Complexity Measures for Assembly Sequences[*]

Michael Goldwasser
wass@cs.stanford.edu

Jean-Claude Latombe
latombe@cs.stanford.edu

Rajeev Motwani
rajeev@cs.stanford.edu

Abstract

Our work examines various complexity measures for two-handed assembly sequences. Many present assembly sequencers take a description of a product and output a valid assembly sequence. For many products there exists an exponentially large set of valid sequences, and a natural goal is to use automated systems to attempt 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 plan is of great value when it comes time to assemble the physical product in mass quantities.

We take a step in this direction by introducing a formal framework for studying the optimization of several complexity measures. This framework focuses on the combinatorial aspect of the family of valid assembly sequences, while temporarily separating out the specific geometric assumptions inherent to the problem. With an exponential number of possibilities, finding the true optimal cost solution seems hard. In fact in the most general case, our results suggest that even finding an approximate solution is hard. Future work is directed towards using this model to study how the original geometric assumptions can be reintroduced to prove stronger approximation results.

1 Introduction

In this paper we study issues of product complexity in the *assembly sequencing* problem. In general terms, the input to an assembly sequencer is a *product*, described by a geometric model of its *parts* as well as their relative positions, and a family of allowable *motions*. The output is a sequence of operations resulting in the construction of the product from its individual parts. Each operation combines a set of subassemblies using a motion from the allowable set.

The use of automation in assembly sequencing has increased rapidly over the years [1, 4, 8, 9, 10, 13, 14, 21, 22, 23, 24]. 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 then testing their feasibility [10, 23]. Theoretical results show that assembly sequencing, in its most general form, is intractable [11, 12, 17, 24]. This fact 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 of these restricted classes, polynomial algorithms were designed which find an assembly sequence if one exists [20, 21]. There are also algorithms which can enumerate all possible assembly sequences [4], however there may be exponentially many such sequences for a given product. A logical continuation is to use automated reasoning to find the "best" assembly sequence under a certain complexity measure.

Several researchers have acknowledged the need to extend automated reasoning to search for better assembly sequences, however results have been limited. Several empirical measures have been suggested in [2]. The system in [24] considers several measures in a simplified model for evaluating the quality of an assembly sequence. For a restricted class of inputs, which have a "total ordering" property, an algorithm is given which produces the minimal length sequence to remove any given part [25]. Using a hierarchical approach to identify common subassemblies in products reduces the computational complexity required for many large products, allowing more effort to be used towards finding a "better" assembly sequence [3]. Although practical, this technique simply delays the eventual need for

*Department of Computer Science, Stanford University, Stanford, CA 94305. Research supported by a grant from the Stanford Integrated Manufacturing Association (SIMA), NSF/ARPA Grant IRI-9306544, and by NSF Grant CCR-9215219. In addition, the third author is supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership Award, and by NSF Young Investigator Award CCR-9357849 with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

better automated reasoning to overcome increasingly large data sets.

A more complete discussion on using automated reasoning to evaluate the complexity of assembly sequences is given by [20], where they suggest several complexity measures including the number of hands used, the length of the longest sequence of operations, and the number of degrees of freedom required. They prove decidability for some simple questions such as, "can a product be assembled entirely from one direction?" Many of these same complexity measures are looked at by the STAAT assembly sequencer, however to optimize over these measures it must either perform an expensive brute force search, or settle for simple heuristics [19].

We attempt to formalize the task of optimizing assembly sequencing for several complexity measures motivated by industrial applications, by introducing a theoretical model generalizing several variants of the assembly sequencing problem. Many of these optimization problem turn out to be NP-complete, and so we approach them using techniques common to the theory of approximability [5, 16].

Section 2 discusses the polynomial time assembly sequencers that motivate our framework. The model is defined in Section 3. Some preliminary results are given in Section 4. Section 5 briefly describes experimental results. Finally, Section 6 contains conclusions and open problems.

2 Background on Decidability

A common approach to devising an assembly plan is to construct a *disassembly* plan, and then to reverse the entire plan. Although in practice these two tasks are not always symmetric, we work under the assumption that they are, and thus we will freely interchange the concepts of assembling and disassembling.

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. The *assembly plan* can be represented naturally as a binary tree. For a more detailed discussion, see [20, 21, 24]. Figure 1 gives an example, taken from [20], of an assembly tree for a simple 2-dimensional product.

Our work build upon the notion of the *non-directional blocking graph (*NDBG*)* [21]. Given any single motion d, a *directional blocking graph (*DBG*)* is defined as follows. A DBG is a directed graph G

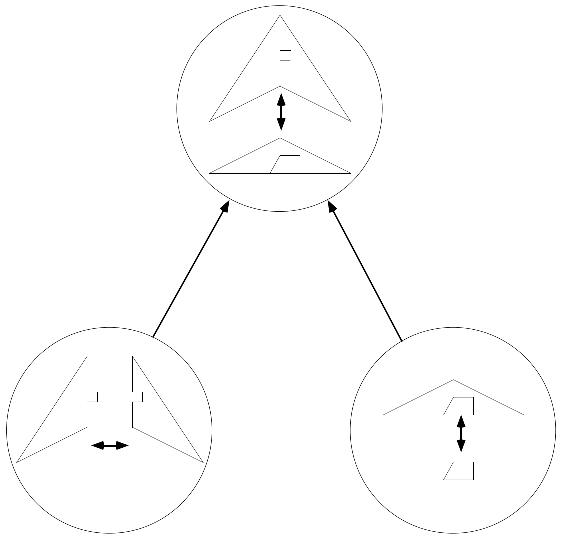

Figure 1: Assembly Tree for a simple product

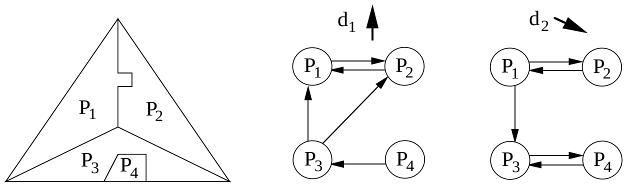

Figure 2: A simple assembly and two DBGs

with a node for each part of the assembly, and an edge $A \rightarrow B$, if A collides with B when the motion d is applied to part A while part B remains stationary. Figure 2, also from [20], gives an example of a 2D product as well as two directional blocking graphs for infinitesimal translation. Notice that a directed cut between subset S and subset T in a DBG represents a collision-free separation.

The construction of the NDBG produces a **polynomially-sized** set of candidate motions, which are representatives for equivalence classes of motions having identical directional blocking graphs. For a given class of motions, this set of blocking graphs completely captures the necessary geometric information for identifying all valid assembly sequences.

Computational geometry techniques allow for the construction of the NDBG for a wide range of motion classes, including infinitesimal translations [20], extended translations (i.e., to infinity) [20], multiple step translations [7], and infinitesimal generalized motions (i.e., rigid body motions) [6, 20].

For each of these families of motion, the NDBG framework immediately provides a polynomial time algorithm for constructing an assembly sequence. After constructing the set of DBG's, an arbitrary assembly sequence can be found by taking any legal separation in any direction, and recursing on the resulting subassemblies. However searching for a "good" sequence is not so simple.

3 A Framework for Analyzing Complexity Measures

Here we define an optimization problem which generalizes this set of assembly sequencing problems. The motivation is that the set of blocking graphs completely characterizes the spatial relationships between the parts, and so we focus solely on that part of the problem. We define the following model which we will call the *Set Decomposition* problem.

INPUT: An abstract set, $\mathcal{P}$, of n "parts", and a polynomial sized family, $\mathcal{F}$, of directed graphs on n nodes. We will call each member of the family a "direction."

OUTPUT: An assembly tree for $\mathcal{P}$ using only directions from $\mathcal{F}$.

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').

Clearly this model is more general then an instance of assembly sequencing. Given any of the original assembly sequence instances, by calculating the DBG's and working from there, we get an instance of the set decomposition problem. However, in the new model we assume nothing about the properties of the individual graphs or their interdependence, whereas an instance coming from an original assembly may have structure due to the underlying geometry.

Therefore, any positive results on the set decomposition problem will immediately yield results for all versions of the assembly sequencing problems which can be converted to DBG's. Negative results on this model do not automatically carry over to the assembly sequencing problem, however such results may highlight additional structure in the original problem which can be utilized for better approximations.

3.1 Possible Tasks

Originally, we said that the goal of an assembly sequencer is to produce an entire assembly tree that completely decomposes the original product into its individual parts. This is indeed a common task, however there are other variants which are highly motivated by industrial applications.

Remove a key part

Instead of disassembling the entire product, it is often desirable to quickly remove a single key part from an assembly without necessarily disassembling the entire product. The motivation for this variant stems from problems of maintenance and of 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. The assembly tree returned is allowed to have leaf nodes which are subassemblies rather than single parts, so long as the key part is isolated at some leaf. The compacted assembly tree will be a simple path from the root down to the key leaf.

Remove a given set of parts

A task that is possibly more difficult than removing a single part would be to remove an arbitrary subset of parts. That is, the goal is to start with the fully assembled product and reach a state where each part in the given set is either isolated or grouped only with other parts from this set.

Break a given contact

Another possible variant that has been suggested is the following. Given a product and a key contact between two parts, the goal is to get those two parts into separate subassemblies. (Note: the two parts need not be removed from the entire assembly, just separated into components that do not include the other key part.)

3.2 Possible Complexity Measures

At this point, we begin to look at measures for deciding which of two assembly plans is the better one for a given product. Of course, every client asked will give a different definition of what they consider better. Even in the same application the exact cost measure may depend on the current stage of the design and on manufacturing concerns not even known at the time. Each section below introduces a "primitive" complexity measure, well motivated by specific aspects of industrial applications. Our hope is that studying these primitive measures in depth can eventually lead to systems which will be able to specialize complexity measures for custom purposes. Many of these measures are generalizations of ideas introduced in [20].

Fewest Number of Directions

The cost of an assembly tree is equal to the number of directions in $\mathcal{F}$ which are used. Note that 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.

Fewest Re-orientations

Here the output is considered to be not only an assembly tree, but also a sequential ordering of the steps consistent with the assembly tree (i.e., a linear extension of the partial order). The cost of a sequence is equal to the number of re-orientations necessary while performing the sequence [24]. In some 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. For instance, many robots perform operations from a vertical direction, in which case using a different motion direction corresponds to re-orienting the subassembly on the assembly line. This is typically slow and might require additional expensive fixtures. Also, using an orientation that was encountered earlier in the process is not any more efficient if the product is not still in that orientation.

Fewest Number of Non-Linear Steps

A linear step is a step where one of the two subassemblies is a single part. The cost of an assembly tree is equal to the number of non-linear steps. The motivation here is that at times parallelism is costly. In theory, every step is based on two recursive subproblems which come together during this step. In a linear step, one of the subproblems is a single part, and so there is really only one subproblem on which to recurse.

Minimum Depth of an Assembly Sequence

The cost of an assembly tree is equal to the depth of the tree. The motivation here is that in many assembly environments, parallelism in production is helpful, and so the minimum depth tree has the quickest "throughput." Also, for the key part problem, this cost is equal to the depth of the *key leaf*. This corresponds to the number of steps that must be taken to free a key part from the rest of the product.

Maximum Depth of an Assembly Sequence

Again, the cost of an assembly tree is equal to the depth of the tree. The motivation here is again from the case where parallelism is undesirable. This captures a different measure then the number of non-linear steps, however it is not clear which is a more accurate measure for practical manufacturing, or if they both have merit.

4 Preliminary Results

We look at set decomposition as a classical optimization problem. For a given complexity measure from Section 3.2, the ideal goal for an algorithm would be to find the true optimal cost solution. However if this is not possible, the next goal would be to find another solution and to *guarantee* that the cost of this solution is not too far away from the cost of the optimal solution. A standard measure for the quality of an approximation algorithm is the *performance ratio* between the cost of the solution returned by the algorithm versus the cost of the optimal solutions. This field of *approximability theory* has been well researched in classical computer science [5, 16]; we examine the set decomposition problem in a similar manner. The importance of this type of analysis over studying purely experimental heuristics is to gain a better understanding of the quality of the approximations and the asymptotic behavior as the input size increases. As products become more complex and more densely packed, such analysis could grow in importance.

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* [18, 16]. 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. In fact, some NP-complete problems can be approximated very well in polynomial time, whereas for other NP-complete problems, it is NP-hard to even find an approximate solution. So 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.

We will concentrate specifically on analyzing the cost measure of fewest re-orientations, however most of the following results hold for all complexity measures listed in Section 3.2. Using approximation-preserving reductions, we prove that the task of fully decomposing an assembly is at least as hard as removing a given keypart. Then we claim that removing a keypart in this model is at least as hard as the SetCover problem (defined in [5]). We rely on a result of [15] claiming the hardness of approximating SetCover. Finally, we use a self-amplification technique to further strengthen the hardness results of approximating the set decomposition problem.

Theorem 1 *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 while using the fewest number of re-orientations.*

The intuition of the reduction is that we will take an instance problem of removing a key part k, and we will construct an instance problem of separating two parts by breaking k into two distinct parts k_1 and k_2 which are interlocked unless they have been completely separated from all other parts.

We modify each graph in $\mathcal{F}$ by breaking up part k and adding edges (k_1, k_2) and (k_2, k_1). Finally we add in one new graph which is the complete graph with the two edges (k_1, k_2) and (k_2, k_1) removed. Notice that this graph will allow parts k_1 and k_2 to separate only if they are the only two parts in a subassembly, and this graph is the only one which will ever allow these parts to be separated. Additionally, for all other subassemblies, this graph will be strongly connected and thus will not provide any legal separations.

Similarly, this reduction can be done in the opposite direction using different techniques, and therefore the key part and key pair tasks are essential identical in this framework.

Theorem 2 *The problem of separating a key pair of parts from each other can be reduced, in an approximation-preserving fashion, to the problem of fully decomposing the product while using the fewest number of re-orientations.*

The intuition of the reduction is that we will take an instance problem of separating parts i and j, and we construct a full decomposition problem by adding one extra directional graph which will allow the entire product to fall apart if either i or j is missing, but will allow nothing on any subassembly with both i and j. This simulates both the fact that we don't care about what happens to subassemblies that do not include either part, and the fact that once i and j have been separated from each other, the problem is essentially solved. The newly created graph allowing this behavior is shown in Figure 3.

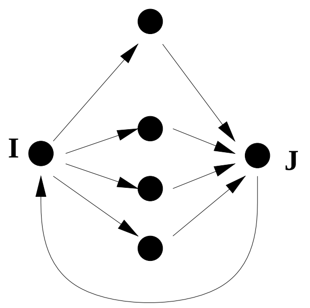

Figure 3: Separation of pair reduces to full decomposition

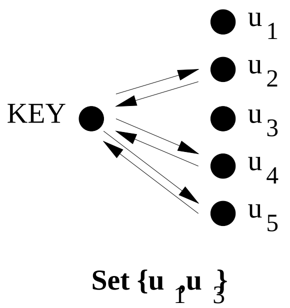

Figure 4: Example construction for SetCover reduction

Theorem 3 *The SetCover problem can be reduced, in an approximation-preserving fashion, to the problem of removing a single part from the rest of the subassembly while using the fewest number of re-orientations.*

Proof: We use the following notation for Set Cover. The input is a collection $\mathcal{S}$ of sets $S_1, S_2, \ldots, S_m$ over a universe of items U, with $|U| = n$. The goal is to pick the minimum cardinality subcollection $\mathcal{S}' \subset \mathcal{S}$ such that each element $u \in U$ is in at least one set $S_i \in \mathcal{S}'$.

Given an instance of the Set Cover problem, we create an instance of the problem of removing a key part as follows. For each item in the universe U, we create a part, and in addition we add one extra part, KEY. Removing KEY will be the goal.

For each set S_i in the collection, we create a graph in the family $\mathcal{F}$. By default, this graph is a star graph centered on KEY, however we delete the edges between KEY and any part j such that u_j is in the set S_i. See Figure 4 for an example.

Notice that for KEY to be completely removed from the rest of the assembly, it must be separated from each part u_j. And so for KEY to be removed completely, then for each u_j, there must be some direction chosen which corresponds to a set S_i which contains u_j.

We claim that any set of directions which admit a valid assembly sequence can be translated directly to a solution to the SetCover problem. Similarly, any solution to the SetCover problem can be translated to a valid assembly sequence. Both of these translations preserve the cardinality of the solutions. ∎

Lemma 4 *[15] There is no polynomial time approximation algorithm for Set Cover with ratio $c \log n$ unless $\tilde{P} = N\tilde{P}$, for any $0 < c < 1/4$.*

The classes $\tilde{P}$ and $N\tilde{P}$ represent the analog of the usual complexity classes with quasi-polynomial bounds. It is also widely believed that $\tilde{P} \neq N\tilde{P}$, and thus this lemma combined with Theorem 3, provide evidence towards the difficulty of accurately approximating the set decomposition problem. It is important to note that the reduction we give from *SetCover* to this model does not seem to be realizable in the original assembly sequencing problem. Rather, for the original problem we can realize a reduction from Rectangle Cover, a geometric version of the set cover problem, which is conjectured to be as hard as *SetCover* [16].

In any event, the set decomposition problem appears to be even more complex that the well understood *SetCover* problem, as is brought out by the following theorem.

Theorem 5 *If there exists a polynomial approximation algorithm for removing a key part with minimum number of re-orientations which achieves a ratio of $c \log n$, then there exists a set-cover approximation with ratio $\sqrt{c(\log n + \log |\mathcal{F}|)}$*

Although we do not give the full proof, the additional strength results from the following amplification of our original SetCover reduction. Originally, selecting a set S_i in the cover corresponded to using an additional *DBG* in the decomposition problem, thus incresing the cost by one unit for both problems. We can instead using a "locking" device which will force an algorithm so solve a complicated recursive version of the decomposition problem to simulate selecting a set for the SetCover problem. This penalty amplifies the cost of mistakenly choosing too many sets in the *SetCover* solution.

5 Experimental Results

Although there exists the hardness result for *SetCover* in Lemma 4, there exists a simple greedy algorithm which achieves the given performance ratio. It would be nice to prove similar results about greedy algorithms for the decomposition problem. For the problem of removing a key part using as few steps as possible, we considered greedy heuristics such as removing as many parts as possible in each step. Unfortunately many greedy heuristics are easily defeated in the worst case by counterexamples attaining poor performance ratios.

We implemented such heuristics and the experimental results show that the pitfalls causing poor performance are not isolated examples, rather are quite common. We used data sets obtained from products run through the STAAT assembly sequencer [19]. Larger data sets were generated randomly, modeling pseudo-assemblies.

6 Conclusions

A great deal of research has lead to the advancement of automated assembly sequencers, and their promise towards industrial use. However, it seems that the quality and the cost effectiveness of the sequences should be considered before using the assembly plans in manufacturing. This framework takes a first step at providing a theoretical basis for analyzing the optimization of assembly sequencing.

Together, our results show that in this general model, there is little hope of finding an algorithm which can reasonably approximate the optimal solution for any of these cost measures. However the reductions we give are not realized geometrically. The question is whether the original geometric version of this problem is equally as hard, or whether the set of blocking graphs that result from actual assemblies have additional properties which can be utilized algorithmically.

Recall, each cell of an NDBG represents a subset of the directional space, within which the blocking relationships are constant. The border between two cells is caused by two parts whose blocking relationship reaches a sliding contact, and therefore the blocking graphs between any two neighboring cells in an NDBG differ by at most one edge. Although it is not clear how to take advantage of this specific structure, our hope is that properties such as this may provide additional strength for better approximations.

Acknowledgements

The authors wish to thank Leo Guibas, Danny Halperinn and Randy Wilson for many conversations during the formation of this model, and to Sampath Kannan for many helpful discussions and suggestions reguarding the reductions between versions of the problem.

References

[1] D. F. Baldwin. Algorithmic methods and software tools for the generation of mechanical assembly sequences. M.Sc. thesis, Massachusetts Inst. Tech., Cambridge, MA, 1990.

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

[3] S. Chakrabarty and J. Wolter. A hierarchical approach to assembly planning. In *IEEE Inter. Conf. of Robotics and Automation*, pages 258–263, 1994.

[4] T. L. De Fazio and D.E. Whitney. Simplified generation of all mechanical assembly sequences. *IEEE Inter. Journal of Robotics and Automation*, 3(6):640–658, 1987.

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

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

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

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

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

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

[11] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: P-space hardness of the "Warehouseman's Problem". *Internat. J. Robot. Res.*, 3(4):76–88, 1984.

[12] 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.

[13] S. S. Krishnan and A. C. Sanderson. Path planning algorithms for assembly sequence planning. In *IEEE Inter. Conf. on Intelligent Robotics*, pages 428–439, 1991.

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

[15] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proc. 25th Annu. ACM Sympos. Theory Comput. (STOC 93)*, pages 286–293, 1993.

[16] R. Motwani. Approximation algorithms. Tech. Report STAN-CS-92-1435, Dept. Comput. Sci., Stanford Univ., Stanford, CA, 1992.

[17] B. K. Natarajan. On planning assemblies. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 299–308, 1988.

[18] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Systemm Sci.*, 43:425–440, 1991.

[19] B. Romney, C. Godard, M. Goldwasser, and G. Ramkumar. An efficient system for geometric assembly sequence generation and evaluation. In *Proceedings of the ASME International Computers in Engineering Conference*, page To appear, 1995.

[20] R. Wilson and J.-C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(1), 1995.

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

[22] R. H. Wilson, L. Kavraki, and T. Lozano-Pérez. Two-handed assembly sequencing. Tech. Report STAN-CS-93-1478, Dept. Comput. Sci., Stanford Univ., Stanford, CA, June 1993.

[23] R. H. Wilson and J. F. Rit. Maintaining geometric dependencies in and assembly planner. In *IEEE Inter. Conf. of Robotics and Automation*, pages 890–895, 1990.

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

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