SAND98-2761C

An Optimizing Algorithm for Automating Lifecycle Assembly Processes*

Terri L. Calton
Intelligent Systems and Robotics Center
Sandia National Laboratories
Albuquerque, NM 87185-1008
tlcalto@sandia.gov
Phone: 505-845-7949
Fax : 505-284-4591

Russell G. Brown
Intelligent Systems and Robotics Center
Sandia National Laboratories
Albuquerque, NM 87185-1008
rgbrown@sandia.gov
Phone: 505-844-5596
Fax: 505-284-4591

Abstract

Designing products for easy assembly and disassembly during its entire lifecycle for purposes including service, field repair, upgrade, and disposal is a process that involves many disciplines. In addition, finding the best solution often involves considering the design as a whole and by considering its intended lifecycle. Different goals and constraints (compared to initial assembly) require us to re-visit the significant fundamental assumptions and methods that underlie current assembly planning techniques. Previous work in this area has been limited to either academic studies of assembly planning or applied studies of lifecycle assembly processes, which give no attention to automatic planning. It is believed that merging these two areas will result in a much greater ability to design for, analyze, and optimize the disassembly and assembly processes.

Summary

Many manufacturing companies today expend more effort on upgrade and disposal projects than on clean-slate design, and this trend is expected to become more prevalent in coming years. However, commercial CAD tools are better suited to initial product design than to the product's full lifecycle. Computer-aided analysis, optimization, and visualization of lifecycle assembly processes based on the product CAD data can help ensure accuracy and reduce effort expended in planning these processes for existing products, as well as provide design-for-lifecycle analysis for new designs.

In this paper we attempt to find a balance between the academic studies of assembly planning and applied studies of lifecycle assembly processes while maintaining a high level of automated planning capabilities. We briefly examined some of the issues in lifecycle engineering which are currently considered critical and used these issues to motivate the current direction in which our automated assembly planning tool, Archimedes, is developed. We summarized the existing capabilities of our system and described recent developments within system that are intended to enable its use in a more general lifecycle framework by allowing users to compute disassembly plans tailored to specific service, upgrade, and/or dismantlement strategies. Specifically, we described a "shortening algorithm" which makes it much easier for Archimedes to find shortest or least-cost disassembly sequences to gain access to specific parts and subassemblies. In practice the shortening algorithm provides an effective way to find the lowest-cost servicing disassembly plan. It allows replacing one or several subassemblies, gaining access for inspections, and optimizing disassembly to minimize service-related costs. The user can specify the set of parts for the service operation, enable the optimizer, start the planner, and let it run until the cost of the best plan so far stays the same for several iterations. At that point the plan will, with high probability, be the one desired. If not, the user can either rerun the planner for a longer duration, or add constraints to help the planner find the minimum cost service plan.

* Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

DISCLAIMER

An Optimizing Algorithm for Automating Lifecycle Assembly Processes*

Terri L. Calton and Russell G. Brown

Intelligent Systems and Robotics Center
Sandia National Laboratories
Albuquerque, NM 87185-1008

Abstract

Designing products for easy assembly and disassembly during its entire lifecycle for purposes including service, field repair, upgrade, and disposal is a process that involves many disciplines. In addition, finding the best solution often involves considering the design as a whole and by considering its intended lifecycle. Different goals and constraints (compared to initial assembly) require us to re-visit the significant fundamental assumptions and methods that underlie current assembly planning techniques. Previous work in this area has been limited to either academic studies of assembly planning or applied studies of lifecycle assembly processes, which give no attention to automatic planning. It is believed that merging these two areas will result in a much greater ability to design for, analyze, and optimize the disassembly and assembly processes.

1 Introduction

Many manufacturing companies today expend more effort on upgrade and disposal projects than on clean-slate design, and this trend is expected to become more prevalent in coming years. However, commercial CAD tools are better suited to initial product design than to the product's full lifecycle. Computer-aided analysis, optimization, and visualization of lifecycle assembly processes based on the product CAD data can help ensure accuracy and reduce effort expended in planning these processes for existing products, as well as provide design-for-lifecycle analysis for new designs. In this paper we attempt to find a balance between the academic studies of assembly planning and applied studies of lifecycle assembly processes while maintaining a high level of automated planning capabilities.

To be effective, computer-aided assembly planning systems must allow users to express the plan selection criteria that apply to their companies and products as well as to the lifecycles of their products. In addition, the planning system must provide efficient and easy-to-use optimization features to minimize assembly and disassembly costs. In the next section we introduce and briefly describe our automated assembly planning system known as Archimedes. We further go on to describe the principles and implementation of our system and how it lends itself to the planning of later lifecycle assembly processes; specifically, we describe our constraint-based approach for representing and reasoning about assembly constraints. Section 3 provides an overview of previous research efforts conducted in the areas of automated analysis of lifecycle assembly processes and optimization strategies in assembly sequencing. Section 4 introduces our optimizing search algorithm. We focus on the development of an algorithm that is designed to automatically select least-cost disassembly sequences for service-oriented operations. Finally, in Section 5 we conclude with a summary of our research efforts.

2 Assembly Planning System

2.1 Overview

Archimedes is a constraint-based interactive assembly planning software tool used to plan, optimize, simulate, visualize, and document sequences of assembly [8]. Given a CAD model of the product, the program automatically finds part-to-part contacts, generates collision-free insertion motions, and chooses assembly

* Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

order. The engineer specifies a quality metric in terms of application-specific costs for standard assembly process steps, such as part insertion, fastening, and subassembly inversion. Combined with an engineer's knowledge of application-specific assembly process requirements, Archimedes allows systematic exploration of the space of possible assembly sequences. The engineer uses a simple graphic interface to place constraints on the valid assembly sequences, such as defining subassemblies, requiring that certain parts be placed consecutively with or before other parts, declaring preferred directions, etc. Archimedes considers thousands of combinations of ordering and operation choices in its search for the best assembly sequences and ranks the valid sequences by the quality metric. Graphic visualization enables the engineer to easily identify process requirements to add as sequence constraints. Planning is fast, enabling an iterative constrain-plan-view-constrain cycle. For some restricted classes of products, it determines plans that optimize a given cost function, graphically illustrates those plans with simulated robots, and facilitates the generation of robotic programs to carry out those plans in a robotic workcell.

2.2 Constraints Background

In experiments with product designers and assembly process engineers, we have found that a high level of interactivity is critical to successful application of an assembly planner. Our framework provides a library of constraint types [7], from which the user can instantiate constraints on the assembly plan. While the focus of this paper is not constraints, we feel that is necessary to provide a background on our implementation of constraints since our constraint system provides underlying mechanics to our optimization algorithms.

Constraints are implemented as filters, procedures that either accept or reject assembly operations. Each constraint is described to the user in straightforward terms based on manufacturing and assembly sequencing concepts and defined using a simple graphic interface. During planning, each proposed assembly operation is passed to the constraint's filter function, which returns true or false depending on whether the operation satisfies the constraint or not. Only an operation that satisfies all current constraints is feasible.

For use in a standard state-space search method (such as generating an AND/OR graph for the assembly), it is important that the filter functions take single assembly operations as input, rather than more complex information such as a sequence of operations. In a state-space search, a given operation appears only once in the state graph, and is either present or not. Hence its feasibility cannot depend on operations that come before or after it.

2.2 Assembly Planning

The approach taken in assembly planning is critical to the design, implementation, and performance of a user constraint system. It especially affects special-purpose routines for efficiency. Archimedes generates two-handed monotone assembly sequences in reverse, starting from the more highly constrained, fully assembled state. The search space is an AND/OR graph of subassembly states and the operations used to construct them from smaller subassemblies. The strategy is designed to generate a first plan as quickly as possible, like a depth-first search, but to avoid getting caught by bad early decisions as a depth-first search would. This is critical to achieve the desired view-constrain-replan cycle of interaction. During each pass of the search algorithm, a single assembly sequence is generated, making random choices of operations to construct each subassembly. The first time any subassembly is visited, only a single operation is generated to construct it, and the known subassemblies of that operation are then visited. Bounds on quality measures for each subassembly and operation are stored and propagated in the AND/OR graph as they are generated. This allows useless search paths to be identified and pruned, and an optimal plan to be identified when it becomes available. The same algorithm functions as an any-time algorithm to optimize the assembly sequence when the user requests. The planner uses a non-directional blocking graph of each subassembly [14] to efficiently determine assembly operations that might be performed to construct that subassembly and then checks these operations for geometric collisions. This check serves as a built-in filter. Operations are then checked against the list of user constraints.

3 Previous Work

Previous efforts to incorporate a comprehensive set of user constraints in assembly planners were based on liaison precedence relations which specified logical combinations of part connections that must be established either before or after others. Precedence relations were pioneered by Bourjault [2] and greatly extended by DeFazio and Whitney [4]. Wolter et al [15] analyze the expressive power of precedence relations in detail. Precedence relations are quite powerful, but they can be very difficult to write correctly or understand as a user of an assembly planner. In our computer-aided assembly

planning system, described in [8], we chose a procedural approach over precedence relations for reasons of efficiency and simplicity of implementation. Our system demonstrates that an assembly planning system can achieve comprehensive constraint coverage while maintaining the advantages of a procedural representation.

As in initial assembly, the product design and known process constraints are inputs to creating such plans. However, for lifecycle assembly planning processes different goals and constraints, compared to initial assembly, require significant re-analysis of fundamental assumptions and methods underlying current assembly planning techniques. Some of those issues that require re-analysis include:

- Disassembly operations: The mechanics of disassembly operations must be characterized as to feasibility and cost, and differ greatly from their assembly counterparts for planning purposes. For instance, pressfits are rarely possible to disassemble without damaging one or both parts, which is sometimes acceptable.
- Partial assembly: (Dis)assembly does not always proceed from or result in individual parts. For instance, a field upgrade may only require partial disassembly of a system to replace specified subassemblies.
- Non-monotonic assembly: In the assembly planning literature, operations are non-monotonic when they leave parts in intermediate positions rather than placing or removing them completely. For instance, removing three screws from an access plate and leaving it hanging on the fourth screw is non-monotonic.
- Destructive disassembly: In some applications, operations that destroy parts (i.e., cutting, tearing, or melting) are acceptable in disassembly.
- Dismantlement optimization: This problem has an erratic cost criterion, since a small amount of disassembly sometimes makes huge gains in recovery/disposal gains/costs. This makes the optimization computationally difficult compared to initial assembly.

There is little prior art on planning lifecycle assembly processes and the issues thus raised. Non-monotonic assembly planning is the most difficult issue computationally and is know to be PSPACE-hard [15], and the only system to generate such plans was limited to an impracticably small number of parts [6]. The only known study of planning methods for destructive disassembly uses a simplified model of destructive operations that does not correspond to what is seen in practice [5]. The commercially available ReStar system [10] attempts to optimize disassembly processes for recycling and is based on a service-assembly planner described in [12]. Both rely on user input to determine all possible operations, making them impractical on products of more than ten to twenty parts.

Related research outside assembly planning is more extensive. Programs from Boothroyd-Dewhurst do enable design for service and recycling by analyzing plans entered by the user, but do no planning or optimization. Finally, researchers in concurrent engineering and green engineering have studied design-for-service and design-for-disposal (for instance [1,3,10]), but lack of assembly planning capability limits them to heuristic and statistical methods.

Milner and Graves [9] describe a tool which aided in the assessment of the true cost of assembling a product. The problem that they addressed was: Given a product design, determine the lease cost assembly system for that product. They developed a heuristic search through the multitudes of sequences to find those of nearly least-cost using simulated annealing (SA) to make such a search. However, a primary drawback of this system was that the least-cost sequences found by SA were often not of good engineering quality because engineering nuances could not be captured by the cost function.

Although Archimedes has proven to be a very powerful tool for optimizing assembly sequences, the above mentioned technical issues forced us to revisit our fundamental planning algorithms. Due to the flexibility inherent in our constraint system and our optimizing search algorithm, we have employed additional constraints in our system which allow us to address some of these drawbacks and limitations. In our system, we merged our constraint-based assembly planning algorithms with SA heuristics to produce optimal disassembly sequences in hopes of building a foundation for future research in planning for and optimizing the lifecycle assembly mentioned above.

4 Optimizing for Service-oriented Part Removal

As a first approach to service-oriented part removal, we employ the Archimedes disassembly planner essentially as-is, by specifying a constraint, which we refer to as REQ_SUCCESS_PART (parts), which essentially cuts the planner off when none of the desired parts are left in a subassembly. In general, this does not produce very satisfactory disassembly plans, as the first disassembly sequence produced by the planner typically contains large numbers of unnecessary operations, which have nothing to do with getting at the desired parts. As a next step, we implement the general search-optimization strategy that has been incorporated into Archimedes. This strategy uses a hill-climbing variety of the standard A* search [11] to repeatedly probe the subassembly

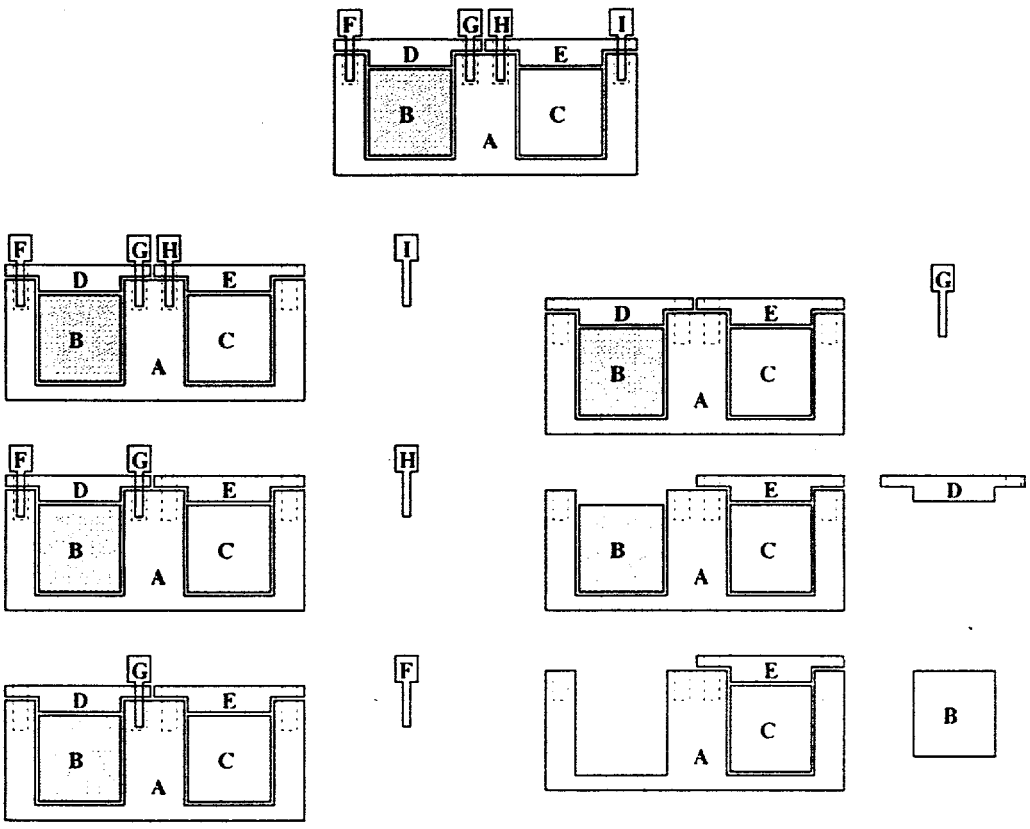

Figure 1. Example of an assembly from which we wish to remove a specific part, B: the initial plan removes parts H and I unnecessarily.

tree, looking for better (less costly) disassembly sequences. Running with a REQ_SUCCESS_PART constraint, and the optimizer enabled, causes Archimedes to iteratively run the disassembly planner, with an aim toward finding an "optimal" disassembly sequence. The debilitating drawback to this approach is that the size of the subassembly tree makes it impossible to find optimal solutions in reasonable time, for assemblies with more than about 20 parts. The nature of the optimization scheme is such that the algorithm will run until the cost of the best disassembly sequence found so far is equal to the lower bound on the cost of the best disassembly sequence possible. The problem is that if the base assembly admits a large number of possible disassembly steps, the execution time required to find the best possible disassembly sequence is prohibitive. Even proving that the best possible sequence is at least two operations deep requires the exploration of tens of thousands of disassembly sequences. Finding a "best" disassembly sequence can also take a prohibitively long time. "Best" means that a person can analyze an assembly sequence and say with certainty that a particular sequence is clearly the best possible. Archimedes may still be unable, using the general-purpose optimizer, to find that sequence, let alone say that no other sequence is better.

One of the reasons that the general-purpose optimizer does not provide adequate support for service-oriented part removal is that it is not capable of eliminating, in an efficient manner, what humans would think of as "obviously unnecessary" part removals from the disassembly sequence. For example, consider the partial disassembly sequence shown in Figure 1; the task is to remove the part labeled B from the assembly for service or replacement. It is obvious that the correct disassembly sequence is to remove the screws labeled F and G from the assembly and then to remove the cover plate, D. Because Archimedes has no way of knowing, a priori, that removing screws H and I will not help in gaining access to part B, it might return the sequence shown in the figure; removal of parts I, H, F, G, D, and then B. In this case the number of parts is small enough that the optimizer would quickly discover the sequence {F, G, D, B}, and also quickly discover that no cheaper sequence would achieve the removal of part B. But if the assembly

had, say, 40 or 50 parts, Archimedes might run for an arbitrarily long time before discovering the sequence {F, G, D, B}, and would run even longer before proving to its own satisfaction that a cheaper sequence did not exist.

In order to improve Archimedes' ability to find sequences that do not contain "obviously unnecessary" part removals, we implemented a specific module that operates on existing plans to improve their efficiency. The planner invokes two steps when a REQ_SUCCESS_PART constraint is declared. First, the standard disassembly planner is used to provide an initial sequence P that enables the removal of the desired part (in this case, B). Second, a subroutine (hereafter referred to as the "shortening algorithm") is invoked which attempts to shorten P by eliminating removal operations from it.

Algorithmically, this is accomplished by attempting to indefinitely defer specific removal operations. For the disassembly sequence shown in Figure I, {I, H, F, G, D, B}, the planner attempts to determine if, instead of removing part I first, part H could be the first part removed. Since this is possible, the planner goes on to consider the sequence {H, I, F, G, D, B}. This sequence has all of the same operations, and the same cost, but has deferred the removal of part I for one operation.

The planner then tries to defer I's removal by another step: {H, F, I, G, D, B}. It is successful, so it goes on to try {H, F, G, I, D, B}, {H, F, G, D, I, B}, and finally {H, F, G, D, B, I}. For this final disassembly sequence, however, the planner realizes that it does not need to remove part I, having already removed part B, and so it returns the plan {H, F, G, D, B}, which is cheaper by the cost of removing part I. The purpose of "marching" part I through the sequence in this fashion, rather than directly eliminating it from the plan, is to allow the contact and collision analyses to be applied at each step to ensure that we do not try to illegally remove parts by, for example, pulling them directly through part I.

If the planner is not successful in deferring the removal of the first part indefinitely, it goes on to try again with the second part, then the third part, and so on, until it reaches the end of the plan. If it never succeeds, then it simply returns the original plan. If, on the other hand, it does succeed in deferring the removal of any particular part indefinitely then it stops. The planner then re-computes the "best plan", and repeats, until it fails to indefinitely defer the cost of any part removal. In our example, after specifically generating the removal operations needed to provide the disassembly sequence {H, F, G, D, B}, the planner then goes on to try to defer the removal of part H, generating, in order, the sequences {F, H, G, D, B}, {F, G, H, D, B}, {F, G, D, H, B}, and finally {F, G, D, B, H}, which is truncated to {F, G, D, B}, which is, in this case, the optimal solution.

It must be stressed that this additional algorithm does not guarantee convergence of the optimized search for a cheapest disassembly sequence. This is true for two reasons. First, this algorithm does not provide any information about minimum possible costs. In this case, if there were 40 or 50 parts in the assembly, the algorithm would do nothing toward proving that there was not a cheaper sequence. If cost and length of the disassembly sequence are considered to be identical for the purposes of illustration, then this algorithm would do nothing toward showing that there does not exist a length 3 disassembly sequence terminating in the removal of part B from the assembly in Figure 1.

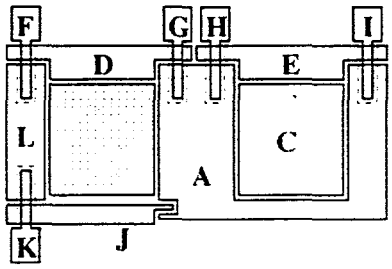

Figure 2. Example of assembly where the shortening algorithm is not guaranteed to find an optimally short disassembly sequence.

The second reason that the results of this algorithm are not guaranteed is that the effects of this algorithm are only visited upon plans found by the existing disassembly planner. Suppose for example, that part B was held within its cavity in part A not only by the plate D, but also by a plate J, which is held in place only by a single screw, K, shown in Figure 2. Then the optimal removal sequence would be {K, J, B}. Until the planner finds a sequence containing {K, J, B} as a subsequence, the shortening algorithm will not be able to produce {K, J, B} as an alternative.

5 Conclusion

In this paper, we briefly examined some of the issues in lifecycle engineering which are currently considered critical and used these issues to motivate the current direction in which the Archimedes assembly planning tool is developed. We summarized the existing capabilities of Archimedes and described recent developments within Archimedes that are intended to enable its use in a more general lifecycle framework by allowing users to compute

disassembly plans tailored to specific service, upgrade, and/or dismantlement strategies. Specifically, we described a "shortening algorithm" which makes it much easier for Archimedes to find shortest or least-cost disassembly sequences to gain access to specific parts and subassemblies. In practice the shortening algorithm provides an effective way to find the lowest-cost servicing disassembly plan. It allows replacing one or several subassemblies, gaining access for inspections, and optimizing disassembly to minimize service-related costs. The user can specify the set of parts for the service operation, enable the optimizer, start the planner, and let it run until the cost of the best plan so far stays the same for several iterations. At that point the plan will, with high probability, be the one desired. If not, the user can either rerun the planner for a longer duration, or add constraints to help the planner find the minimum cost service plan.

References

[1] D. Allen and T. R. Roose. Life cycle assessment and design for environment. In P. M. Eisenberger. Editor, Basic Research Needs for Environmentally Responsive Technologies of the Future: An Integrated Perspective of Academic, Industrial, and Government · Researchers. Princeton Materials Institute, 1996.

[2] A. Bourjault. *Contribution à une aproche méthodologiuqe de l'assemblage automatisé: élaboration automatique des séquences opératories.* PhD thesis, Faculté des Sciences et des Techniques de l'Université de Franche-Comté, 1984.

[3] D. S. Burks, M. D. Marks, and K. Ishii. Life cycle design for recyclability. In D. Navin-Chandra, editor, Green Engineering, Academic Materials Institute, 1996.

[4] T. L. De Fazio and D. E. Whitney. Simplified generation of all mechanical assembly sequences. IEEE Journal of Robotics and Automation, RA-#(6):640-658, 1987. Errata in RA-4(6):705-708.

[5] R. Gadh, Z. Ashai, and K. Lee, Computer-aided design-for-disassembly; as applied to design-for-environment. Technical Report I_CARVE-1091, Dept. ME, Univ. of Wisconsin, 1994.

[6] R. L. Hoffman. Automated assembly in a CSG domain. In Proc. IEEE Intl. Conf. On Robotics and Automation, pages 210-215, 1989.

[7] R. E. Jones and R. H. Wilson. A survey of constraints in assembly planning. In Proc. IEEE Intl. Conf. On Robotics and Automation, pages 1525-32, 1996.

[8] R. E. Jones, R. H. Wilson, and T. L. Calton. Constraint-based interactive assembly planning. In IEEE Intl. Conf. On Robotics and Automation pages 913-920, 1997.

[9] J. M. Milner and S. C. Graves. Using simulated annealing to select least-cost assembly sequences. In IEEE Intl. Conf. On Robotics and Automation, pages 2058-2063, 1994.

[10] D. Navin-Chandra. ReStar: A design tool for environmental recovery analysis. In Proc. Intl. Conf. On Engineering Design, 1993.

[11] N. J. Nilsson. Principles of Artificial Intelligence. Springer-Verlag, 1980.

[12] A. K. Subramani. Development of a Design for Service Methodology: PhD thesis, Univ. of Rhode Island. 1992.

[13] R. H. Wilson. Minimizing user queries in interactive assembly planning. IEEE Trans. On Robotics and Automation, 11(2):308-312, 1995.

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

[15] J. D. Wolter. On the Automatic Generation of Plans for Mechanical Assembly. PhD thesis, Univ. of Michigan, 1988.