SANDIA REPORT
SAND2000-0210
Unlimited Release
Printed February 2000

Very Large Assemblies: Optimizing for Automatic Generation of Assembly Sequences

Terri L. Calton

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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

Sandia National Laboratories

SAND2000-0210
Unlimited Release
Printed February 2000

Very Large Assemblies: Optimizing for Automatic Generation of Assembly Sequences

Terri L. Calton
Intelligent Systems and Robotics Center
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1008

Abstract

Sandia's Archimedes 3.0© Automated Assembly Analysis System has been applied successfully to several large industrial and weapon assemblies. These have included Sandia assemblies such as portions of the B61 bomb, and assemblies from external customers such as Cummins Engine Inc., Raytheon (formerly Hughes) Missile Systems and Sikorsky Aircraft. While Archimedes 3.0© represents the state-of-the-art[1] in automated assembly planning software, applications of the software made prior to the technological advancements presented here showed several limitations of the system, and identified the need for extensive modifications to support practical analysis of assemblies with several hundred to a few thousand parts. It was believed that there was substantial potential for enhancing Archimedes 3.0© to routinely handle much larger models and/or to handle more modestly sized assemblies more efficiently. Such a mature assembly analysis capability was needed to support routine application to industrial assemblies that overstressed the system, such as full nuclear weapon assemblies or full-scale aerospace or military vehicles.

[1] Prior to the technological advancements presented here, Archimedes represented the state-of-the-art in assembly planning software. Today, two years later, Archimedes is still at the forefront of technology in automated assembly planning software systems.

Intentionally Left Blank.

Preface

This document serves as the technical report required by Sandia National Laboratories Laboratory Directed Research and Development (LDRD) Program for the project entitled "Analysis of Very Large Assemblies". The LDRD Program financially supported the entire project and this document summarizes the technical achievements made throughout the project's two-year history. The work conducted under the purview of the LDRD focused on increasing the automatic analysis capabilities for large assemblies. Sandia's Automated Assembly Analysis System, Archimedes 3.0©, was used as the foundation for integrating algorithms to automatically analyze assembly and disassembly plans for large assemblies. The results produced under the purview of this LDRD, together with the results produced under the purview of three other LDRDs, (a three-year (FY97 and FY99) LDRDs project entitled "Automatic Planning of Life Cycle Assembly Processes [1]" and two two-year (FY98-FY99) LDRD project entitled "Ergonomics in Life Cycle Assembly Processes [2]" and "Feature Reduction of Geometric Solid Models for Analysis Tools [3]") constitute the features used to upgrade Archimedes 3.0© to Archimedes 4.0. Archimedes 3.0© was copyrighted in 1998. The copyright for Archimedes 4.0 is in progress.

Intentionally Left Blank.

Contents

Figures

Tables

1

Introduction

Sandia's Archimedes 3.0© Automated Assembly Analysis System has been applied successfully to several large industrial and weapon assemblies. These have included Sandia assemblies such as portions of the B61 bomb, and assemblies from external customers such as Cummins Engine Inc., Raytheon (formerly Hughes) Missile Systems and Sikorsky Aircraft. While Archimedes 3.0© represents the state-of-the-art[2] in automated assembly planning software, applications of the software made prior to the technological advancements presented here showed several limitations of the system, and identified the need for extensive modifications to support practical analysis of assemblies with several hundred to a few thousand parts. It was believed that there was substantial potential for enhancing the Archimedes 3.0© to routinely handle much larger models and/or to handle more modestly sized assemblies more efficiently. Such a mature assembly analysis capability was needed to support routine

[2] Prior to the technological advancements presented here, Archimedes represented the state-of-the-art in assembly planning software. Today, two years later, Archimedes is still at the forefront of technology in automated assembly planning software systems.

application to industrial assemblies that overstressed the system, such as full nuclear weapon assemblies or full-scale aerospace and military vehicles.

The emphasis of past Archimedes' projects has been on the development of a practical, but theoretically sound and novel assembly analysis software system. Thus, efforts, which could have been pursued in the interest of efficiency usually, took second place to implementing basic theoretical capabilities and addressing usability issues. However, we believed that there was substantial potential for enhancing Archimedes 3.0© to routinely handle much larger models, perhaps up to a few thousand parts. The difficulties and corresponding opportunities fell in four categories: contact analysis, memory utilization, planning time and geometric simplification.

Section 2 discusses contact analysis deficiencies in the Archimedes 3.0© System and methods that were developed to overcome them. In the Archimedes 3.0© System, analysis of contact geometry between nearby parts must be performed before actual assembly planning can begin. The existing approach may be overly thorough for practical situations, but methods were needed to provide a more optimal balance between theoretical thoroughness and practical efficiency. We believed that the main problem was that the CAD models input in Archimedes 3.0© were analyzed as a whole, rather than as having any spatial segmentation. The non-directional blocking graph theory, which is at the heart of the Archimedes' algorithms, limits how much flexibility we have in this area. A full analysis is beyond this report's scope, but we were confident that it would be practical to limit the full assembly analysis to a relatively small number of common contact orientations, and apply the full local contact information only in local contexts as needed. We believed major execution efficiency gains were possible through such work, with only minimal impact on the robustness of the assembly planning process. Also, the execution time of the contact analysis itself, while not as important as the method of handling contact data during planning, would be made more efficient through more attention to spatial segmentation.

Section 3 focuses on memory utilization deficiencies of the Archimedes 3.0© System and presents methods developed to overcome them. Simply loading large models was problematic. The importance of good memory utilization in allowing the Archimedes 3.0© System to load and process large models was emphasized by previous and planned efforts in this area. The "Automatic Planning of Life Cycle Assembly Processes" provided a capability [1,2,3] in Archimedes 3.0© to load only one copy of similar parts (such as fasteners), thus eliminating one of the memory inefficiencies.

Sections 4 and 5 focus on optimizing planning algorithms developed and integrated into the Archimedes 3.0© System. Model loading and contact analysis are usually performed relatively infrequently, whereas the majority of the computer time is in repeated searching for valid, or improved, assembly plans. This search process is complex, but a number of possible shortcuts or efficiency improvements were identified. Section 4 focuses on the automatic planning for manufacturing generative processes. Generative process planning describes the methods process engineers use to modify manufacturing (or process) plans after a design is complete. A completed design may be the result from the introduction of a new product based on an old CAD design, an assembly or subassembly upgrade based on CAD, or modified product CAD designs used for a family of similar products. This section describes methods that allow users to automatically combine assembly plans resulting from independent applications of the Archimedes 3.0© System to individual subassemblies of a larger assembly to form a complete assembly plan for the entire assembly. It further describes the implementation of constraint rules enabling a user to automatically "reconcile" existing constraints applied to an older version of an assembly to a new version of an assembly, which might have a different number (or ordering) of parts.

Section 5 discusses automatic planning for partial disassembly (work conducted jointly with the "Automatic Planning of Life Cycle Assembly Processes" LDRD) and several algorithmic solutions to increase the efficiency of the Archimedes 3.0© planning capabilities. The section covers the inclusion of a newly implemented constraint, REQ_SUCCESS_PART(s), into the Archimedes 3.0© System to accommodate service-oriented assembly processes. The instantiation of a REQ_SUCCESS_PART(s) constraint allows a user to specify a part or collection of parts that must be removed from an assembled product. This forces the planner to halt when none of the desired parts (as specified in the constraint) are left in the assembly. The section further describes a computer algorithm (and its implementation), which optimizes the search strategy for the REQ_SUCCESS_PART(s) constraint. This search strategy uses a hill-climbing technique of the standard A* search to repeatedly probe the subassembly tree, looking for better (less costly) disassembly sequences. An additional algorithm, referred to as the *shortening algorithm,* is also described. The purpose of the shortening algorithm is to "shorten" the initial sequence by eliminating unnecessary removal of parts not specified in the REQ_SUCCESS_PART(s) constraint set.

Section 6 describes geometric simplification algorithms implemented specifically for the Archimedes 3.0© planner and synergistic geometric simplification algorithms developed under the "Feature Reduction of Geometric Solid Models for Analysis Tools" LDRD. Finally, Section 7 concludes the report a summary of the technological advancements, some experimental results and conclusions and suggests areas of future research.

Intentionally Left Blank.

2

Contact Analysis Improvements

In the Archimedes 3.0© Automated Assembly Analysis System, analysis of contact geometry between nearby parts must be performed before actual assembly planning can begin. The current approach may be overly thorough for practical situations, but methods were needed to provide a more optimal balance between theoretical thoroughness and practical efficiency. We believed that the main problem was that the CAD models input in the Archimedes 3.0© System were analyzed as a whole, rather than as having any spatial segmentation. The non-directional blocking graph theory, which is at the heart of the Archimedes' algorithms, limits how much flexibility we have in this area. A full analysis is beyond this report's scope, but we were confident that it would be practical to limit the full assembly analysis to a relatively small number of common contact orientations, and apply the full local contact information only in local contexts as needed. We believed major execution efficiency gains were possible through such work, with only minimal impact on the robustness of the assembly planning process. Also, the execution time of the contact analysis itself, while not as important as the method of handling contact data during planning, would be made more efficient through more attention to spatial segmentation.

2.1 Initial Contact Analysis

We determined that the majority of the time the code expended during contact analysis was not in an initial pass over the assembly prior to running the planner a single time, but rather was in the repeated computation of a full assembly analysis. A full-assembly contact analysis had to be performed every time a single mating OVERRIDE or one of a small subset of the assembly CONSTRAINTS was invoked. For example, a particular 369-part assembly with 197500 facets took, late in FY97, approximately three minutes for a full contact analysis. Addition of a single MATE OVERRIDE would necessitate a full re-computation of the contact analysis, with only a small part of the computation performed in the original analysis being reused. At that time, redoing the contact analysis took roughly 2 1/4 minutes. We restructured the contact analysis so that only parts involved in new or revised OVERRIDES or CONSTRAINTS would be subjected to further analysis. After restructuring, addition of a single OVERRIDE or CONSTRAINT, involving for example, two of the 369 parts, took only about 3 seconds, and almost 98% improvement for that assembly. For a more complicated assembly, or one with more parts, the savings would be even greater. This a very significant from the user's perspective, because users of the Archimedes 3.0© System often react to problems in the produced assembly plan like modern programmers; they fix one or two problems, then rerun the planner. So, reducing the contact analysis time on second and subsequent runs reduces the amount of user time required very dramatically.

2.2 Contact Analysis Cache/Preservation

We have also added the ability to cache contact analysis on disk, enabling the user to carry contact information across invocations of the Archimedes 3.0© Automated Assembly Analysis System. A more careful assessment of the effects of OVERRIDES and certain CONSTRAINTS on existing contact analysis was made. As a result, a more selective contact update scheme was implemented. Once the initial contact analysis for an assembly has been achieved, the improved scheme reduces the time required to update the contact analysis after adding a single OVERRIDE or selected CONSTRAINT from the full amount required by the initial analysis (as much as several hours, for assemblies with a few hundred parts) down to a few seconds to a few minutes, depending on the complexity and numbers of the individual parts involved in the introduction of OVERRIDES. Since the ability to save and retrieve the contact information has also been added, this means that the full, all parts against all other parts contact analysis need only be performed once for any given assembly.

3

Memory Utilization Improvements

Simply loading large models into the Archimedes 3.0© System was problematic. The importance of good memory utilization in allowing the System to load and process large models was emphasized by previous and planned efforts in this area. The "Automatic Planning of Life Cycle Assembly Processes" provided a capability [1,2,3] in the Archimedes 3.0© System to load only one copy of similar parts (such as fasteners), thus eliminating one of the memory inefficiencies. To improve memory utilization within the system, two areas were focused on. The first was reducing multiple instantiations of the same part to a single representation with in memory while the second focused on the graphical output data structures.

3.1 Single-Part-Multiple-Instantiation

A Single-Part-Multiple-Instantiation (SPMI) feature was developed and integrated into the Archimedes 3.0© System. A toggle in the File menu was added that the user to determine whether or not to invoke the SPMI feature at load-time. SPMI allows the Archimedes 3.0© System to put multiple instances of a given part into an assembly without representing each instance separately, with the entire storage overhead that that entails. The toggle control is necessary, because SPMI is achieved by storing only one copy of each part's ACIS and facetted data, with each in its own coordinate frame. In order to analyze and display multiple instances of a part, its ACIS and facetted date must be transformed into the assembly's common coordinate frame. This can be costly for complex parts, and is therefore primarily for use in cases where representing each instance of each part separately makes the process use too much memory, or where using SPMI markedly reduces page swapping.

By restructuring some of the graphical output code, we were able to achieve a 93% reduction in the required storage for each part in an assembly (from about 2 Megabytes/part down to about 140 Kilobytes/part) for the electronic subassembly array shown in Figure 3.1. In terms of assembly size limitations, this raises the bar from a previous limit of about 600-8000 parts for assemblies running on a 32-bit SGI, to a level in excess of 10,000 parts. Effectively, required memory has ceased to be a limitation on the size of assemblies.

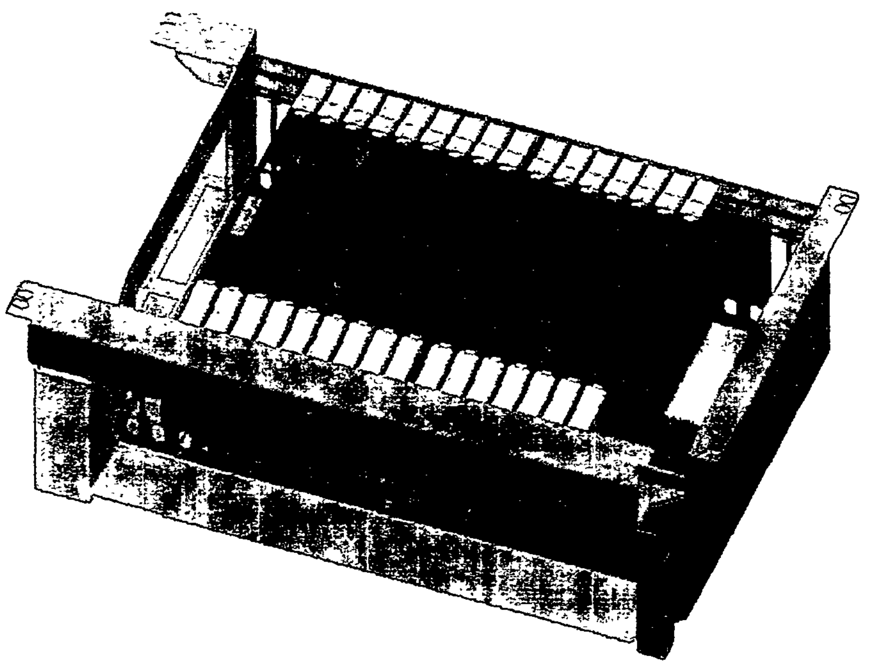


Figure 3.1. Electronic array subassembly.

4

Automatic Planning for Manufacturing Generative Processes

Generative process planning describes the methods process engineers use to modify manufacturing (or process) plans after a design is complete. A completed design may be the result from the introduction of a new product based on an old design, assembly upgrade, or modified product designs used for a family of similar products. Typically, an engineer designs an assembly and process plans are created capturing the manufacturing processes, including the assembly sequence, the methods used to put the piece parts together, the cost of the piece parts, labor costs, etc. When new products originate as a result of an upgrade, the geometry of parts may change, and/or additional components and subassemblies are added to or are omitted from the original design. As a result, process engineers are forced to create a "new" set of process plans. Often times, this is a tedious and time-consuming task, even if only a single component is involved in the upgrade. The task is further complicated by the fact that the process engineer is forced to manually generate these plans for each product upgrade. Assembly planners, including Archimedes 3.0©, can not automatically handle the upgrade modifications. To generate new assembly plans for the product upgrade, engineers have to manually re-specify the manufacturing plan selection criteria and re-run the planners. To remedy this problem, special-purpose routines have been added to the Archimedes 3.0© planning algorithms and constraint framework.

Section 4.1 provides an overview of the motivational factors for implementing automatic generative process planning techniques. Section 4.2 places the Archimedes 3.0© System in the context of generative process planning. Section 4.3 discusses the geometric issues associated with top-level assembly planning, and the special-purpose routines that were implemented within the Archimedes 3.0© framework to solve those problems. Section 4.4 continues this discussion, but at the subassembly-level. Methods are presented for saving, restoring, and propagating the subassembly analyses for top-level assembly analysis. Section 4.5 describes the implementation of constraint rules enabling a user to automatically "reconcile" existing constraints applied to an older version of an assembly to a new version of an assembly. Finally, Section 5.6 concludes the Section and presents future research topics in this area.

4.1 Background and Motivation

The Archimedes 3.0© System has been applied to hundreds of assemblies, ranging from automotive and aircraft to such things as designing assembly sequences for several weapon safety devices and for the B61. The B61, with improved non-nuclear components, has replaced the B53 in the U.S. stockpile. The scope of the modifications to the B61 requires exhaustive testing to certify the modified bomb's safety, functionality, and reliability. In an early experiment, Archimedes 3.0© was applied to the B61 center-case for the B61 Alt 339 Retrofit Program. It was estimated that 2.5-3 person months were required to manually create training documentation for the retrofit operations using a commercial animation package.

In an effort to reduce the time, Archimedes 3.0© was applied to the B61 center-case assembly. Unfortunately, the experiment showed that there were many assembly planning issues associated with CAD revisions that went beyond the disassembly issues discussed in the previous section. For instance, the first step required to apply Archimedes 3.0© was to translate the Pro/ENGINEER® data to the ACIS format. Initially, the entire center-case assembly, containing 547-parts and represented by over 600K facets, was selected for the Archimedes 3.0© application; however, due to translation the problems a 303-part subassembly (345K facets) was exercised during the experiment. Effectively, the original design was modified by removing parts.

Archimedes 3.0© was first applied to the original (larger) solid model to identify any inconsistencies in the CAD model. This allowed for the detection of critical design flaws to be caught early in the re-manufacturing phase and a reduction in scheduling and costs. Next, Archimedes 3.0© was used to test the feasibility of disassembly, checking for geometric accessibility for removal of parts. Since Archimedes 3.0© plans only for straight-line motions, and this assembly contained a number of flexible parts, such as cables, that could not use straight-line assembly motions, the part-mating operations involving those parts were overrode. This was a long and tedious manual process. When it was decided to exercise the smaller assembly, these same tedious steps had to be repeated, almost identically to the first application, since Archimedes 3.0© can not reconcile the differences (part count, geometry, constraints and overrides) between the two assemblies automatically.

This same problem was inherent in the application of Archimedes 3.0© to the nose assembly (shown in Figure 5.1) at the Federal Manufacturing & Technologies Facility at Allied Signal in Kansas City. In this case a new modification and proposed design of the B61 nose assembly incorporated new radar hardware and sophisticated structure elements to withstand high-shock environments. During hardware evaluation stages at Allied Signal, the system was used to determine re-manufacturability of the B61 nose assembly. Pro/ENGINEER® parts were transferred to the Archimedes 3.0© System for validation, assembly plans were evaluated, and assembly instructions and options were evaluated with process engineers. The nose assembly contains 88 parts described by 17.5Mb of ACIS® data (translated from Pro/ENGINEER®) and approximately 110,000 facets.

Unlike the center-case assembly, the nose assembly went through several revisions before being finalized; however just like the center-case assembly application, for each revision the assembly planning steps had to be repeated (often duplicated). To complicate matters further, Archimedes 3.0© had to be applied to each subassembly even if they were identical.

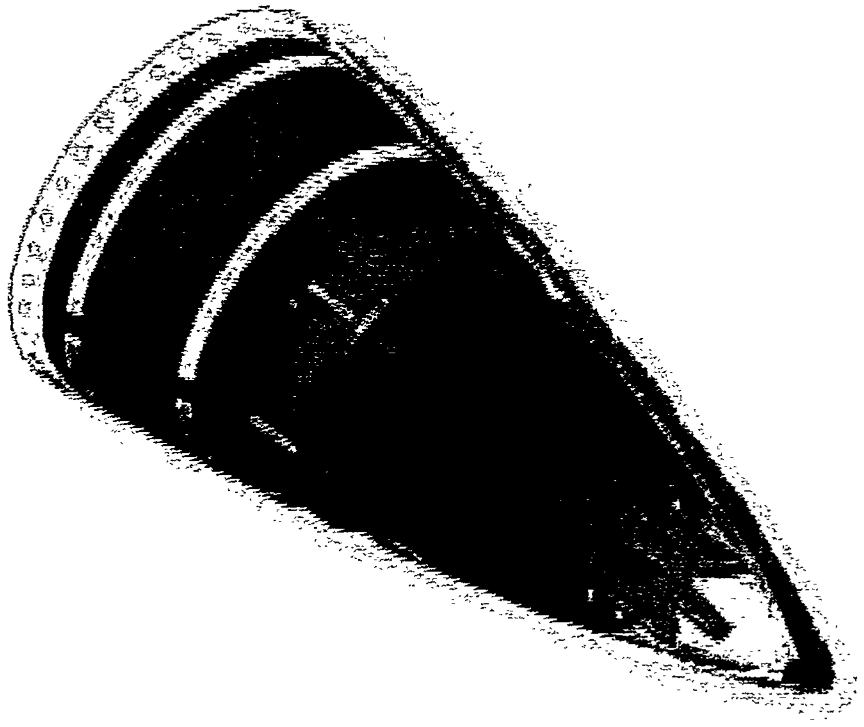


Figure 4.1. The B61 nose assembly.

4.2 Generative Process Planning Issues

In the previous section four fundamental problems were identified during the application of Archimedes 3.0© to the B61.

1. Automatic planning with subassemblies: When assemblies are built by combining subassemblies, it is sometimes more practical to analyze the assembly at the subassembly level. That is, analyze each subassembly independently and then combine the analyses at the top-level (the assembly level). Archimedes 3.0© plans for assembly at the top-level of the assembly and does not allow the propagation of information resulting from independent applications at the subassembly-level to the top-level.
2. Multiple identical subassemblies: If an assembly contains exact copies of the same subassembly and that subassembly requires any instantiation of constraints and overrides, the Archimedes 3.0© user must instantiate them in each subassembly (duplicating his/her efforts on each of the identical subassemblies). This is because Archimedes 3.0© plans for assembly at the top-level of the assembly.
3. Assembly upgrade via re-design: An assembly may be modified by removing part(s), by changing the shape of the part(s), by adding parts(s), or by any combination of the three. Relating to the first two problems is the inability of the Archimedes 3.0© System to automatically reconcile not only these modifications between design revisions, but also any constraints and overrides applied to the assembly for each generation.
4. Gaining access to selected parts for servicing[3]. This is the often associated with partial disassembly. Partial disassembly does not always proceed from or result in the removal of all of the parts in an assembly. For instance, a field upgrade may only require partial disassembly of a system to replace specified subassemblies. Archimedes 3.0© does not plan for partial disassembly.

The problems identified above are not restricted to Archimedes 3.0©. They represent a fundamental class of problems inherent in all assembly planners and have plagued the manufacturing community for years. It is only recently, with the advancements in computer technology, that these problems have been brought to the forefront. With the exception of problem 4, the remainder of this section is devoted to solving these problems. The underlying principles, as they relate to automatic assembly planning, are discussed, and solutions to each are provided. The solutions have been implemented as extensions to the Archimedes 3.0©. Solutions to problem 4 are provided in the next section.

[3] This issue is addressed in more detail in the next chapter. Archimedes 3.0© was applied to the B61 tail-section for parachute inspection and servicing.

4.3 Re-design

There are two fundamental issues associated with assembly design modification: geometry and function. For purposes of assembly planning only the geometric will be discussed. There are three geometry-related design modification principles for any given assembly. An assembly may be modified by removing part(s), by changing the shape of the part(s), by adding parts(s), or by any combination of the three.

In general, part removal is primarily motivated by a company's need to reduce the time it takes to put the product together. Obviously, the greater the number of parts, the higher the assembly costs. From an assembly standpoint, it is the simplest form of modification to deal with. Changing the geometry of a part is often the result of assembly facility and shop floor contingencies. For instance, some shapes are better suited for robotic assembly as opposed to manual assembly. The most difficult issue to address when planning assembly processes automatically for a product is the case where new parts are added to an existing assembly design.

Figure 4.2 shows an example assembly[4] having gone through 3 design revisions. Initially, the assembly was comprised from 24 pieces: a base plate, two side plates, a top rail, and 20 various fastener piece parts (nuts, bolts, and washers). In an effort to reduce costs by reducing time spent assembling the product (i.e., reduce part count), several re-designs were made. In the first revision (R1), the fastening processes were addressed. As a result the fasteners were replaced with self-locking screws and inserted from the top. Hence, the results from the first revision reduced the part count to 8. In the second revision (R2), the base plate and two side plates were re-designed into a single unit. With this modification, two new parts were incorporated. These are the clamp-like fasteners to hold the top rail in place. In the final revision (R3), the base is re-designed with tabs to hold the top rail, and the clamp-like fasteners are removed. Thus, the product has been reduced to 2 components.

In this simple example, all three re-design principles were implemented. This is true for almost all product upgrades. The first revision was a result of removing parts and adding new ones. In the second revision, part geometry changed and new parts were added. In the third revision, part geometry changed and parts were eliminated. The fundamental problem relating these re-design principles to automatic assembly planning is the inability of the planners to automatically recognize the changes between generations.

To address the first, the removal of parts, a geometric override was added to the Archimedes 3.0©override architecture that removes all associations of that part with others (e.g., part contacts, overrides, and constraints) and effectively hides the part from the user's view. In Archimedes 3.0©, routines to save and restore assembly plans, assembly constraints, and geometric overrides are implemented at the top-level assembly. This allows a user to analyze an assembly at the top-level and save all of the analysis information. When the system is applied to the same assembly at a later time or to different generations of that assembly, the information may be invoked by restoring the files.

[4] This assembly was borrowed from [Boothroyd, G., "Assembly automation and product design", Marcel Dekker, New York, 1991, p. 14.] for illustrative purposes only. Hypothetical processes are used to illustrate the principles of re-design as they relate to automatic assembly planning.

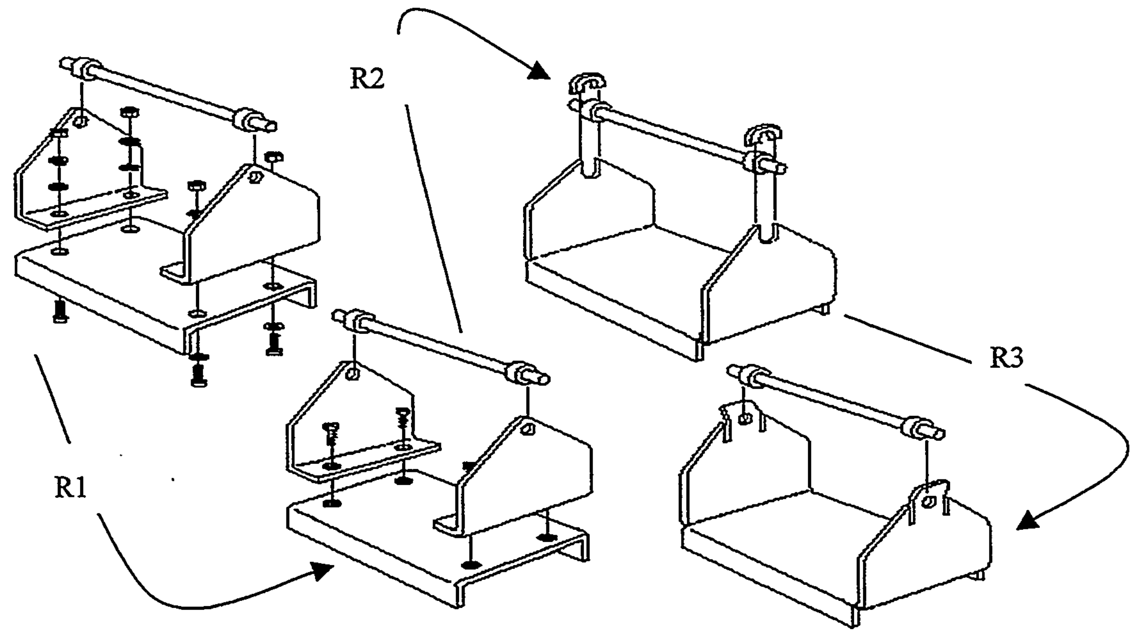


Figure 4.2. Example of the effects of re-design.

When the user loads in the assembly, the constraints and override files are automatically loaded. The assembly is represented by data bit-vector. The length of vector corresponds to the number of parts in the assembly. In all constraints and overrides, a "0" in a particular bit means one thing about a part and a "1" means something else, depending the type of constraint or override that is implemented. In this case, a "0" in the bit-vector notifies the system that that particular part is no longer in the assembly. While the part is still present in the assembly tree (i.e., the length of the bit-vectors for all constraints and overrides is constant), for all intense purposes it has been removed.

To address the second, changing component geometry, the Archimedes 3.0©contact analysis routines automatically check contacts between parts. If the re-design changes the contacts between the parts, the user is automatically informed and is given the opportunity to address the issue. The same holds true for constraints and overrides. If a previously defined constraint or override is now in conflict, the user is automatically informed and is given an opportunity to address the issue(s).

The third issue, the addition of parts, is the most difficult issue. Because Archimedes 3.0© plans for assemblies at the top-level and the length of data bit-vector representing the number of parts at the top-level is fixed, the Archimedes 3.0© System can not plan for assemblies upgrades at the top-level when the part count increases. This is a major research area on its own, and attention to solving this problem should be given to future work in this area.

4.4 Planning with Subassemblies

This section deals with the generative process planning problems, 1 and 2, discussed in Section 4.2.

1. Analyzing each subassembly independently and then combining the analyses at the top-level (the assembly level).
2. Multiple identical subassemblies: If an assembly contains exact copies of the same subassembly and that subassembly requires any instantiation of constraints and overrides, the Archimedes 3.0© user must instantiate them in each subassembly (duplicating his/her efforts on each of the identical subassemblies).

It is pointed out in the previous section that Archimedes 3.0© plans for assembly at the top-level of the assembly and does not allow the propagation of information resulting from independent applications at the subassembly-level to the top-level. The first step to solving these problems was to incorporate save and restore routines for the constraints and overrides resulting from the application of Archimedes 3.0© to the subassemblies, which would automatically load when Archimedes 3.0© was applied at the top level of the assembly. The underlying problem with this approach is how to resolve conflicts between the constraints and overrides when they are propagated to the top. Section 4.5 addresses the conflict resolution issues. Here, the methodologies for automatically propagating the information generated at the subassembly-level to the top-level are presented.

4.4.1 Automatic Propagation

File-restoration subroutines were incorporated into Archimedes 3.0© to automatically load and propagate the constraints and overrides from the subassembly files to the top-level assembly. When Archimedes 3.0© is applied at the subassembly-level, constraints and overrides are stored under the default name of the subassembly (e.g., subassembly-name.constraints and subassembly-name.overrides). When loading an assembly (the base assembly or top-level assembly), the subassembly constraints are automatically loaded using the subassembly-name.constraints default file. For the constraint restoration subroutine that restores two subassembly sets, bits in the data vector are set as follows for: (visible - 0 and group - 1). For the constraint restoration subroutine that restores three subassembly sets, bits in the data vector are set as follows for: (visible - 1, secondgroup - 0, and group - 0). For each subassembly file restoration, the number-of-parts bit for the full assembly set is equal to the number of parts in the subassembly. The restoration algorithm changes the subassembly's portion of the assembly data vector to be that read from the subassembly file. Any parts in the vector not belonging to the subassembly are set to 0. The algorithm changes the number-of-parts bit to equal the number of bits set in the data vector.

When loading an assembly, the subassembly overrides are also automatically loaded using the subassembly-name.overrides default file. The subassembly overrides are loaded with a new override class feature, called IsTopLevel, set 0 (or *false*), to indicate that they were loaded from the subassembly's overrides file, not from the base assembly's overrides file. The assembly overrides are created with IsTopLevel set to 1 (*true*). Only top-level overrides are saved for an assembly. Conflicting overrides made at the base assembly level are intended to take precedence over overrides made locally to the subassembly.

4.4.2 Demonstration of Propagation Effects

To help illustrate the propagation of design modifications imposed at the subassembly-level for later use in planning at the top-level, conceptual designs of two assemblies are provided below.

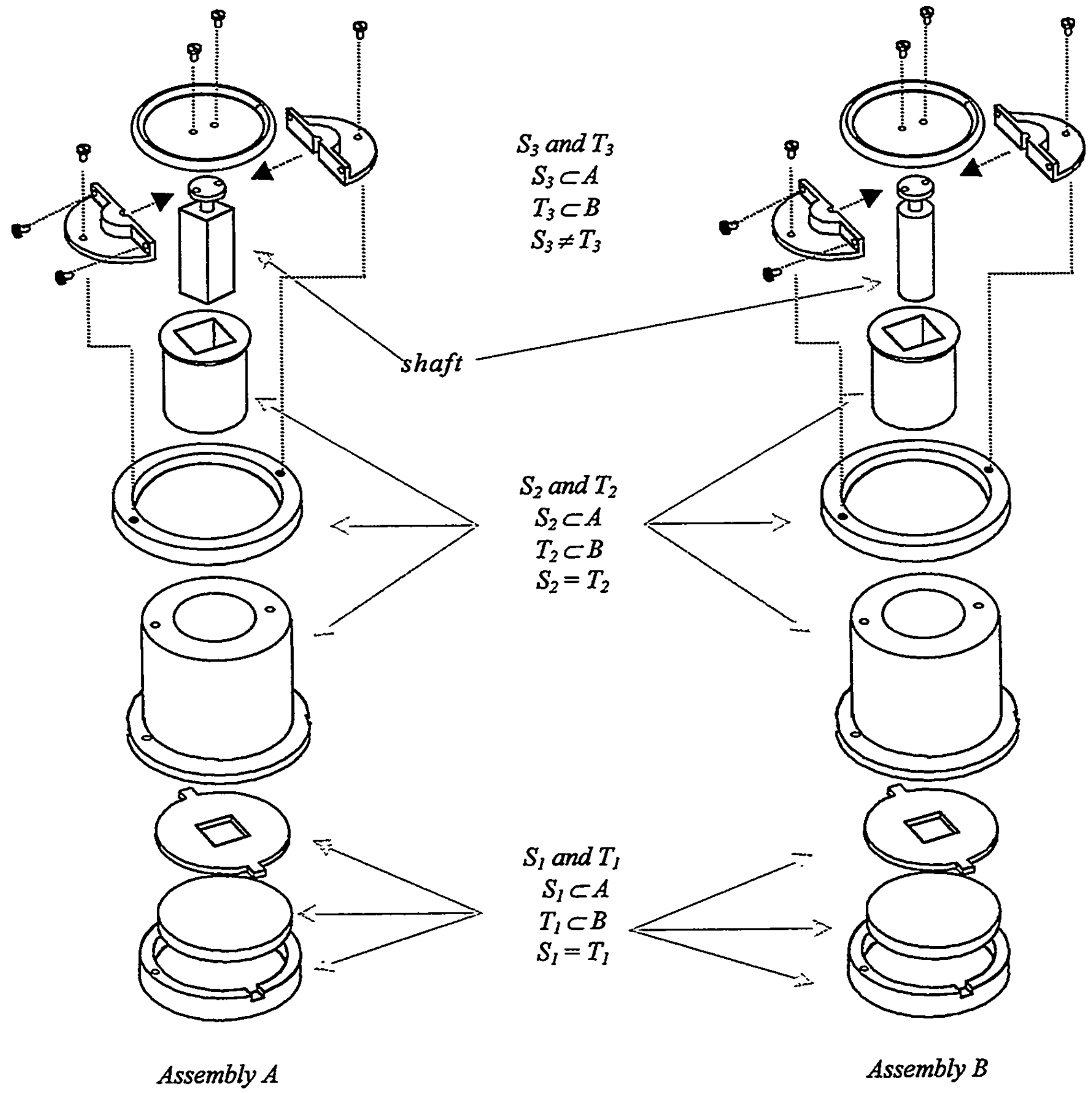


Figure 4.3. Demonstration of propagation effects for planning with subassemblies. Exploded diagrams for Assembly A and Assembly B.

Figure 4.4 shows exploded diagrams of two very similar assemblies, *A* and *B*. The only difference between the two is the shape of the shaft in each. Assemblies *A* and *B* are made from three subassemblies and 6 fasteners. Viewing the diagram from the bottom up, the first three parts make up the first set of subassemblies, S_1 *and* T_1, $S_1 \subset A$, $T_1 \subset B$, $S_1 = T_1$. The next three parts make up the second set of subassemblies, S_2 *and* T_2, $S_2 \subset A$, $T_2 \subset B$, $S_2 = T_2$. All the remaining parts (with the exception of the fasteners) make up the third set of subassemblies, S_3 *and* T_3, $S_3 \subset A$, $T_3 \subset B$, $S_3 \neq T_3$.

Suppose that *Assembly B* is modified at the top-level by changing the shape of a part in T_2 as shown in Figure 4.4. Then the change only affects *B*. However, if the *B* is modified at the subassembly-level (at T_2) then *A* is no longer feasible.

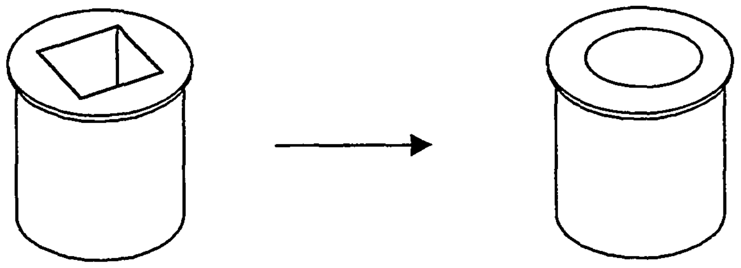


Figure 4.4. Design modification for *Subassembly* T_2.

On the other hand, suppose that *Assembly A* is modified by lengthening the shaft and by cutting a rectangular hole in the plate to slide it into (see Figure 4.5). Then, in this case, the change does not affect *Assembly B* at any level of planning.

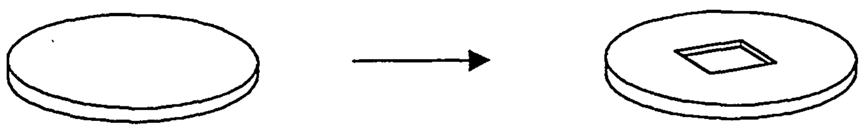


Figure 4.5. Design modification for Subassembly S_1.

4.5 Conflict Resolution between Generative Plans

Rules were incorporated in Archimedes 3.0© to resolve conflicts between top-level and subassembly-level constraints and overrides (when restoring, adding, editing, or activating) at the top-level of the assembly. This section describes the implementation of the constraint and override rules enabling the Archimedes 3.0© System to automatically "reconcile" existing constraints applied to an older version of an assembly to a new version of an assembly.

4.5.1 Constraints

Based on the various constraint and intended purposes, four different methodologies were developed for the implementing the rules. A list of all constraints implemented in Archimedes 3.0© is provided in [1,3]. Table 4.1 lists constraint types that have been added to the Archimedes 3.0© for life cycle assembly processes. In defining the methodologies, the term *current constraint* refers to the constraint that is being added, edited, or activated. The term *existing constraint* refers to the constraint that is in conflict with the current constraint.

The methodologies include:

1. Methods that suspend the existing constraint on conflict give top-level constraints precedence over subassembly constraints during restoration, and new additions or edit changes precedence over existing. The former happens because top-level constraints are restored last. (See Table 4.2.)
2. Methods that suspend the current constraint on conflict give subassembly-level constraints precedence over top-level constraints during restoration, and existing constraints take precedence over new additions or edit changes. The former happens because top-level constraints are restored last. In these cases it is assumed that the designer of the subassembly "knows" best. (See Table 4.3).
3. Methods that union constraints are shown in Table 4.4. Only one constraint of that type to be active.
4. Methods that do no conflict resolution, implement the current constraint addition or change regardless of any existing constraints. These constraints are listed in Table 4.5.

Table 4.1. Additional constraints not provided in [3]. Those marked with a ~ are additional constraints required for life cycle assembly process planning.

Constraint Name	Purpose	Scope
MIN_SIMUL_ LIAISONS	Minimize the use of simultaneous liaison creation. In some contexts, actions are awkward when higher numbers of liaisons are being established by the action	strategic
MIN_REORIENT~	Minimize the number of assembly orientations in the plan.	strategic
REQ_SUCCESS_PART~	Allows the user to specify a part or collection of parts that must be removed from an assembled product. This is especially useful for servicing, repair, and upgrade operations.	strategic
REQ_SUBSEQUENCE_ PARTS~	Allows the user to specify the order in which a subsequence of parts is assembled. This useful for controlling the planning algorithms in circumstances where a specific ordering is already known in advance.	strategic
REQ_RECONFIGURE~	Allows the user to specify reconfigurations of the assembly so that parts can be moved to and from their final configurations.	strategic

Table 4.2. List of constraints and rules that give top-level constraints precedence over sub-assembly constraints. Those marked with * are constraints not available in Archimedes 3.0©.

Constraint Name	Rules
REQ_ORDER_LIAISON	For two REQ_ORDER_LIAISON constraints, if current group intersects with existing second group, and the current second group intersections with existing group, then suspend the existing constraint.
REQ_ORDER_PART	For two REQ_ORDER_PART constraints, if the current group intersects with the existing second group, and the current second group intersects with existing group, then suspend the existing constraint.
REQ_PATHS_AXIAL	If two REQ_PATHS_AXIAL constraints intersect and their required paths are not equal, then suspend the existing constraint.
REQ_STACK	If two REQ_STACK constraints intersect and their required trajectories are not equal, suspend the existing constraint.
REQ_SUBSEQ	If two REQ_SUBSEQ constraints are the same type (assembly or disassembly) and their groups intersect, then suspend the existing constraint.
REQ_TOOL	If two REQ_TOOL constraints primary parts are the same, suspend the existing constraint.
REQ_SUBSEQUENCE_ PARTS*	If two REQ_SUBSEQUENCE_PARTS groups intersect, then suspend the existing constraint.

Table 4.3. List of constraints and rules that give subassembly-level constraints precedence over top-level constraints. Those marked with * are constraints not available in Archimedes 3.0©.

Constraint Name	Rules
REQ_SUBASSY	If two REQ_SUBASSY constraints intersect, but neither is a subset of the other, then suspend the current constraint. If the current REQ_SUBASSY intersects with an existing REQ_SUBASSY_WHOLE, but neither is a subset of the other, then suspend the current constraint.
REQ_SUBASSY_ WHOLE	If two REQ_SUBASSY_WHOLE constraints intersect, but neither is a subset of the other, then suspend the current constraint. If current REQ_SUBASSY_WHOLE intersects with an existing REQ_SUBASSY, but neither is a subset of the other, then suspend the current constraint.

Table 4.4. List of constraints and rules that union top-level and subassembly constraints. Those marked with * are constraints not available in Archimedes 3.0©.

Constraint Name	Rules
REQ_ORDER_FIRST	If the current REQ_ORDER_FIRST intersects with an existing REQ_ORDER_FIRST, suspend the existing constraint. Union current constraint with any existing REQ_ORDER_FIRST constraints and delete the existing constraints.
REQ_ORDER_LAST	If the current REQ_ORDER_LAST constraint intersects with an existing REQ_ORDER_FIRST constraint, suspend the existing constraint. Union the current constraint with any existing REQ_ORDER_LAST constraints and delete the existing constraints.
REQ_SUCCESS_ PART*	Union current constraint with any existing REQ_SUCCESS_PART constraints and delete the existing constraints.

Table 4.5. List of constraints that give top-level precedence and have no conflicts. Those marked with * are constraints not in Archimedes 3.0©.

Constraint Name	Rules
REQ_LINEAR_PARTS	None.
PRH_STATE	None.
PRH_SUBASSY	None.
REQ_CLUSTER	None.
REQ_FASTENER	None.
REQ_LINEAR_ CLUSTER	None.
REQ_PART_SPECIAL	None.
REQ_STAT	None.
MIN_SIMUL_ LIAISONS*	None.
MIN_REORIENT*	None.

4.5.2 Overrides

Rules for resolving conflicts between subassembly overrides and top-level overrides are much simpler than the constraints. When applying the system to the assembly at the top-level, overrides for the subassemblies (parsing the subassembly tree bottom-first, deleting similar overrides at it goes) are automatically loaded. The system then loads the overrides for top-level assembly (deleting similar overrides, in this case any non-top-level overrides of same type for the same part), if any exist. On restoring all top-level overrides from a file, the system removes all top-level overrides and then loads in new top-level overrides (deleting similar overrides).

4.6 Conclusions and Future Work

To the author's knowledge no automatic assembly planner has ever compensated for automatic planning for generative processes. The methodologies presented and implemented in this section are natural progressions towards fully automating generative process planning.

The system has been tested on numerous assemblies and has shown significant increases in efficiencies in planning for assembly upgrades. However, it is difficult to obtain precise measures of efficiency since there many variables affecting the process (e.g., the user of the system, the size of the assembly, the number of initial constraints and overrides to produce a feasible process plan, etc.).

Future work needs to be directed towards the development of algorithms to fully automate the propagation of the assembly constraints and overrides when parts are added to an assembly.

5

Optimizing Search Algorithms

As in initial assembly, the product design and known process constraints are inputs to creating such plans. But for lifecycle assembly planning, the goal of the process is often different. Examples are replacement of particular subassemblies, or least-cost dismantlement and disposal. Furthermore, the disassembly and assembly processes used to meet these goals have important differences from initial manufacturing. In planning for and optimizing lifecycle disassembly sequences, there are substantial technical issues that require fundamental rethinking of the procedural representation mentioned above. That which is addressed in this section is partial assembly (or disassembly). In a partial disassembly process, assembly and disassembly does not always proceed from or result in individual parts. For instance, a service-oriented task may require only partially disassembly of a component, and then subassemblies are placed in it.

In the sections that follow, automatic planning for partial disassembly is discussed. Section 5.1 covers the inclusion of a newly implemented constraint, REQ_SUCCESS_PART(s), into the Archimedes 3.0© System to accommodate for service-oriented assembly processes. The instantiation of a REQ_SUCCESS_PART(s) constraint allows a user to specify a part or collection of parts that must be removed from an assembled product. This forces the planner to stop when none of the desired parts (as specified in the constraint) are left in the assembly. Section 5.2 further describes a computer algorithm and its implementation that optimizes the search strategy for the REQ_SUCCESS_PART(s) constraint. This search strategy uses a hill-climbing technique of the standard A* search to repeatedly probe the subassembly tree, looking for better (less costly) disassembly sequences. An additional algorithm, referred to as the *shortening algorithm,* is also described. The purpose of the shortening algorithm is to shorten the initial sequence by eliminating unnecessary removal of parts not specified in the REQ_SUCCESS_PART(s) constraint set. Finally, Section 5.3 concludes the section with an example application.

5.1 REQ_SUCCESS_PART(s)

As a first approach to service-oriented part removal, the Archimedes 3.0© disassembly planner was employed essentially as-is, by specifying a constraint, which is referred to as REQ_SUCCESS_PART(s), 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, a more general search-optimization strategy was incorporated into Archimedes 3.0©. This strategy uses a hill-climbing variety of the standard A* search to repeatedly probe the subassembly tree, looking for better (less costly) disassembly sequences. Running with a REQ_SUCCESS_PART constraint, and the optimizer enabled, causes Archimedes 3.0© 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 few hundred possible disassembly steps, the execution time required to find the best possible disassembly sequence is prohibitive. This demonstrates that even the best possible sequence is at least two operations deep and 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 3.0© may still be unable, using the general-purpose optimizer, to find that sequence, let alone say that no other sequence is better.

5.2 Shortening Algorithm

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 5.1; 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 3.0© 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 3.0© 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 3.0©, ability to find sequences that do not contain "obviously unnecessary" part removals, a specific module that operates on existing plans to improve their efficiency was implemented. 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. This is accomplished by attempting to indefinitely defer specific removal operations. For the disassembly sequence shown in Figure 1, {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, the planner realizes that it doesn't 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.

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 recomputes the "best plan", and repeats, until it fails to indefinitely defer the cost of any part removal. In this 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.

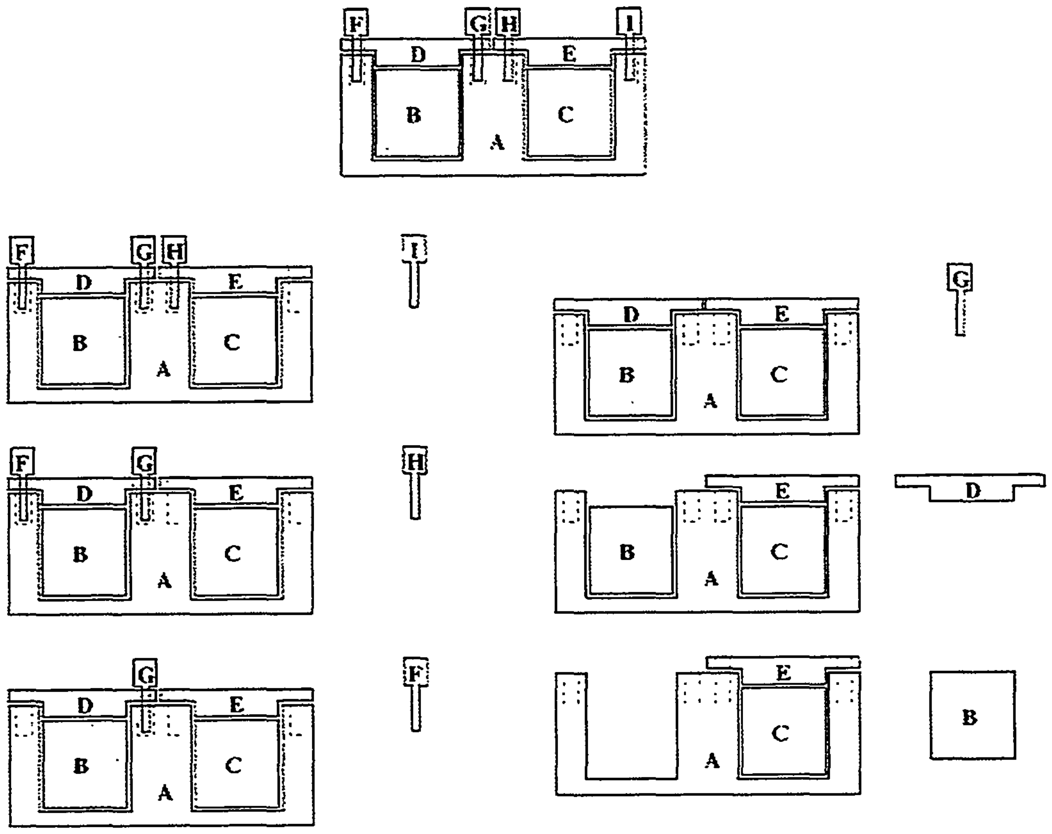


Figure 5.1. Example of an assembly from which a specific part is to be removed.

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 5.1.

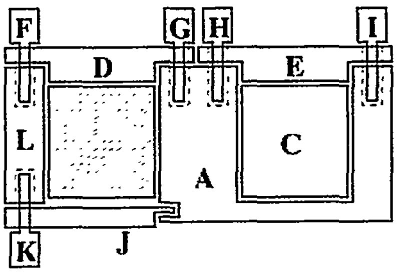


Figure 5.2. Example of assembly where the shortening algorithm is not guaranteed to find an optimally short disassembly sequence.

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 in Figure 5.2, that part B was held within its cavity in part A, not only by the plate D, but also by a plate J, which came into the bottom of B's cavity (rather than the bottom of the cavity being a solid part of A), but was held in place only by a single screw, K. Then the optimal removal sequence would be {K, J, B}. Until the planner found a sequence containing {K, J, B} as a subsequence, the shortening algorithm would not be able to produce {K, J, B} as an alternative.

However, in spite of these limitations, in practice the shortening algorithm does provide an effective way to find a lowest-cost servicing disassembly plan. 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.

5.3 Experiments

The Archimedes 3.0© planner, extended with the constraint system has been applied to a number of actual assemblies from sources in government and industry. To date, the newly integrated shortening algorithm has only been implemented in test case scenarios.

The B61, with improved non-nuclear components, has replaced the B53 in the U.S. stockpile. The scope of the modifications to the B61 requires exhaustive testing to certify the modified bomb's safety, functionality, and reliability.

With the shortening algorithm, it is possible to generate fewest-removed-parts-disassembly plans for easy access to parts that require servicing or replacement on a regular basis, such as the parachute shown in the B61 tail-section in Figure 5.3. As part of routine maintenance, the parachute is regularly removed from the tail-section with a flared cone, which changed the appearance of the weapon somewhat in the rear, where the fins existed before to allow the bomb to spin in a certain way. In Figure 5.3, the picture on the left depicts the tail-section subassembly in its assembled configuration. On the right is a picture showing the tail-section with the parachute partially removed after the shortening was applied.

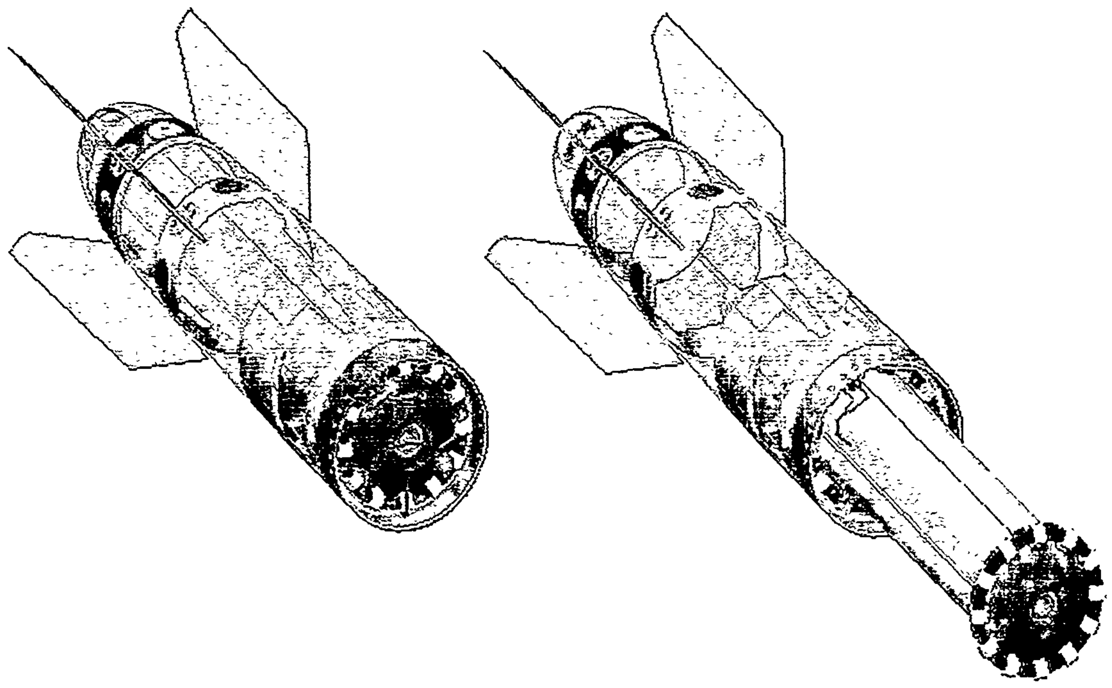


Figure 5.3. The tail-section subassembly of the B61 Bomb.

The tail-section subassembly contains 258 parts described by 17.5Mb of ACIS® data (translated from Pro/ENGINEER®) and slightly over a million facets. Archimedes 3.0© was first applied to the solid model to identify any inconsistencies in the CAD model. This allowed for the detection of critical design flaws to be caught early in the re-manufacturing phase (servicing) and a reduction in scheduling and costs. Next, Archimedes 3.0© was used to test the feasibility of disassembly, checking for geometric accessibility for removal of parts. Since Archimedes 3.0© plans only for straight-line motions, and this assembly contained a number of flexible parts, such as cables, that could not use straight-line assembly motions, the part-mating operations involving those parts were overrode. After loading and faceting all the parts, Archimedes 3.0© took approximately three minutes to find all contacts and produce a disassembly plan.

Intentionally Left Blank.

6

Geometric Simplification

A wide variety of solid modeling packages is currently available which makes it easy for a designer to create sophisticated components and systems in a relatively short time. These packages shorten the initial design process and make modifications relatively simple. The down side to these packages is that they encourage designs with high geometric complexity. This added complexity, while often necessary for function or for manufacture, tends to greatly complicate the analyses that the designer might need to perform on those designs. This includes such analyses as assembly planning.

While the *design for simplicity* paradigm traditionally formulated and encoded at Sandia facilitates eliminating those design features which are not strictly necessary for manufacture or function, this paradigm made only small progress towards removing irrelevant geometry for a particular analysis engine, and none at all for existing designs.

6.1 Facetted Data

The first approach towards geometric simplification addressed facetted data. Since the size of the facetted representation of an assembly can get quite large for large, complicated assemblies, the capability for an Archimedes 3.0© user to re-specify the facetting parameters used on the loaded assembly was added. This has the effect of speeding up the contact and collision analysis to a small extent, but more importantly, improves the rendering speed for plan playback, at the cost of somewhat cruder animation, and possibly requiring extra mating OVERRIDES for certain tight-tolerance insertions.

Next, a simple, face-to-face converter was written that converts STL facetted data to the ACIS solid model format. Algorithms were written and implemented in Archimedes 3.0© that extended this facility to sew together geometrically identical but topologically separate edges of adjacent faces and merge faces which share the same plane. The ability to export the resulting connected components to file, one component per file, was added. This provided the user limited ability to separate STL-format assemblies into their individual parts.

In addition to the above, algorithms were developed and implemented to provide functionality that takes a facetted part, and fits cylindrical patches to its faces, as appropriate. This facility then replaces each set of fitted facets with the cylindrical patch. The cylinder fitting operation works for sets of faces that approximate a full cylinder with ends which are planar cuts or partial cylinders (e.g., half-rounds) with ends that are right-planar cuts. Next, means to adjust the perimeter of the cylindrical patch were provided to better approximate the perimeter of the set of fitted faces.

6.2 Feature Reduction in CAD Models

Work conducted under the "Feature Reduction of Geometric Solid Models for Analysis Tools" [4] was partially combined with the efforts pursued under this LDRD to simplify geometric complexities of very large assemblies for efficient assembly analysis.

A suite of geometric simplification heuristics was developed which enable a designer to quickly produce a simplified design which eliminates those features of the original design which are not pertinent to a given analysis [4]. Secondary efforts focused on managing the complexity of solid models which have been completely or partially converted to a faceted (polygonal, planar faces only) representation. The simplest feature addressed was a simple hole. A simple hole meaning a hole whose cross section is a circle or ellipse. The hole could be a "blind hole", that is a hole that is a dead end, such as a crew hole or it could be a hole that goes all the way through some material, a "through hole", that one might pass a rod or pin through. By examining the objects that contained these holes, it was clear that a simple hole in an object generated a large number of polygons. Indeed, punching a hole through an object can generate around 30 polygons in each single polygon the hole pierces and the walls of the hole can be comprised of about 50. These numbers are controllable through parameter settings in the facetter but these numbers are not atypical.

As an example of the difference it can make, the Archimedes 3.0© System was applied to an assembly comprised of some printed circuit boards housed by aluminum frame (see Figure 3.1). Applying the remove hole simplifier to one of the circuit boards cut the polygon count of the circuit board to one sixtieth, from 24,000 to 400, without changing the geometry necessary for Archimedes 3.0© to do its work. The contact analysis portion of the Archimedes 3.0© run dropped from 4 hours to 6 minutes. Obviously, not all cases will be this well suited to hole removal, but it was a real case. In another case, applying the hole removal simplifier to a fire set housing, the polygon count dropped 30%.

Intentionally Left Blank.

7

Summary

Significant progress in reducing the four main resources (memory utilization, contact analysis computation time, planning time and geometric simplification) identified in the original proposal as barriers to application of the Archimedes 3.0© System to Very Large Assemblies was made. The approach taken was to start with the Archimedes 3.0© System and (1) Refine algorithms and data-structures to make the planner capable of handling thousands of parts using realistic amounts of time and memory. (2) Provide more flexible means of interacting with subassemblies and also with assemblies which change over time (generative design). (3) Improve the robustness of the code and algorithms to increase the breadth of scope, as well as the size, of assemblies which Archimedes 3.0© can handle. And (4), incorporate the ability to reduce the geometric complexity of assembly elements for improving planning and playback time for very large assemblies. A summary of the experimental results is presented below.

After restructuring data structures for improved contact analysis capabilities, the addition of a single OVERRIDE or CONSTRAINT, involving for example, two of 369 parts, contact analysis took only about 3 seconds, and almost 98% improvement for that assembly. For a more complicated assembly, or one with more parts, the savings would be even greater. This a very significant from the user's perspective, because users of Archimedes 3.0© often react to problems in the produced assembly plan like modern programmers; they fix one or two problems, then rerun the planner. So, reducing the contact analysis time on second and subsequent runs reduces the amount of user time required very dramatically.

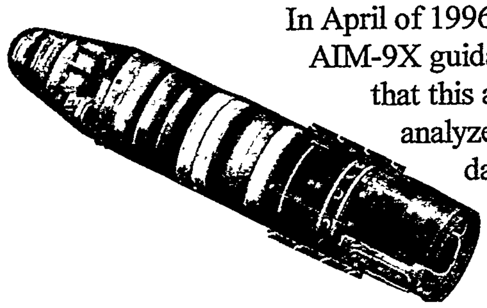
Once the initial contact analysis for an assembly has been achieved, the improved scheme reduces the time required to update the contact analysis after adding a single OVERRIDE or selected CONSTRAINT from the full amount required by the initial analysis (as much as several hours, for assemblies with a few hundred parts) down to a few seconds to a few minutes, depending on the complexity and numbers of the individual parts involved in the introduction of OVERRIDES. Since the ability to save and retrieve the contact information has also been added, this means that the full, all parts against all other parts contact analysis need only be performed once for any given assembly.

By restructuring some of the graphical output code, we were able to achieve a 93% reduction in the required storage for each part in an assembly (from about 2 Megabytes/part down to about 140 Kilobytes/part) for the electronic subassembly array shown in Figure 3.1. In terms of assembly size limitations, this raises the bar from a previous limit of about 600-8000 parts for assemblies running on a 32-bit SGI, to a level in excess of 10,000 parts. Effectively, required memory has ceased to be a limitation on the size of assemblies.

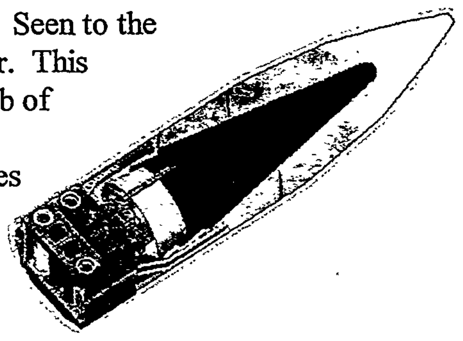
Capabilities enabling the user to more quickly and easily interact with large assemblies, in terms of providing mating and manufacturing OVERRIDES, and assembly CONSTRAINTS were added. The user can now interact with an assembly at the subassembly level, establishing appropriate CONSTRAINTS and OVERRIDES, and have those CONSTRAINTS and OVERRIDES show up at the full assembly level. To the author's knowledge no automatic assembly planner has ever compensated for automatic planning for generative processes. The methodologies presented and implemented in this Section 4 are natural progressions towards fully automating generative process planning. The system has been tested on numerous assemblies and has shown significant increases in efficiencies in planning for assembly upgrades. However, it is difficult to obtain precise measures of efficiency since there many variables affecting the process (e.g., the user of the system, the size of the assembly, the number of initial CONSTRAINTS and OVERRIDES to produce a feasible process plan, etc.). Future work needs to be directed towards the development of algorithms to fully automate the propagation of the assembly CONSTRAINTS and OVERRIDES when parts are added to an assembly.

A realistic assessment of the time taken by the planner on various types of planning tasks indicated that the planner was, realistically, more than adequately fast for the basic assembly and disassembly planning processes. Specifically, at the current time, for a particular assembly of approximately 350 parts, we can find a single assembly strategy in just over two minutes. Since the growth in planning time is approximately quadratic, and assembly of 2,000 parts would be expected to take roughly 36 times as long, or slightly in excess of an hour, not counting contact-analysis (which only needs to be performed once). Where we found a major shortfall was in the process of optimizing assembly and disassembly strategies. While the basic planner just takes apart an assembly once, and then stops, when we begin optimizing, we find ourselves calling the planner many, many times, looking for a "best" assembly/disassembly plan. Each call to the planner finds a single path (sequence) through the disassembly tree for an assembly, and assigns a cost to that path. The difficulties are that the state space for this search is extremely large, and that the general purpose search heuristics are not effective enough at restricting the search, for the search engine to locate an optimal strategy for assemblies of even as few as 25 parts. We have implemented special purpose search strategies that, for specific types of disassembly tasks, use a simulated-annealing-like process to locally optimize disassembly plans provided by the general-purpose planner. This enables Archimedes 3.0© to find optimal disassembly plans in a few minutes in most cases, though it is still beyond the capability of Archimedes 3.0© to prove, generally, that no better sequences are possible. We can run this planner on assemblies that are much larger than the 20-25 part limit that was the previous limit, and brings this kind of planning into the realm of possibility for assemblies of hundreds of parts.

As an example geometric simplification (hole simplification) can make, the Archimedes 3.0 System was applied to an assembly comprised of some printed circuit boards housed by aluminum frame (see Figure 3.1). Applying the remove hole simplifier to one of the circuit boards cut the polygon count of the circuit board to one sixtieth, from 24,000 to 400, without changing the geometry necessary for Archimedes 3.0© to do its work. The contact analysis portion of the Archimedes 3.0© run dropped from 4 hours to 6 minutes. Obviously, not all cases will be this well suited to hole removal, but it was a real case. In another case, applying the hole removal simplifier to a fire set housing, the polygon count dropped 30%.



In April of 1996, the Archimedes assembly analysis code was applied to the AIM-9X guidance section (left). The challenge in this demonstration was that this assembly was much larger than any that had been previously analyzed. With 471, parts, 170 Mb of Pro/E data, 55 Mb of ACIS data, and 800,000 facets was at least ten times as large as any assembly previously planned for automatically. This feat was to assembly planning what the TeraFlop was to the world of supercomputing.



Early in 1999, the Archimedes team set yet another record. Seen to the right, is a target array assembly used as a missile interceptor. This assembly contains 1750 parts, 140 Mb of Pro/E data, 18 Mb of ACIS data, and 1,800,000 facets. Using the technologies Developed under the purview of this LDRD, the Archimedes analysis surpasses the previously set record three fold. Time required to load and perform contact analysis is 10 minutes. The time to plan an initial sequence is 8 minutes while subsequent planning takes on the order of 1 to 2 minutes.

Intentionally Left Blank.

8

References

1. Terri L. Calton. On the automatic generation of plans for life cycle assembly processes. *Sandia Report. SAND99-2808*. January 2000.
2. Terri L. Calton and Ralph R. Peters. Human assisted assembly processes. *Sandia Report. SAND99-2809*. January 2000.
3. R.E. Jones, R.H. Wilson and T.L. Calton. On constraints in assembly planning. *In IEEE Int'l Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 849-863, 1998.
4. Peter Watterberg. Geometric simplification of analysis models. Sandia Report. *SAND 99-3117*. December 1999.

Intentionally Left Blank.

Distribution:

1	MS	0188	LDRD Office, 4001
1		0507	C. M. Hart, 2600
1		1002	P. J. Eicker, 15200
1		1010	M. E. Olson, 15222
1		1006	P. Garcia, 15271
10		1008	T. L. Calton, 15221
6		1008	S. Blauwkamp, 15200 Library, 15221
1		9018	Central Technical Files, 8940-2
2		0899	Technical Library, 4916
1		0612	Review & Approval Desk, 4912 For DOE/OSTI
1		0161	Patent and Licensing Office, 11500