least one* task of P_j. We define the OR-*degree* of an instance as the maximum sized $|P_j|$ over all OR-tasks t_j; similarly, the AND-*degree* of an instance is the maximum sized $|P_i|$ over all AND-tasks t_i. The constraints can be represented by a precedence graph, with a node for each task, t_i, and a directed edge from t_j to t_i whenever $t_j \in P_i$. A *leaf-task* is one with $P_i = \emptyset$, and thus it 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 (as in [19, 20]). We say an instance of AND/OR scheduling has *internal-tree precedence constraints* if there are no cycles, and if the only nodes with more than one outgoing edge are leaf nodes.

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. 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. An overview of the reductions is given in Figure 6, although without noting the applicable cost measures, and the full proofs are given later in this section.

Theorem 1 [G2 $\Longrightarrow$ G4]
The problem of removing a key part from the rest of the assembly can be reduced, in an approximation-preserving fashion, to the problem of separating a key pair of parts from each other for all five cost measures.

Theorem 2 [G4 $\Longrightarrow$ G2]
The problem of separating a key pair of parts from each other can be reduced, in an approximation-preserving fashion, to the problem of removing a key part from the rest of the assembly, for all five cost measures.

Theorem 3 [G2 $\Longrightarrow$ G1]
The problem of removing a key part from the rest of the assembly can be reduced, in an approximation-preserving fashion, to the problem of fully decomposing the product, for the following cost measures: fewest directions, fewest re-orientations, or fewest non-linear steps.

Theorem 4 [G2/R1/C5 $\Longrightarrow$ G2]
Minimizing the number of removed parts for the keypart problem when restricted to linear moves can be reduced, in an approximation-preserving fashion, to minimizing either the number of directions, number of re-orientations, minimum depth, or number of removed parts for the problem of removing a key part with or without the linear restriction. (polynomial blowup in number of graphs)

Theorem 5 [AND/OR $\Longrightarrow$ G2/R1/C5]
An instance of the AND/OR scheduling problem can be written directly as a special case of the problem of minimizing the number of removed parts for the problem of removing a key part when restricted to linear moves. The number of graphs is exactly equal to the OR-degree of the scheduling problem.

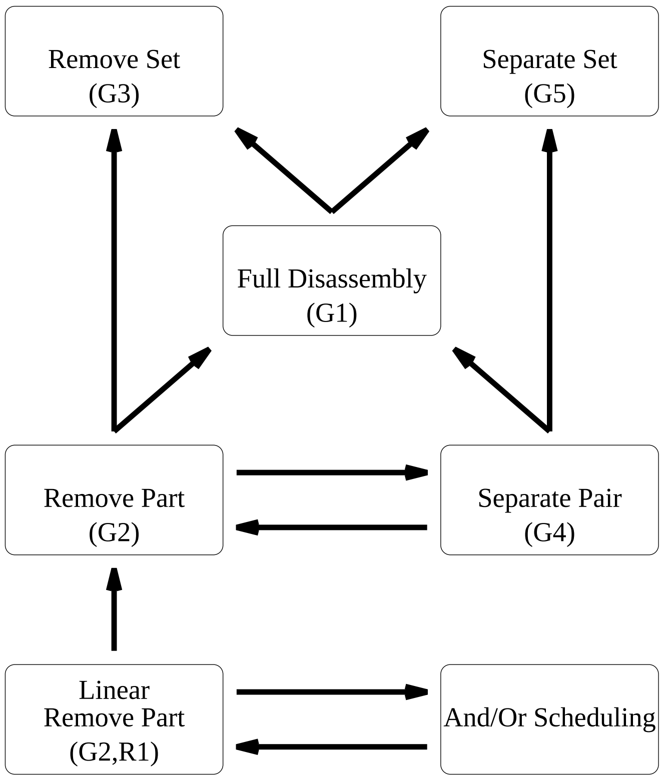


Figure 3: Reductions between variants of VAS

Theorem 6 [G2/R1/C5 $\Longrightarrow$ AND/OR]
Minimizing the number of removed parts for the problem of removing a key part when restricted to linear moves can be reduced, in an approximation-preserving fashion, to the problem of AND/OR *scheduling. (polynomial blowup in size)*

6.1 Proofs

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, this graph is useless for any other operations.

We claim that there is a one-to-one correspondence between solutions of the two instances. Any solution for removing part k in the original problem, can be mimicked in the new problem to separate k and k' from the rest of the problem, and then one final operation using the extra graph can separate k and k'. Similarly, any solution to separate k from k' must end with such a move, and thus the rest of the sequence can be mimicked for the first problem.

The input size for the reduction is increased by one part and one graph, and for all cost measures, the costs of the corresponding solutions differ by at most an additive error of one. ▮

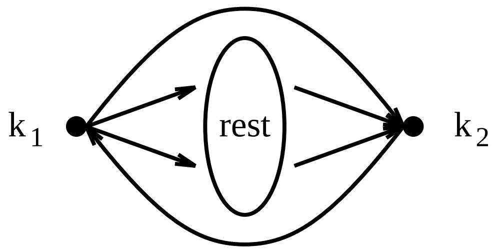


Figure 4: Separating a pair reduces to removing a part

Proof of Theorem 2:
Here, 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 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.

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

Proof of Theorem 3:
Again, we give a very simple modification to translate an instance of the problem of removing a key part k, into an instance of fully disassembling a product. For this reduction, we simply create one additional graph which allows the entire product to fall apart if the key part is missing.

Specifically, we take each graph in $\mathcal{F}$ without modification, and insert one new graph with the edges (k, a) and (a, k) for all other parts a. For all subassembly not containing k, this graph will allow complete disassembly with additional cost one in terms of the number of directions, number of re-orientations or number of non-linear steps. (Notice that for Cost C4, there may be a logarithmic increase in the depth). Furthermore, this graph is strongly connected, and hence of no use, for any subassembly which contains k.

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

Proof of Theorem 4:
For this reduction, we will take an instance which when restricted to linear moves, and we will enforce this linear restriction by converting each graph into n graphs, each which only allows the removal of one part. Given an instance of the problem of removing a given part when restricted to linear steps, we create the following unrestricted instance.

We keep the same set of n parts, and we create a new family of $n|\mathcal{F}|$ graphs. Since $\mathcal{F} = \text{poly}(n)$, then so is $n|\mathcal{F}|$. For each pair (p, d), 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) and (a, p) for each part a if edge $(a, p) \in d$. 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 parts which immediately block p in direction d are exactly those parts which must be removed to disconnect p in our new graph.

With this claim, we immediately get an approximation preserving reduction, because there is a one-to-one correspondence between solutions of the original problem and solutions of the new problem. 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 removed parts, 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, this construction immediately proves that when restricted to linear moves, minimizing the number of removed parts reduces to minimizing the number of directions, the number of re-orientations, or the number of steps. ∎

Proof of Theorem 5:
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 OR-degree of the scheduling problem. By default, each of the graphs is complete, however we will delete the following edges. For an AND-task, t_i, we will modify the first graph by deleting edges (t_i, a) for all $a \notin P_i$. In this way, part t_i can be removed using this graph, if and only if all of its corresponding predecessors have been previously removed. For an OR-task, t_j, with degree Δ, we will modify the first Δ graphs as follows. For each $a \in P_j$, we will modify one of the graphs by deleting all edges (t_j, b) for all $b \neq a$. In this way, part t_j can be removed using this graph so long as part a is priorly removed. Therefore, if any one of the predecessors has been removed, then there will be some graph which allows for the removal of t_j with a linear move. This VAS instance is exactly the original AND/OR scheduling instance, where the number of parts removed is equal to the number of scheduled tasks. ∎

Proof of Theorem 6:
Given an instance of removing a key part with linear steps, we create an instance of AND/OR scheduling as follows. The intuition is that we want to equate the removal of a part with the scheduling of *two* tasks, namely one tasks which says "I am about 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 will create a task, t_p, and for each pair (p, d), with part $p \in \mathcal{P}$ and direction $d \in \mathcal{F}$, we will create task $\hat{t}_{(p,d)}$. We will equate task $\hat{t}_{(p,d)}$ with the statement "I am about to remove part p in direction d," and thus we make it an AND-task with task t_a in its predecessor set for every edge $(p, a) \in d$. We will equate task t_p with the statement, "part p has been removed" and so we make it an OR-task with task $\hat{t}_{(p,d)}$ in its predecessor set for each direction d.

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. In this sense, we get an approximation preserving reduction, in that any t-approximation to the AND/OR scheduling instance gives us a t-approximation to the original VAS instance. However, our construction uses 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. ∎

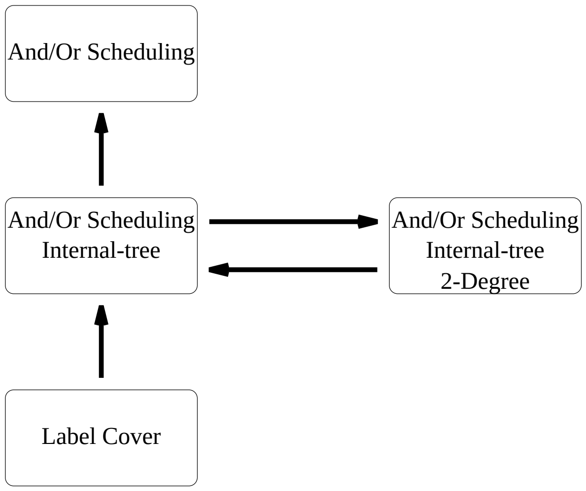

And/Or Scheduling
And/Or Scheduling Internal-tree
And/Or Scheduling Internal-tree 2-Degree
Label Cover

Figure 5: Reductions between variants of AND/OR scheduling

7 Inapproximability of AND/OR Scheduling

We begin by considering the AND/OR scheduling problem when restricted to *internal-tree* precedence constraints, as defined in in Section 5. We will look at the problem where we only charge an algorithm for the *leaves* that it schedules[7]. We show the inapproximability of this problem by showing that the LABELCOVER$_{\text{min}}$ problem[3, 4] is a special case. Following this, we show that the lower bound applies even when we further restrict the AND/OR problem to have degree bounded by two. Finally, we convert this bound on the number of leaves scheduled, into a bound on the total number of scheduled nodes for the general AND/OR scheduling problem. An overview of the reductions is given in Figure 7.

Theorem 7 *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* AND*-degree and* OR*-degree are bounded by two.*

Theorem 8 *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.*

7.1 Proofs

Proof of Theorem 7: The LABELCOVER problem, defined in [4], is an artificial generalization of the SET COVER problem introduced in approximability theory. The input is a regular bipartite graph, $G = (U, V, E)$, a set of labels $\{1, 2, \ldots, N\}$, and a partial function $\Pi_e : \{1, 2, \ldots, N\} \longrightarrow \{1, 2, \ldots, N\}$ for each edge $e \in E$. A labeling associates a non-empty set of labels with every vertex in $U \cup V$. It 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 LABELCOVER$_{\text{min}}$ is to give a labeling which covers all edges, while minimizing the total number of labels assigned to nodes of U.

[7]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.

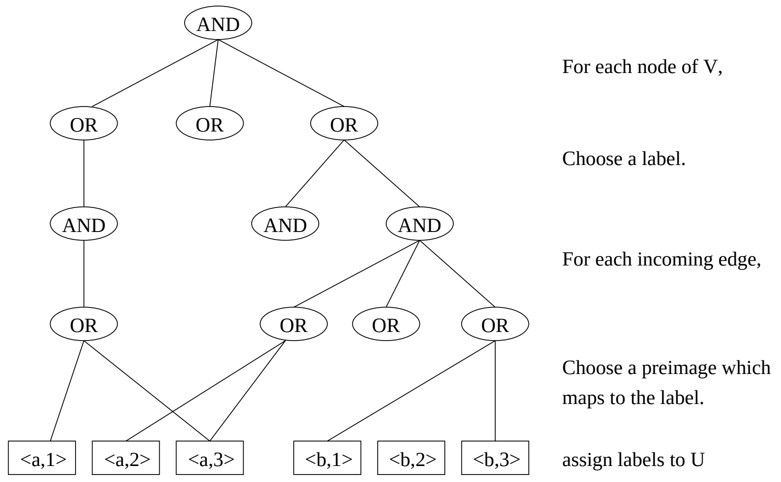


Figure 6: LabelCover as and/or scheduling with internal-tree precedence

Given an instance of LabelCover$_{\text{min}}$, we express it as an instance of and/or scheduling with internal-tree precedence constraints. 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 for a valid labeling, 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 cannot be covered unless one of the pre-images of l' from mapping Π_e is assigned to u.
- The fifth level has a leaf for each pair $\langle u, l \rangle$, and corresponds to label l being assigned to vertex u.

This completes the construction. It can be seen that there is a one-to-one correspondence between valid labeling in the LabelCover$_{\text{min}}$ instance and valid solutions to the and/or scheduling instance. 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 LabelCover$_{\text{min}}$ instance (namely, in $|U|$, $|V|$ and N).

Combining this with the result of [4] which proves a similar lower bound for the approximability of LabelCover$_{\text{min}}$, we get that it is quasi-NP-hard to achieve an $2^{\log^{1-\gamma} n}$ approximation for any $\gamma > 0$.

Given an instance with unbounded degree, we can bound the in-degree in the obvious way, by replacing each internal node with a tree of bounded degree nodes. Assume there were originally I internal nodes and L leaves, and that I is polynomially bounded in L. The maximum fan-in 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 fan-in two. The new instance has $I(I+L)$ internal nodes, which is still polynomially-bounded in L. ∎

Proof of Theorem 8: The only difficulty in switching from the measure of counting scheduled leaves to counting all scheduled nodes is that the overhead of the internal nodes may have a significant cost, changing our approximation ratio. This can be remedied quite easily.

Assume we have a hard instance of internal-tree scheduling from Theorem 7, with I internal nodes and L leaves. We convert this to a general instance of AND/OR scheduling by hanging from each leaf a chain of I new nodes. Notice that the OR-degree is not increased, although this new instance no longer has internal-tree precedence constraints.

In this way, the problematic additive cost can be made arbitrarily small, and so the only issue in completing the proof is to address the input size. In Theorem 7, the value n was assumed to be the number of leaves. In our new instance, the total number of nodes is n', however we can rely on the fact that $n' = \text{poly}(n)$, and thus an approximation ratio of $2^{\log^{1-\gamma} n'}$ is less than a ratio of $2^{\log^{1-\gamma'} n}$ for some $\gamma' > 0$. ∎

8 Inapproximability of Virtual Assemby Sequencing

In this section, we will prove the inapproximability of many variants of the VAS problem. Our first set of results will be a direct consequence of the hardness of AND/OR scheduling shown in the previous section. Our second set of results will show weaker inapproximability results for minimizing the number of re-orientations which apply when the family of motions is restricted to be of constant size (restriction R2). These results come from a natural reduction from the Loading Time Scheduling Problem [8].

Theorem 9 *For the problem of removing a key part, when restricted to linear moves, while minimizing the number of removed parts, it is quasi-NP-hard to achieve a $2^{\log^{1-\gamma} n}$-approximation for any $\gamma > 0$. This result applies even when $|\mathcal{F}| = 2$. (For one graph, this problem is polynomially solvable.)*

Corollary 10 *It is quasi-NP-hard to achieve a $2^{\log^{1-\gamma} n}$-approximation for any $\gamma > 0$, for the following variants of* VAS, *namely for any of the goals, (G1, G2, G3, G4, G5), while minimizing any of the cost measures, (C1, C2, C4, C5), with or without the linear restriction, (R1).*

Theorem 11 *We consider minimizing the number of re-orientations, when the family of motions is restricted to be of constant size, (R2). For any of the goals, (G1, G2, G3, G4, G5), 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 for NP-hard. Both of these facts hold with or without the linear restriction, R1. (for $|\mathcal{F}| = 2$ this problem is trivially solvable.)*

8.1 Proofs

Proof of Theorem 9: This theorem is an immediate result of the hardness of AND/OR scheduling given in Theorem 8, combined with the reduction of Theorem 5. ∎

Proof of Corollary 10: This is a result of Theorem 9, combined with the reductions between variants of VAS given in Theorems 1, 2, 3, and 4. ∎

Proof of Theorem 11: We begin by proving this result for the goal of removing a key part from the assembly. We will give a reduction from the Loading Time Scheduling Problem, defined a follows

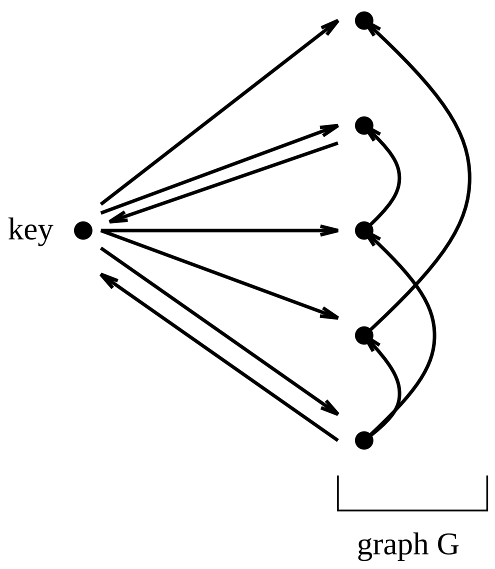


Figure 7: Reduction from Loading Time Scheduling Problem

[8]. There is a set of n jobs, and ρ machines, and each job, j, can only be performed by some subset of the machines, $M(j)$. An algorithm pays for loading a machine, but once that machine is loaded, it may perform any available operations at no additional cost. Finally, the jobs have (standard) precedence constraints, represented by a directed acyclic graph G. The overall cost is the sum of the machine loading times for the sequence. They consider a weighted version where each machine m_i has loading time $l(m_i)$.

We give a reduction from the LTSP problem when all loading times are equal to 1, to the VAS problem of removing a key part (with or without the linear restriction), while minimizing the number of re-orientations. Given an instance of LTSP we create an instance of VAS with a part for each job in the LTSP instance, and one additional part, *key*, whose removal will be our goal. For each machine m, we create a graph $G_m \in \mathcal{F}$. The graph will be a superset of the precedence graph, G, given in the LTSP instance[8], augmented with the edge (key, i) for all parts 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.

We will associate the removal of a part in our problem to the scheduling of a job in the LTSP instance. We claim that our graph G_m allows for the immediate removal of part j by a linear step, if and only if job j can be immediately scheduled on machine m. Assume that job j can currently be scheduled on machine m. In this case it must be that $m \in M(j)$ and that all predecessors of j have already been scheduled. But in this case we claim that vertex j has no outgoing edges in graph G_m, and thus can legally be removed using that graph. Since $m \in M(j)$, then vertex j does not have an edge to *key*, and since all of the predecessors of job j have been previously scheduled, then vertex j does not have any outgoing edges remaining from the original graph G. Similarly, if job j cannot be immediately scheduled, then it must be either because $m \notin M(j)$ or else one of the predecessors of j has not yet been scheduled and cannot be scheduled on this same machine. If $m \notin M(j)$, then both edges (key, j) and (j, key) exist and hence j and *key* cannot be separated. Instead, if one of the predecessors of j, call it b, has not yet been scheduled, than the edges (key, j), (j, b), and (b, key) will exist, and again j cannot be separated from *key*. 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

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

vice versa, and thus we have an approximation preserving reduction. Finally, we note that we need not explicitly require the restriction to linear moves since if our graph allows for a set of parts to be removed at once, then it is also possible to remove them one at a time without re-orienting.

At this point, we rely on results shown in [8], combined with a result of [41], which prove these claims, where the number of machines corresponds to $|\mathcal{F}|$.

Notice that for our construction, the full disassembly problem will be solved exactly as the key part when all parts have been removed from the key part, and so these bounds hold for the full disassembly problem. For the problem of separating a pair, we can use a trick similar to Theorem 1 by modeling the key part as two parts stuck together which can be separated with one additional graph once they are isolated from the other parts. Both of these constructions are valid with or without the restriction to linear steps. ▮

9 Geometric Lower Bounds

It is important to note that the reductions which we gave in our general VAS model do not automatically apply to the original geometric assembly sequencing problems. This includes the reductions proving hardness of problems, and similarly all of the reductions relating the hardness of different variants of our problem. The reason these results do not apply is that a hard instance of the general problem may not be realizable using geometric input. It is possible that by generalizing the original problem, we may have made it much more difficult.

In this section, we realize lower bounds for three different complexity measures. First we consider minimizing the number of directions for any of the five problem goals, in a setting of three-dimensional polyhedral assemblies, when the allowable motions is either infinitesimal or infinite translations. Second, 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 two-dimensional polygonal assembly, showing the inapproximability when restricted to removing one part at a time. Finally, for minimizing the number of parts removed in accessing a key part, 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. As far as we know, this provides the strongest inapproximability results for a simple, combinatorial, geometric problem.

Theorem 12 *We consider minimizng the number of directions used (C1), for a polyhedral assembly in three-dimensions, and either infinitesimal or infinite translations. In this setting, it is NP-hard to minimize the number of distinct directions for any of the goals (G1, G2, G3, G4, G5). This is valid with or without the linear restriction, R1.*

Theorem 13 *We consider minimizing the number of re-orientations used in removing a key part when restricted to linear steps (R1/C2), for a polyhedral assembly in two-dimensions, using either infinitesimal or infinite translations. In this setting we prove, (i) when $|\mathcal{F}| = 3$, minimizing the number of re-orientations is NP-complete; (ii) when $|\mathcal{F}| = 4$, there exists some $c > 0$, such that acheiving a $(1+c)$-approximation is NP-hard; and (iii) there exists some $\delta > 0$, such that achieving a $\log^{\delta} n$-approximation is quasi-NP-hard.*

Theorem 14 *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 by translations to infinity (either individually, or as a group). For this setting, it is quasi-NP-hard to approximate the minimum number of removed disks to 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

Proof of Theorem 12: We begin by considering the goal of removing a key part, and we construct a three-dimensional assembly as follows. For the key part, we create a large flat base rectangle which sits on the bottom of the assembly. Then, the remainder of the parts will be "pegs" which are imbedded far apart into the base piece. The goal will be to isolate the base piece, or remove all the pegs. The key is that we will describe the shape of each peg in a way so that it can only be translated away from the base in a specifc region of the space of directions. As pointed out in [54], this instance looks very much like a Set Cover problem, in that we must chose a minimum number of directions, where each direction allows for the removal of some set of pegs. Unfortunately, it is not possible to realize an arbitrary instance of Set Cover in this way, so the lower bounds for that problem do not apply.

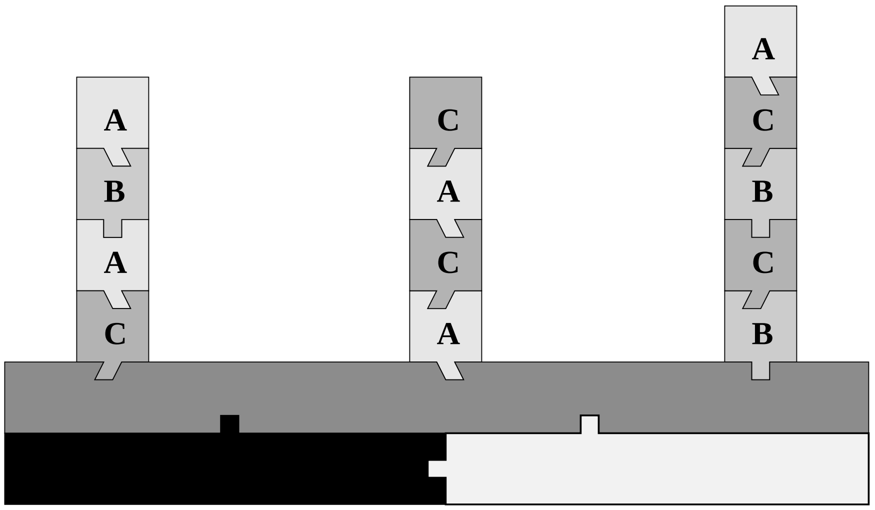
However, we will show that we can realize any instance of a special case of Set Cover which we call Polygon Cover. The problem is the following, given a collection of polygons (possibly degenerate), in the plane, the goal is to pick a minimum number of points so that each polygon contains a point. An even more restricted problem, Rectangle Cover is defined in [42], where all polygons are axis-aligned rectangles, however it is not even known if Rectangle Cover is NP-hard. However, we can prove the NP-completeness of Polygon Cover through a reduction from Planar Vertex Cover, which is known to be NP-complete [18]. Given an instance of Planar Vertex Cover, we can simply let each edge be represented by a (degenerate) polygon. Without loss of generality, in covering the polygons, there is no need to pick a point that is not at a vertex of the graph, and hence this instance is exactly the instance of Planar Vertex Cover.

What remains is to show that any instance of Polygon Cover can be realized using our "pegs" construction. Given a set of polygons, we will think of their spherical projections onto the upper hemispehre. 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, without them interfering with each others removal. Thus we can embed any instance of Polygon Cover and hence we have shown the NP-completeness for removing a given part.

Finally, we note that this exact construction 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 imbeded 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 our cost will be unchanged. ▮

9.2 Finding a Common Supersequence

Proof of Theorem 13: When constrained to using linear moves, we give a construction which reduces the problem of finding a common supersequence [18], to the problem of removing a part from



S = {ABAC, CACA, ACBCB}

Figure 8: Example construction for SCS reduction

an assembly consisting of polygons in two-dimensions. A string T is a supersequence of a string S, if S can be obtained by erasing zero or more symbols of T. Given a finite set of strings over alphabet Σ, a common supersequence is a string T which is a supersequence for each string in the set. Given a set of s strings, with combined length n, over an alphabet of size $|\Sigma| = k$, we build the following instance of removing a part from an assembly. The key part is a long flat base rectangle, and each sequence will be represented as a tower of "square" blocks stacked on the base, with the towers spaced sufficiently 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 into the piece below will restrict the separation to a single direction of motion. The exact direction of motion for each block will be chosen according to the alphabet symbol represented by the block, and each alphabet symbol will be given a unique direction. All of the directions will be chosen to lie in a sufficiently small cone so as to prevent the individual towers from interfering with each other. A (modified) example is given in Figure 8.

If we assume that none of the input sequences contain consecutive occurences of the same character, then we claim that any solution for removing the key part provides us with a supersequence whose length is equal to the number of re-orientations, and vice versa. If the supersequence input does have consecutive occurences of the same character, we can remedy this at the cost of doubling the size of the alphabet by replacing each occurence of character a by the sequence $a_1 a_2$.

The problem of finding the shortest common supersequence is known to be NP-hard [18], and more recently it was shown to be Max-SNP-hard, even over a binary alphabet [9]. Therefore, by doubling the alphabet as above, we get that our problem of removing a part is Max-SNP-hard, when $|\mathcal{F}| \geq 4$. Also, it was shown in [30] that there exists a constant $\delta > 0$, such that it is quasi-NP-hard to approximate the shortest common supersequence to within a factor of $log^{\delta} n$. Finally, even if strings have consecutive occurences of the same symbol, it was shown that for an alphabet of size $|\Sigma| = 3$, that finding a common supersequence with the minimum number of *runs* is NP-complete [41]. A *run* is defined as a group of consecutive occurences 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 problem of

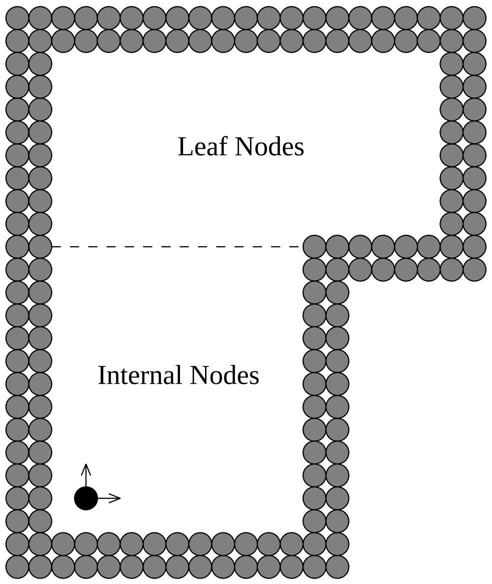


Figure 9: Overview of DISKS construction

removing a key part.

Again, we can apply this construction to prove the identical result about the other possible goals. In our original construction, we note that as soon as our key part is separated from all of the parts, then we have completed the full disassembly problem. For the problem of separating a pair of parts, we must introduce two extra parts which are to be separated. These two parts are placed along the bottom of the original base, and pegs are used as shown in Figure 8 to stick the parts to each other and to the base. Because we are restricted to using linear operations, the only way these two parts can be separated is by first moving the base upwards away from them, and this in turn can only be done after all of the towers are removed from the base. This completes our proof. ▮

9.3 The DISKS Problem

Proof of Theorem 14: Our proof is based on a reduction 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 7, 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, as needed, 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 9.

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 give each such disk an "escape route" to the North by creating a hole in the above wall. For an OR-disk, we create an additional passage to the East.

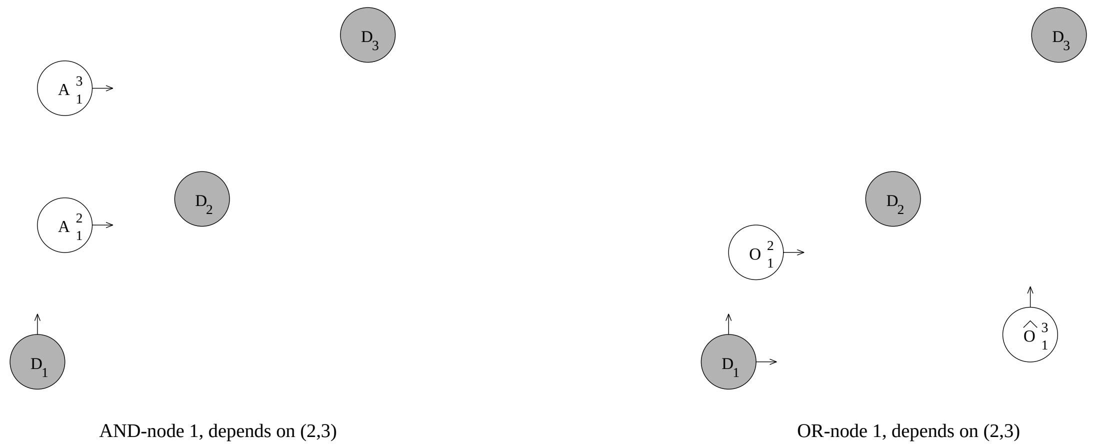


Figure 10: Internal node mechanisms

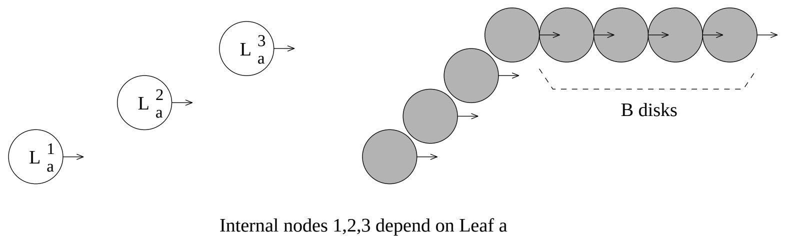


Figure 11: Leaf node mechanism

Finally, 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 $(6i+1, 6k-1)$, and force this disk to exit to the East. 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 we force East, and $\hat{O}_i^l$ located at $(6l-1, 6i+1)$ which we force North. The entire internal node mechanisms are contained in a $(6I+1) \times (6I+1)$ square. Examples are given in Figure 10.

The section for the leaf mechanisms begins at height $6(I+1)$ so as to be higher than the internal mechanisms. We can number the leaf nodes in any order, and we create a separate mechanism for each leaf in a strip of height $2I$. For a given leaf, L_a, we create what we term a *blockade*, to the right of this strip. The blockade consists first of a diagonal chain of to the Northeast of height $2I$, followed by a horizontal chain of B disks to the East of the end of the first chain (where B is determined later). The disk beginning the blockade is centered at $(6(I+1), 6(I+1)+Ia)$. The wall to the East of the blockade is removed, allowing the disks of the blockade an escape. For any disk located in the horizontal strip associated with L_a, escaping to the East will require an additional cost of at least B to break through the blockade. However this cost is only charged once per blockade, after which any disks in the horizontal strip may escape. Now, for every internal node T_i which depends on leaf L_a, we create a disk L_a^i, located at $(6i+1, 6(I+1)+Ia+2i)$, which we force East. Figure 11 shows an example of a leaf mechanism.

To complete the construction, we set the blockade value, $B = 4I(L+I)$, to be greater than the total number of disks in the remainder of the internal and leaf mechanisms combined. In this way, the number of blockades removed dominates any additive costs in the rest of the construction. Finally, we assign $W = B(L+1)$, so that the cost of removing all non-wall disks is less than the cost of digging a single new hole through any part of the wall. For this reason, we may assume without

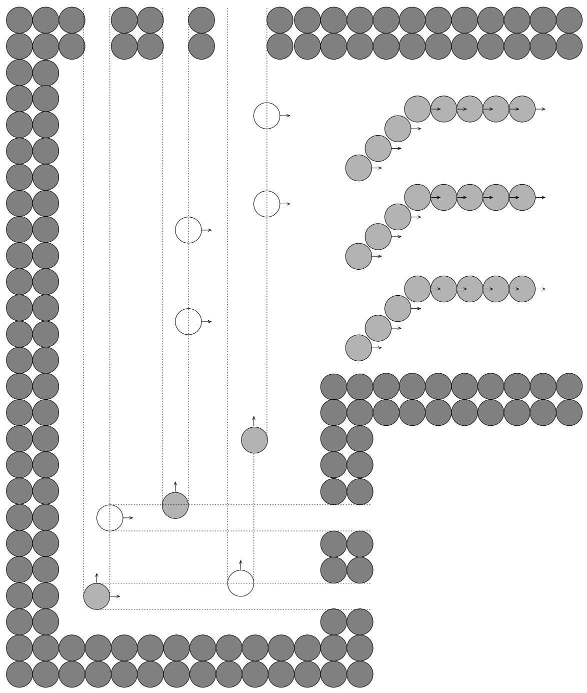


Figure 12: A complete DISKS construction

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 12.

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. In fact, if we allow translations in arbitrary directions, the theorem holds using this same construction. Furthermore, there is no need to force a restriction to linear moves, since moves which remove a group of disks at once could be replaced by a set of linear moves.

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 wall can be extended to block any useful motions in other dimensions, while still using polynomially many disks. ▮

10 Implications of 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 extention to the standard scheduling problem, yet clearly the effect of this change on the difficulty of the problem is quite dramatic. We pose a series of open directons of research related to the theory of approximability and where this problem fits in relation to several other problems.

In Section 7, we consider several versions of this scheduling problem, giving reductions from one to another, and then proved a lower bound of $2^{\log^{1-\gamma} n}$ against the approximability of all of these problems by showing that the easiest of these versions captures the LabelCover$_{\min}$ 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 LabelCover$_{\min}$ 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 LabelCover is truly n^{ϵ}-hard to approximate [4], however it may be possible to strengthen the lower bounds for AND/OR scheduling without necessarily settling the LabelCover conjecture. Furthermore, reasoning about instances of AND/OR scheduling seems to be a bit more intuitive then about instances of a problem such as LabelCover.

10.1 Alternating Levels of Internal-Tree Precedence Constraints

We examined a very structured class of instances of AND/OR scheduling which had what we termed *internal-tree* precedence constraints, and we considered charging only for the *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 in-degree of the internal nodes, we can 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, that is 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, and hence lies in Class II. The AND-node requires that we cover each item in the universe, and each OR-node requires that for the given item, we pick one of the sets which covers that item. Each leaf corresponds to a ground set, and thus the number of leaves scheduled is equal to the number of sets used to cover the universe. If we look again at Figure 6, we see that as soon as we allow two full levels of alternations, this problem captures the LabelCover$_{\min}$ problem, and hence is in Class III. However it is not at all clear that this problem is equivalent to LabelCover as we do not know whether an instance of this restricted AND/OR scheduling can be translated into a LabelCover 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 no constraints are placed on the structure of the precedence graph?

The answer for some of these questions may come from research in the study of monotone boolean formulae. As we mentioned earlier, the internal-tree precedences exactly defines a monotone boolean function on the leaves, where the goal is to satisfy the function using the minimum number of ones. 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 which 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. There is a wealth of research related to monotone formulae and circuits in this respect [36, 46, 56], however it is open to strengthen any of our inapproximability results for AND/OR scheduling.

11 Conclusions & Open Problems

We explain the lack of progress in finding optimal or near-optimal assembly sequences by formally proving the inapproximability for minimzing 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 most of the cost measures we consider. As a special case, we prove similar hardness results for the problem of scheduling with AND/OR precedence constraints. Finally, as our graph-theoretic problem is a generalization, we prove hardness results for several complexity measures in simple geometric settings. For minimizing the number of parts which must be removed to access a key part, we match our strongest inapproximability results, even for a setting consisting entirely of unit disks in the plane, while using simple translations to infinity to remove parts. For minimizing the number of directions used or the number of re-orientations, our geometric lower bounds are far weaker than their graph-theoretic counterparts.

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

The over whelming 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 postive results in this model would immediately apply to all of these geometric settings. Our lower bounds show that success in this model is limited, however 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 manufacture, 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, that 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 below the geneeral 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.

Improving any of our lower bounds, or providing non-trivial upper bounds remains open for either our general problem, or any of the geometric settings we consider. Furthermore, this work intrduces several interesting questions regarding the theory of approximation. The graph-theoretic generalization of assembly sequencing which we introduce captures a variety of previously studied problems, in a fairly intuitive manner. Better understanding the place of these problems in the approximation hierarchy would be helpful. Specifically for the special case of AND/OR scheduling, many of these issues were discussed in Section 10.

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. Finally, our appreciation goes to Chandra Chekuri and Sanjeev Khanna for their participation in the study of scheduling with AND/OR precedence constraints.

References

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

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

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

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

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

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

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

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

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

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

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

[12] 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.

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

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

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

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

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

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

[19] D. Gillies. *Algorithms to schedule tasks with AND/OR precedence constraints.* Ph.D. thesis, University of Illinois, Urbana, IL, 1991.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[51] R. Wilson, L. Kavraki, and T. Lozano-Pérez. Two-handed assembly sequencing. Stanford Technical Report STAN-CS-93-1478, 1993.

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

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

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

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

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