TOWARDS AUTOMATIC PLANNING FOR MANUFACTURING GENERATIVE PROCESSES

TERRI L. CALTON
Intelligent Systems and Robotics Center
Sandia National Laboratories*
Albuquerque, NM 87185-1008

Abstract

Generative process planning describes methods process engineers use to modify manufacturing/process plans after designs are complete. A completed design may be the result from the introduction of a new product based on an old design, an assembly upgrade, or modified product designs used for a family of similar products. An engineer designs an assembly and then creates plans capturing manufacturing processes, including assembly sequences, component joining methods, part costs, labor costs, etc. When new products originate as a result of an upgrade, component geometry may change, and/or additional components and subassemblies may be added to or are omitted from the original design. As a result process engineers are forced to create new plans. This is further complicated by the fact that the process engineer is forced to manually generate these plans for each product upgrade. To generate new assembly plans for product upgrades, engineers must manually re-specify the manufacturing plan selection criteria and re-run the planners. To remedy this problem, special-purpose assembly planning algorithms have been developed to automatically recognize design modifications and automatically apply previously defined manufacturing plan selection criteria and constraints.

Keywords: Assembly Planning, Process Planning

1. Introduction

This paper introduces methodologies that are natural algorithmic progressions of an automated assembly planner towards fully automating generative process planning. Generative process planning describes the methods process engineers use to modify manufacturing (or process) plans after a design is complete. Section 2 introduces an automatic assembly planning framework used as the foundation for automating generative process planning and provides an overview of motivational factors promoting the development of automatic generative process planning techniques. Section 3 places the assembly planner in the context of generative process planning. It further introduces geometric problems associated with top-level assembly planning and special-purpose routines implemented within the assembly planner to solve those problems. Methods are presented for saving, restoring, and propagating subassembly analyses for top-level assembly analysis. Section 4 describes the implementation of constraint rules enabling users to automatically "reconcile" existing constraints applied to an older version of an assembly to a new version of an assembly. Finally, Section 5 concludes the paper and presents future research areas.

2. Background and Motivation

The approach taken to manufacturing planning is obviously critical to the design, implementation and performance of automatic generative process planning. At Sandia National Laboratories, researchers have developed an automatic assembly planner, called Archimedes© [1]. This system was used as the basis for realizing automatic generative process planning.

2.1 Manufacturing Planning Approach

Archimedes is a constraint-based interactive assembly planning software tool used to plan, optimize, simulate, visualize, and document sequences of assembly. Given a CAD model of the product, the program automatically finds part-to-part contacts, generates collision-free insertion motions, and chooses assembly order. Disassembly operations are generated using the Non-Directional Blocking Graph approach discussed in [2]. A graphics workstation's hardware Z-buffer is used to quickly find collisions between complex facetted models. The search space implemented in the system is an AND/OR graph of subassembly states [3] and the operations used to construct them from smaller subassemblies.

During system application, 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 graphical interface to place geometric overrides and manufacturing constraints on

* 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.

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

the valid assembly sequences, such as defining subassemblies, requiring that certain parts be placed consecutively with or before other parts, declaring preferred directions, etc.

Two types of constraints on assembly plans are utilized by the system: strategic and tactical. Strategic constraints apply to the entire assembly and its plan, while tactical constraints only apply to certain subsets of parts. The constraint framework provides a library [4] of constraint types from which a user can instantiate on the assembly plan. This framework provides the underlying mechanics towards assembly optimization and lends itself towards automatic planning for manufacturing generative processes.

2.2 Motivational Applications

The Archimedes 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 bomb. The B61, with improved non-nuclear components, has replaced the B53 in the U.S. stockpile. The scope of modifications made to the B61 requires exhaustive testing to certify the modified bomb's safety, functionality, and reliability. In an early experiment, Archimedes was applied to the B61 center-case. It was estimated that 2.5-3 person months were required to manually create training documentation for retrofit operations using a commercial animation package. In an effort to reduce this time, Archimedes was applied to the center-case assembly. The experiment showed that there were many assembly planning issues associated with CAD revisions that went beyond the existing capabilities. For instance, the first step required to apply Archimedes was to translate the CAD data to the ACIS format. Initially, the entire center-case assembly, containing 547-parts, was selected for analysis; however, due to CAD translation problems a 303-part subassembly was exercised during the experiment. Effectively, the original design was modified by removing parts.

The planner was first applied to the original (larger) solid model to identify 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, the system was used to test feasibility of disassembly, checking geometric accessibility for part removal. Since the planner plans only for straight-line motions, and this assembly contained numerous 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, since the existing planning algorithms could not reconcile the differences (part count, geometry, constraints and overrides) between the two assemblies automatically.

This same problem was inherent in the application of the planner to the B61 nose assembly. Unlike the center-case assembly, the nose assembly went through several revisions before finalization. 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, the planner had to be applied to each subassembly even if they were identical. Further, when the two fully analyzed subassemblies were brought together to form a single assembly, instantiation of all constraints had to be repeated manually instead of automatic inheritance.

3. Generative Process Planning Issues

Section 2.2 identifies three fundamental problems inherent in automatic generative process planning. These problems are not restricted to Archimedes. 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. Section 3 is devoted towards solving these problems. The underlying principles, as they relate to automatic assembly planning, are discussed, and solutions to each are provided.

3.1 Re-design

Two fundamental issues associated with assembly design modification are geometry and function. For purposes of assembly planning only the geometry is discussed. There are three geometry-related design modification principles for any given assembly. An assembly may be modified by (1) removing part(s), (2) changing the shape of the part(s), (3) adding parts(s), or by any combination of the three. From an assembly planning perspective, part removal is the simplest form of modification to deal with, while the addition of parts is the most difficult.

To address the first, the removal of parts, a geometric override was added to the original Archimedes 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 the planner, 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.

When the user loads in the assembly, the constraint 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' contact analysis routines, which automatically check contacts between parts, is used. If the re-design alters the contacts between parts, the user is automatically informed and given the opportunity to address the issue. The same is true for previously defined constraints and overrides.

The third issue, the addition of parts, is the most difficult issue. Because the assembly planner 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 planner can not plan for assembly 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.

3.2 Planning with Subassemblies

This section deals with generative process planning principles, 1 and 2 above. It is pointed out in the previous section that the planner 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 towards solving this problem was to incorporate save and restore routines for the constraints and overrides resulting from the application of Archimedes to the subassemblies, which would automatically load when Archimedes 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 addresses the conflict resolution issues. Here, methodologies incorporated into the planner for automatically propagating information generated at the subassembly-level to the top-level are presented.

3.2.1 Automatic Propagation

When the planner 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. 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 take precedence over overrides made locally to the subassembly.

3.2.2 Demonstration of Propagation Effects

To illustrate the propagation of design modifications imposed at the subassembly-level for later use in planning at the top-level, conceptual designs of two similar assemblies, *A* and *B*, are shown in Figure 1.

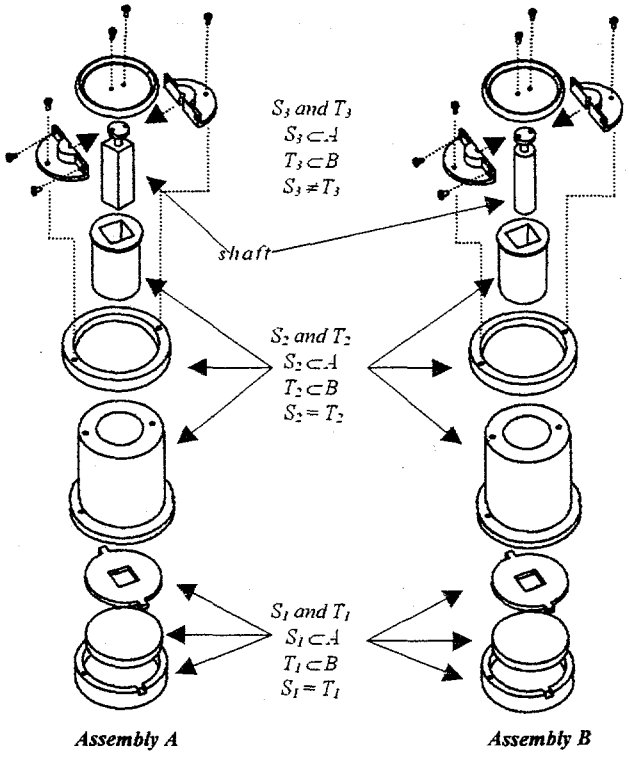

Figure 1. Propagation effects for planning with subassemblies.

The only difference between the two is the shape of the shaft. Assemblies A and B are composed of 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 B is modified at the top-level by changing the shape of a part in T_2 as shown in Figure 2. 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.

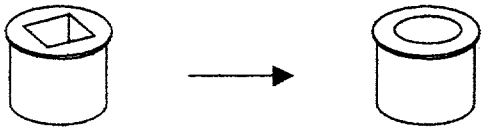

Figure 2. Design modification for *Subassembly* T_2.

On the other hand, suppose A is modified by lengthening the shaft and by cutting a rectangular hole in the plate to slide it into (see Figure 3). In this case, the change does not affect B at any level of planning.

Figure 3. Design modification for Subassembly S_1.

4. Conflict Resolution

Rules were incorporated in Archimedes 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 planner to automatically "reconcile" existing constraints and current constraints.

4.1 Constraints

Based on the various constraints and intended purposes, four different methodologies were developed for implementing the rules. In defining the methodologies for automatic generative process planning, 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. Tables 1-4 present the methodologies.

Table 1. Methods that suspend existing constraints on conflict give top-level constraints precedence over subassembly constraints during restoration, and new additions or edit changes precedence over existing.

Constraint	Definitions and Rules
req_order_ liaison	Require some ordering between 2 or more liaison creations; typically stated in a Boolean form such as 1 (2 and 3), or as a set of such Boolean statements involving many liaisons [5,6]. **Rule:** For 2 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	Require ordering between particular part insertions. **Rule:** 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	Require that each assembly action be along one of the 6 coordinate directions of a given coordinate system, or a selected subset of these 6 directions. **Rule:** If two REQ_PATHS_AXIAL constraints intersect and their required paths are not equal, then suspend the existing constraint.
req_stack	Specifies a set of parts to be assembled one at a time in a given direction. **Rule:** If 2 REQ_STACK constraints intersect and their required trajectories are not equal, suspend the existing constraint.
req_sub seq	Require that a particular assembly subsequence be used somewhere in the plan. This might be invoked because the sequence is particularly efficient or reliable. The front-fill then back-fill subsequences of [7] are relatively complex examples. **Rule:** If two REQ_SUBSEQ constraints are the same type (assembly or disassembly) and their groups intersect, then suspend the existing constraint.
req_tool	Requires that a collision-free placement of a given tool use-volume must exist in the assembly during a certain operation. See [8] for more details. **Rule:** If two REQ_TOOL constraints primary parts are the same, suspend the existing constraint.
req_sub-sequence_ parts	Allows a user to specify the order in which a subsequence of parts is assembled. **Rule:** If two REQ_SUBSEQUENCE_PARTS groups intersect, then suspend the existing constraint.

Table 2. Methods that suspend current constraints on conflict give subassembly-level constraints precedence over top-level constraints during restoration, and existing constraints take precedence over new additions or edit changes.

Constraint	Definitions and Rules
req_subassy	Require a particular subassembly be used [9]. **Rule:** 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	The same as REQ_SUBASSY, but tells the planner in addition not generate a plan to construct subassembly. **Rule:** 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 3. Constraints/rules unioning top-level and subassembly constraints.

Constraint	Definitions and Rules
req_order_ first	Require that an assembly plan start with a given part. **Rule:** 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	Requires that a certain part or set of parts be placed last. **Rule:** 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	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,/upgrade **Rule:** Union current constraint with any existing REQ_SUCCESS_PART constraints and delete the existing constraints.

Table 4. Methods that do no conflict resolution implement the current constraint addition or change regardless of any existing constraints.

Constraint	Definitions
req_lin_part	Requires that parts be inserted one at a time [10].
prh_state	Do not allow the assembly to enter a given state [11].
prh_subassy	Prohibit use of certain subassemblies, or possibly any subassembly containing certain part combinations.
req_cluster	Require set of parts be added to the assembly consecutively, (without interruption by other parts [12]).
req_fastener	Require certain parts be treated as fasteners[13,14].
req_linear_ cluster	A combination of REQ_CLUSTER and REQ_LINEAR_SUBSET.
req_part_sp ecial	Any special-purpose part constraint, such as those dealing with liquids, springs, snap-fit parts, etc
req_stat	Requires a set of parts to be in the stationary subassembly when mated with any of another set.
min_simul-liaisons	Minimize use of simultaneous liaison creation. In some contexts, actions are awkward when higher numbers of liaisons are established by an action.
minreorient	Minimize the number of assembly re-orientations.

4.2 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).

5. Conclusions and Future Work

A synopsis of problems and partial solutions associated with automating generative process planning has been provided. While many assembly planners exist, Archimedes is the only known system, which truly generates assembly plans automatically; and, to the author's knowledge no automatic assembly planner has ever compensated for automatic planning for generative processes. The methodologies presented in this paper are natural algorithmic progressions of the Archimedes system 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 and the results are proportional with task difficulty. For example, it took approximately 1/2 day to analyze and instantiate constraints on the nose section of the B61 mentioned in Section 2.2. With the generative process planning capabilities, this time was reduced to planning time only, approximately 1 minute. It is difficult to obtain precise measures of efficiency since there many variables affecting the process (e.g., user experience, user familiarity with the assembly, assembly size, number of initial constraints and overrides, etc.).

Three geometry-related design modification principles were presented: modification by (1) removing part(s), (2) changing the shape of the part(s), (3) adding parts(s), or by any combination of the three. Automatic planning algorithms for (1) and (2) were presented and tested. 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.

References

[1,4] R. E. Jones, R. H. Wilson, and T. L. Calton. Constraint-based interactive assembly planning. In *Proc. IEEE Transactions On Robotics and Automation,* 1998.

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

[3] P. Langley. Systematic and nonsystematic search strategies. In *Artificial Intelligence Planning Systems: Proc. Of the First Intl. Conf.,* 1992.

[5,12] N. Boneschanscher and C. J. M. Heemskerk. Grouping parts to reduce the complexity of assembly sequence planning. In *Information Control Problems in Manufacturing Technology 1989: Selected Papers from the 6th IFAC/IFIP/IFORS/IMACS Symposium.* E. A. Puente and L. Nemes, Eds. New Yourk: Paergamon Press, 1989, pp. 233-238.

[6] J. D. Wolter, S. Chakrabarty, J. Tsao. Mating constraint languages for assembly sequence planning. In *IEEE Trans. Robot. Automat.*, to be published.

[7,11] D. F. Baldwin, T. E. Abell, M.-C. M. Lui, T. L. De Fazio, and D. E. Whitney. An integrated computer aid for generating and evaluating assembly sequences for mechanical products. *IEEE Trans. On Robotics and Automation,* 7(1):78-94, 1991.

[8] R. H. Wilson. Geometric reasoning about assembly tools. Tech. Report SAND95-2423, Sandia Labs, 1996.

[9] 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.

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

[13] L. S. Homem de Mello and A. C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. In *IEEE Trans. Robot, Automat.*, vol. 7, pp. 228-240, Apr. 1991.

[14] J. M. Miller and R. L. Hoffman. Automatic assembly planning with fasteners. In *Proc. IEEE Int'l. Conf. On Robotics and Automation.* 1989, pp 69-74.