To appear in *IEEE Transactions on Robotics and Automation*

On Constraints in Assembly Planning

Rondall E. Jones
Intelligent Systems and Robotics Center
Sandia National Laboratories
Albuquerque, NM 87185-1008

Randall H. Wilson*
Manufacturing Systems Technology Division
Eastman Kodak Company
Albuquerque, NM 87112-1881

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

Abstract

Constraints on assembly plans vary depending on product, assembly facility, assembly volume, and many other factors. Assembly costs and other measures to optimize vary just as widely. To be effective, computer-aided assembly planning systems must allow users to express the plan selection criteria that apply to their products and production environments.

We begin this article by surveying the types of user criteria, both constraints and quality measures, that have been accepted by assembly planning systems to date. The survey is organized along several dimensions, including strategic *vs.* tactical criteria; manufacturing requirements *vs.* requirements of the automated planning process itself; and the information needed to assess compliance with each criterion. The latter strongly influences the efficiency of planning.

We then focus on constraints. We describe a framework to support a wide variety of user constraints for intuitive and efficient assembly planning. Our framework expresses all constraints on a sequencing level, specifying orders and conditions on part mating operations in a number of ways. Constraints are implemented as simple procedures that either accept or reject assembly operations proposed by the planner. For efficiency, some constraints are supplemented with special-purpose modifications to the planner's algorithms. Fast replanning enables an interactive plan–view–constrain–replan cycle that aids in constraint discovery and documentation. We describe an implementation of the framework in a computer-aided assembly planning system and experiments applying the system to a number of complex assemblies, including one with 472 parts.

DISCLAIMER

DISCLAIMER

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

1 Introduction

Constraints on assembly plans come from a wide variety of sources. Design requirements, part and tool accessibility, assembly line and workcell layout, requirements of special operations, and even supplier relationships can determine which orders of assembly are feasible. Among feasible orderings, a similarly diverse set of quality measures determine which is preferred or optimal. Together, the set of hard constraints and quality measures comprise a complex set of *criteria* that real assembly plans must satisfy. Computer-aided assembly planning systems promise to help product and assembly system designers manage this complexity and choose good assembly sequences.

However, there are so many types of assembly criteria that it is impractical for a single program to encode them all, and in many cases they are company- or product-specific.[1] Hence a practical assembly planning system must have the ability to manage assembly criteria in a general way to find a feasible and best assembly sequence. In response to this apparent need, many types of criteria have appeared in the literature on automated assembly planning or have been implemented in experimental systems. However, the criteria often occur stated in different ways and in a variety of forms and contexts. In this article, we survey assembly constraints and quality measures and present a new framework to support constraints in interactive assembly planning.

We survey the assembly planning literature to identify, synthesize, and organize common criteria on assembly plans. We focus on *what* criteria appear in the literature, rather than *how* they have been implemented. Implicit in our survey is the assumption that if a criterion has appeared in the literature then it is either useful or implementable—either of which makes it interesting when considering an assembly planning implementation. We group related criteria and categorize them along several dimensions, including strategic *vs.* tactical criteria; manufacturing requirements *vs.* requirements of the automated planning process itself; and the information needed to assess compliance with each criterion. The latter strongly influences the efficiency of planning, because criteria requiring only single actions to assess compliance are more amenable to typical state-space search and optimization methods.

We then narrow our focus to hard constraints on assembly plans. We present a framework for representing and reasoning about assembly constraints of many types. The user chooses constraints from a library of standard constraint types, with a simple graphic interface for defining the specifics of the constraint. Constraints are implemented as *filters*, which are simple procedures that either accept or reject assembly operations proposed by the planner. Any constraint that can be encoded as a filter can be added to the constraint library in a straightforward way. Some constraints are supplemented with special-purpose modifications to the planner's algorithms for greater efficiency. We have been able to encode a large majority of the constraints from our survey in this framework.

The constraint framework has been implemented and tested extensively in the Archimedes assembly planning system [15]. In a typical interaction, the system generates and animates a plan, the user adds constraints that need to be satisfied by the plan, asks for a new plan, and repeats the process until a satisfactory plan is found. Our users have found this interactive constraint discovery and planning process to be very natural and productive. In fact, we view the novel form

[1] For example, one of our customers requested a constraint that would model solder possibly dripping on fragile parts. Another requested that our system model the effects of different assembly sequences on tolerances when welding in a certain fixture.

of interaction enabled by the constraint system to be one of the main contributions of this article. We describe our tests of the user constraint system on a number of complex industrial assemblies, including one with an order of magnitude more parts than has been previously demonstrated.

The rest of this paper is organized as follows. The next section presents our survey approach and introduces a number of terms used to define and discuss the criteria. In section 3 we give a short systematic name to each criterion and briefly define it, with references to representative papers in which it appeared. Section 4 then discusses the results of the survey in detail, particularly with regard to our categorizations of criteria.

We then proceed to describe our constraint framework and implementation. Section 5 presents an overview of our framework, including a comparison to previous approaches for representing and reasoning about assembly constraints. Section 6 gives a more detailed view and discusses several important features. Section 7 describes experience with our implementation of the framework and several complex assemblies. Finally, Section 8 concludes and describes future work.

2 Preliminaries

This section presents our survey approach and introduces a number of terms used to define and discuss the criteria in following sections.

2.1 Approach

Our approach has the following motivations and guiding principles:

Inclusion: First, our goal is to provide perspective on how a large variety of assembly criteria, and their potential implementations, interrelate. When in doubt, we err on the side of inclusion. We include all criteria mentioned in the assembly planning literature with which we are familiar. In addition, we include criteria that have arisen in discussions with industrial assembly personnel, or with which we have experience from our own work, or which arose during the survey as seeming natural and potentially useful. However, no claim is made that any of these latter criteria are new or novel. Finally, we include a number of constraints that occur mainly as limitations of automated assembly planning systems. Although these last constraints emanate from an artificial source, they are important to document nonetheless.

Exclusion: On the other hand, this is not an attempt to catalog all *possible* assembly criteria. For example, subassemblies and clusters are two similar but different concepts. Although an engineer might require or prohibit use of a particular subassembly, only *requiring* a cluster makes sense. Thus, the logically possible constraint of *prohibiting* a *cluster* does not seem useful, and is not included.

In the literature, criteria are often discussed together with features of the computer programs which implement them. Such software features may superficially seem like criteria, when they are actually just "housekeeping" functions. For example, a feature to DELETE A STATE [1] represents a constraint, because it rules out certain assembly sequences. The related software feature of UNDELETE A STATE just clears a previously selected constraint, and thus would not be included here. Similarly, a state transition diagram may include redundancies which need to be removed [1],

but the removal process does not constitute a constraint on the assembly plan or the assembly planner.

Finally, note that a number of the criteria could be naturally used as "suggestions" to guide the system in a general way or to help the planner find a good plan more quickly. We originally listed suggestions separately in [13], but most were just modifications of the corresponding constraints. We are investigating the use of suggestions or guidance in our assembly planning implementation, but we do not include suggestions as separate criteria here.

Grouping: Often very similar ideas can be expressed quite differently. For example, "prohibiting a plurality of unconnected subassemblies" [6] and "deleting branched assembly lines" [2] are the same as requiring "linear" sequences [29] (see definitions below). As a result, we arbitrarily choose the latter terminology for inclusion. Similarly, a quality measure that minimizes parallelism in a plan is very close to the concept of maximizing linearity, so we again include only one. Some decisions we have made to combine criteria into classes may arguably be incorrect, given sufficient interest in the details of variants in the class. We have tried to produce a manageable size list, trading off loss of detail with clarity of presentation.

Combining criteria: Constraints can be combined using logical conjunctions, disjunctions, and negations, while quality measures may be combined using arithmetic operators, giving weightings to various measures. We do not delineate which criteria may be combined with which others and in what ways. Instead, we have attempted to identify the basic criteria that might be the terms in such composite criteria. Of course, combinations of criteria must be considered carefully when implementing an assembly planning system.

2.2 Terms and Abbreviations

The following terms are used in the later sections of this article. For terms that appear commonly in names of criteria, we give the abbreviation.

AND/OR graph: a commonly used representation of a set of assembly sequences, listing subassemblies and actions that create larger subassemblies from smaller ones [12].

assembly: a set of parts, in given relative positions.

assembly action: (abbreviated as ACTION) any single operation of bringing together parts or subassemblies, or of moving parts or subassemblies. It is a more general term than "insertion."

cluster: a group of parts to be assembled in uninterrupted sequence, but for which the sequence is not specified [4]. For example, all the bolts holding a lid to an assembly might be defined to be a cluster.

connected: a subassembly is connected if its graph of parts and liaisons is connected.

insertion: an action that mates two subassemblies.

liaison: a relationship between two parts which are touching or effectively touching, whether physically attached or not [5].

linear: An insertion is linear if the moving subassembly contains only one part [29]. A plan is linear if all of its insertions are linear.

monotone: An action is monotone if all parts involved end the action in their relative positions in the final product [29]. A plan is monotone if all of its actions are monotone. In a monotone plan, parts never take temporary positions except while they are being mated. Hence a monotone assembly plan for an n-part assembly has exactly $n-1$ insertions.

stable or stability: (abbreviated STAB) resistant to unwanted change due to effects of gravity, motion, etc.

state: the parts of an assembly, in a certain relation to one another, constituting a stage in assembly.

state transition diagram: a commonly used representation of a set of assembly sequences, listing states and the feasible transitions between them.

subassembly: (abbreviated SUBASSY) a non-empty subset of parts of an assembly. Note that in the limit a subassembly may refer to a whole assembly or a single part.

3 List of Criteria

In this section we list and define the criteria we have identified in our search of the assembly planning literature. We give a short systematic name for each criterion, in many cases using abbreviations from the terms list above. Constraints begin with either "REQ-" or "PRH-" for requiring or prohibiting certain situations, respectively. Quality measures begin with either "MAX-" or "MIN-", with the obvious meanings. The criteria are listed here in alphabetical order of their systematic name.

While we have tried to be exhaustive in the criteria we include, we do not claim to list all papers that have mentioned any given criterion. Instead, we give references to at most one or two representative papers in which each appeared. In addition, some criteria are so general and ubiquitous (e.g. MIN-COST and MIN-TIME) that they appear in almost every assembly planning paper.

In our constraint framework, we have further refined some of the criteria listed here, and created additional ones that were requested by our users. However, our constraint framework imposes an additional source of bias on the survey, which may not be valid for all implementation approaches. Hence we list the criteria here in their original form (i.e. as in [13]). In subsection 6.3 we describe additional constraints that we implemented within our framework.

MAX-LINEAR: Maximize the number of linear insertions in the plan [28].

MAX-PARALLEL: Maximize some measure of the degree of parallelism in the plan [10, 28]. This is closely related to maximizing flexibility at each stage of assembly.

MAX-STAB: Maximize some measure of the stability of states in the plan [19].

MIN-AWKWARD-ACTION: Minimize the awkwardness of an assembly action or sequence of assembly actions [1].

MIN-AWKWARD-GRIP: Minimize the awkwardness of gripping a part or a set of parts [2].

MIN-COST: Minimize the overall cost of the plan. The cost measurement used may vary widely, from human estimates of the cost of each assembly step to algorithmic estimates of the cost of certain factors in each action.

MIN-COST-FIXT: Minimize the overall cost of assembly fixtures for the plan [3, 24, 31].

MIN-DIREC: Minimize the "directionality" of the assembly plan [29]. Directionality might measure the number of insertion directions required by the plan, the range of directions, or the number of direction changes.

MIN-FIXT-COMPLEX: Minimize fixture complexity [3, 24, 31]. This is one way to approximate minimization of fixturing cost.

MIN-REFIXT: Minimize the number of refixturings of the evolving assembly [2, 24].

MIN-REORIENT: Minimize the number of assembly reorientations in the plan [1, 15]. MIN-COST-FIXT, MIN-REFIXT, MIN-DIREC, and MIN-REORIENT are all closely related and often used to approximate each other.

MIN-SIMUL-LIAISON: Minimize the use of simultaneous liaison creation [1]. In some contexts, actions are more difficult or awkward when higher numbers of liaisons are being established by the action.

MIN-TIME: Minimize the time required to execute the plan.

MIN-TOOLCHANGE: Minimize the number of tool changes in the plan [15].

PRH-ACTION: Prohibit a particular action, such as a particular simultaneous liaison creation or state transition [2, 7].

PRH-COLLISION: Do not allow parts to interpenetrate during motions. This a fundamental constraint.

PRH-STATE: Do not allow the assembly to enter a given state [2].

PRH-SUBASSY: Prohibit use of a certain subassembly, or possibly any subassembly containing certain part combinations. For example, one may need to avoid a hard-to-fixture arrangement [7] containing a set of key parts.

PRH-SUBSEQ: Prohibit a particular subsequence of actions.

REQ-ACCESS-TEST: Require that sufficient space be available to perform a test.

REQ-ACCESS-TOOL: Require that sufficient space be available for a tool (manual tool, robotic gripper, welder, laser, etc.) to be applied [21, 26].

REQ-ACTION: Require that a particular assembly action be used in the plan, due to its desirability [1]. This is a minimal case of REQ-SUBSEQ.

REQ-CLUSTER: Require that a set of parts be added to the assembly consecutively, i.e. without interruption by other parts [4].

REQ-CONNECT: Require that every subassembly in the plan be connected. This common constraint is implicit in cut-set methods such as [1, 11].

REQ-FASTENER: Require that certain parts be treated as fasteners for other parts [11, 21]. The fasteners must be placed immediately after the fastened parts are mated.

REQ-LINEAR: Require that parts be inserted one at a time [1, 29]. This is a common constraint as well as a common limitation in other planning systems.

REQ-LINEAR-SUBSET: Same as REQ-LINEAR, but applying only to a subset of operations or parts.

REQ-MONOTONE: Require that the plan be monotone [29]. This is a very common limitation of automated planners; [8] is one system that does not impose this constraint.

REQ-ORDER-FIRST: Require that the assembly plan start with a given part, such as a "chassis" [16], or a set of parts.

REQ-ORDER-LIAISON: Require some ordering between two or more liaison creations; typically stated in a boolean form such as $1 \geq (2\&3)$, or as a set of such boolean statements involving many liaisons [6]. This is a common and powerful type of constraint, analyzed in [30].

REQ-ORDER-PART: Require an ordering between particular part insertions.

REQ-ORDER-SPECIAL: Require a part, liaison, or other ordering not classifiable as one of the above three. For example, [10] mentions the case of certain liaisons that must not be created until some measured result of another subassembly has been obtained.

REQ-PART-SPECIAL: Any special-purpose part constraint, such as those dealing with liquids, springs, snap-fit parts, etc.

REQ-PATHS-AXIAL: Require that each assembly action be along one of the six coordinate directions of a given coordinate system, or a selected subset of these six directions. Common special cases are *uniaxial* constraints (allowing motions in either direction along one axis) [17] and *unidirectional* constraints [16]. This constraint is a limitation of many assembly planning systems.

REQ-PATHS-STR: Require that each assembly action consist of a single translation, or a single combined translation and rotation about a fixed axis [28]. This is a common limitation of assembly planners.

REQ-STAB-ACTION: Require that a part or parts be stably fixtured during some assembly action such as a reorientation [20] or insertion [24].

REQ-STAB-STATE: Require that a subassembly be stably fixtured versus gravity [3, 31].

REQ-STATE: Require that a given state be in the plan [1].

REQ-SUBASSY: Require that a particular subassembly be used in the plan [1, 7].

REQ-SUBSEQ: 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 [2] are relatively complex examples.

REQ-TWO-HANDED: Require that each insertion mate exactly two subassemblies [27, 29]. Except in very theoretical papers (e.g. [22]) this assumption is universal.

4 Categories of Criteria

In this section we categorize the criteria above in ways that may provide insights as to their interrelations and possible approaches to implementation. We categorize criteria according to the following attributes:

- the origin or source of the criterion,
- the degree of obligation of the criterion,
- whether the criterion is of strategic or tactical scope, and
- the information required to assess compliance with a constraint or compute the value of a measure.

We restrict each criterion to being in one category for each attribute, which may in some cases constitute oversimplification. The criteria are listed in Table 1 along with their categories. We define and discuss these categorizations in more detail in the following subsections.

4.1 Origin

First, it is natural to consider the source of the constraint. We use just two categories: *assembly issues per se* and *planning issues.* Assembly issues arise directly from physical constraints and manufacturing concerns, such as fixturing, manipulability, or issues of assembly line layout. Closely related are quality measures arising from a need to optimize one of the three basic aspects of assembly: assembly cost, time, or performance/reliability. In contrast, planning issues are those that arise from the assembly planning process itself, such as simplifying assumptions and limitations of planning systems, or the need to handle large numbers of possible sequences practically.

Most criteria mentioned in the literature are assembly issues. Examples include the need for an assembly to be stable with respect to the forces that arise during an action (REQ-STAB-ACTION), and the need to perform all the assembly actions vertically in robotic assembly using SCARA-type robots (REQ-PATHS-AXIAL). Such criteria may be objectively evaluatable, or may be subjective, such as the judgement that a required manipulation is "awkward" (e.g. MIN-AWKWARD-ACTION). There is a rich set of such "real-world" criteria in the literature.

Criteria that arise from planning issues should be viewed differently from those arising from assembly issues, in that an assembly planning system need not include them in order to be comprehensive, if the methods used in the planning system do not themselves require these criteria in order to be effective.

Some criteria have origins in both assembly and planning issues; this is true for all the strategic constraints in Table 1 except PRH-COLLISION. Many assembly planning systems have technical

CRITERION NAME	OBLIGATION	SCOPE	INFO REQ'D	ORIGIN(SUBTOPIC)
PRH-COLLISION	constraint	strategic	action	assembly(fundamental)
REQ-LINEAR	constraint	strategic	action	assembly(line layout)
REQ-PATHS-AXIAL	constraint	strategic	action	assembly(simplicity)
REQ-TWO-HANDED	constraint	strategic	action	assembly(simplicity)
REQ-MONOTONE	constraint	strategic	action	planning(simplifying)
REQ-PATHS-STR	constraint	strategic	action	planning(simplifying)
REQ-CONNECT	constraint	strategic	state	assembly(fixturing)
PRH-ACTION	constraint	tactical	action	assembly(various)
REQ-ACCESS-TOOL	constraint	tactical	action	assembly(manipulation)
REQ-CLUSTER	constraint	tactical	action	assembly(mfg. efficiency)
REQ-FASTENER	constraint	tactical	action	assembly(various)
REQ-LINEAR-SUBSET	constraint	tactical	action	assembly(line layout)
REQ-ORDER-FIRST	constraint	tactical	action	assembly(mfg. efficiency)
REQ-ORDER-LIAISON	constraint	tactical	action	assembly(various)
REQ-ORDER-PART	constraint	tactical	action	assembly(various)
REQ-PART-SPECIAL	constraint	tactical	action	assembly(various)
REQ-STAB-ACTION	constraint	tactical	action	assembly(fixturing)
REQ-SUBASSY	constraint	tactical	action	assembly(various)
REQ-ACTION	constraint	tactical	plan	assembly(various)
REQ-STATE	constraint	tactical	plan	assembly(various)
PRH-STATE	constraint	tactical	state	assembly(various)
REQ-STAB-STATE	constraint	tactical	state	assembly(fixturing)
REQ-ACCESS-TEST	constraint	tactical	state	assembly(manipulation)
PRH-SUBASSY	constraint	tactical	state	planning(fixturing)
REQ-SUBSEQ	constraint	tactical	action	assembly(various)
PRH-SUBSEQ	constraint	tactical	sub-plan	assembly(various)
REQ-ORDER-SPECIAL	constraint	tactical	various	assembly(various)
MAX-LINEAR	measure	strategic	action	assembly(line layout)
MIN-REFIXT	measure	strategic	action	assembly(cost or time)
MIN-REORIENT	measure	strategic	action	assembly(cost or time)
MIN-SIMUL-LIAISON	measure	strategic	action	assembly(reliability)
MIN-TIME	measure	strategic	action	assembly(time)
MIN-TOOLCHANGE	measure	strategic	action	assembly(time)
MAX-PARALLEL	measure	strategic	plan	assembly(time)
MAX-STAB	measure	strategic	plan	assembly(fixturing)
MIN-COST	measure	strategic	plan	assembly(cost)
MIN-COST-FIXT	measure	strategic	plan	assembly(fixturing)
MIN-DIREC	measure	strategic	plan	assembly(cost)
MIN-FIXT-COMPLEX	measure	strategic	plan	assembly(fixturing)
MIN-AWKWARD-ACTION	measure	tactical	action	assembly(manipulation)
MIN-AWKWARD-GRIP	measure	tactical	action	assembly(manipulation)

Table 1: Assembly Planning Criteria. Criteria are sorted first by their entry in column two, then by their entry in column three, etc., and lastly by the criterion name.

limitations that only allow them to generate plans that are linear or that have axial insertion paths, for instance. However, linear plans also avoid the complexity of merging assembly lines and often minimize the number of assembly fixtures required; axial insertion paths, particularly the sub-case of vertical insertions, are preferred for robotic as well as manual assembly. We have generally chosen the "assembly issue" classification for such criteria. The main exceptions are REQ-MONOTONE and REQ-PATHS-STR, which in our opinion are more often planner limitations than criteria the user wishes to impose, due to the computational complexity required to generate nonmonotone plans or complex insertion paths automatically.

In the last column of Table 1 we have indicated the origin of each constraint, and have added a subcategory comment regarding a more specific topic that criterion might be expected to address. Although not well shown in Table 1, some criteria are sub-cases of others. For example, minimizing fixture complexity (MIN-FIXT-COMPLEX) is an approach to minimizing cost of fixturing (MIN-COST-FIXT), which is itself a subcase of minimizing cost (MIN-COST).

4.2 Obligation

A second, and more useful, categorization is the degree of obligation of the criterion. *Constraints* are absolute, either requiring or prohibiting certain features of assembly plans. *Quality measures* select assembly plans which, at least conceptually, either maximize or minimize a scalar function (abbreviated MAX and MIN).

Note that in many cases constraints can be naturally transformed into quality measures and vice versa. For example, REQ-PATHS-AXIAL could be changed to MAX-PATHS-AXIAL, which would prefer axial insertion paths but still allow non-axial insertions. Similarly, MIN-DIREC could instead be REQ-DIREC, which would only allow a certain number of assembly directions. We list each criterion in the form we found it in the literature. Some issues have appeared as both constraints and measures, such as REQ-LINEAR and MAX-LINEAR, in which case we list both.

In [13] we recognized an additional category of obligation called *suggestions*. The meaning of a suggestion is less clear than a constraint or measure; the planner is free to obey a suggestion or not, depending on other factors. All five suggestions recognized in [13] were versions of other constraints. We now believe that suggestions are more properly understood as versions of constraints that the user believes will make planning easier, but the user does not wish to constrain the ultimate choice of plan. As a result, we have elected to leave the separate category "suggestion" out of this paper.

4.3 Scope

Regardless of the level of obligation a criterion carries, its scope of relevance to the plan or the planning program may be *strategic* or *tactical* (the terminology is taken from [7]). Strategic criteria are applicable during the whole assembly planning process, or to all of the assembly actions. In many cases strategic criteria reduce the size of the search space considerably. On the other hand, tactical criteria address a physically local situation or apply to a relatively small portion of the assembly plan.

Strategic constraints most often arise from limitations of the assembly environment or the planner itself. Tactical constraints more often address stages of the assembly process for a particular product. An example of a tactical constraint is a requirement to have access to a portion of

an assembly at a particular stage in the assembly plan so that a test can be executed (REQ-ACCESS-TEST). Some tactical constraints, such as REQ-ACCESS-TEST, arise due to factors in the assembly process that the assembly planning system does not or cannot consider. Other tactical constraints, such as REQ-SUBASSY, may arise because the user wishes to impose insight on the assembly planner.

Similarly, quality measures are considered strategic when they refer to quantities that are meaningful for an entire plan. For example, the cost of fixturing for an entire plan (MIN-COST-FIXT) is important, but the fixturing cost for any one operation is not even clearly defined. Most quality measures we have identified are strategic; this is natural, because optimizing the overall quality of the plan is most important. The few tactical measures we have found are subjective judgements supplied by the user, and are in fact most often implemented as tactical constraints. For instance, a PRH-ACTION constraint might be applied in order to optimize a MIN-AWKWARD-GRIP measure [2].

4.4 Information Required

The final categorization we use is the information that must be supplied to a criterion for calculation at any given time. For constraints, this is the information that is needed to determine compliance; for quality measures, it is the information needed to calculate the value of the measure for the plan. Most criteria fall into one of two classes: those that require an entire *plan* to calculate, and those that require only local information such as single assembly *states* or *actions*. We use the term *sub-plan* in Table 1 to refer to a sequence of actions smaller than a plan.

In determining the information required to assess a criterion, we prefer in general more limited or local information. For a given constraint, if constraint compliance for a plan is equal to a logical conjunction of constraint applications to all of the individual actions or states in the plan, then we class the information required for that constraint as action or state, respectively. Similarly, if a quality measure for a plan is a sum of the quality measure for each individual action, then we consider the information required to be just an action.

For example, REQ-STAB-STATE could be implemented by evaluating states one at a time (whether algorithmically or interactively by a human) to determine if they pass the chosen stability criterion. Similarly, to implement REQ-ORDER-PART, each proposed assembly action can be compared to a list of part order constraints. In contrast, assembly fixtures and hence their cost (MIN-COST-FIXT) depend on the entire plan or large pieces of it. As a general rule, tactical criteria require local information, but in many cases strategic criteria can also be implemented with only local information.

We should note that the information requirement listed for some of the criteria may seem too localized. We have found that it is possible to implement a number of these criteria using only a description of each action. For example, one might think that REQ-SUBSEQ would require an entire plan before the presence of a given subsequence can be verified. However, each action can be constrained to either (1) keep the relevant parts together if all are present, or (2) remove them in the order required by the REQ-SUBSEQ constraint. In this way, only plans that contain the constraint can be generated. However, we have found no analogous way of implementing PRH-SUBSEQ by only constraining actions. See Section 6 for more examples of constraint implementation.

The information required is an important implementation factor for a criterion. Criteria requir-

ing only local information are much more amenable to standard state-space search and optimization techniques, as we will see in Section 5.

5 Constraint Framework

The wide variety of criteria in our survey present a difficult problem for anyone implementing an assembly planning system: how does one provide a rich enough assortment of criteria in a single maintainable system? In this section we introduce and give an overview of our solution to this problem for constraints only. We describe a framework for interactive user constraints in assembly planning, using a library of constraint types implemented as filters on assembly operations. Section 6 gives a more detailed view of certain aspects of our framework. We have implemented a large subset of the constraints identified in the survey and tested the resulting system on several complex assemblies provided by our customers; these experiments are described in Section 7.

Our constraint framework has the following qualities:

User Friendly Each constraint can be described simply in terms familiar to the user, has straightforward effects, and combines with others in a very predictable way.

Maintainable Each constraint simply provides a filter procedure that disallows some assembly operations. The filter implementations are completely independent, allowing new constraint types to be added easily.

Efficient Because the filters are procedures, they can be implemented in the most efficient way for that constraint. They test individual assembly operations, so they are compatible with standard state-space search and optimization methods. Furthermore, special-purpose methods can be added to improve efficiency.

The result is a comprehensive and extensible library of simple but useful constraints that enable a new, highly interactive mode of assembly planning.

5.1 Previous Work

Our filters are an instance of *generate-and-test*, a standard paradigm in artificial intelligence [23] as well as in assembly planning [7, 11]. Attaching special-purpose methods to constraint tests is also a well known efficiency technique [23], although the methods we use to accomplish this are novel (see Section 6).

Assembly planning systems that allow user-defined constraints have generally been of two types. In the first type (for example [6, 7]), users must specify all constraints before the long process of plan generation begins. In our experience, this is rarely practical: the user finds it very difficult to list *all* constraints on assembly ordering until some possible plans are considered. In the second type of system (e.g. [2]), a large space of plans is first generated, and then undesirable operations and states are pruned interactively by the user. However, the space of plans quickly becomes too large to edit or even generate as assemblies become moderately complex. We approach this problem interactively by generating one or a small number of assembly plans that satisfy the current set of constraints, allowing the user to impose additional constraints, then repeat. Our approach is foreshadowed in a much more limited form by [25].

Previous efforts to incorporate a comprehensive set of user constraints in assembly planners were based on liaison precedence relations. Precedence relations specify logical combinations of part connections that must be established either before or after others. Precedence relations were pioneered by Bourjault [5] and greatly extended by DeFazio and Whitney [6]. Wolter *et al* [30] analyze the expressive power of precedence relations in detail. Precedence relations are quite powerful, but they can be very difficult to write correctly or understand as a user of an assembly planner. We considered translating all of our constraints into precedence relations internally, but chose a procedural approach instead for reasons of efficiency and simplicity of implementation. Our system demonstrates that an assembly planning system can achieve comprehensive constraint coverage while maintaining the advantages of a procedural representation.

5.2 Constraints

Our framework provides a library of constraint types, from which the user can instantiate constraints on the assembly plan. Each constraint is described to the user in straightforward terms based on manufacturing and assembly sequencing concepts and defined using a simple graphic interface. For instance, to instantiate a REQ-SUBASSY constraint, the user simply selects the parts that must belong to the subassembly. Multiple REQ-SUBASSY constraints can be instantiated if desired, each with a different set of parts.

Constraints are implemented as filters. A filter is nothing more than a simple procedure that accepts or rejects assembly operations. During planning, each proposed assembly operation is passed to the constraint's *filter function*, which returns `true` or `false` depending on whether the operation satisfies the constraint or not. Only an operation that satisfies all current constraints is feasible. For instance, consider an operation placing subassembly S_1 into subassembly S_2.[2] The filter function of a REQ-SUBASSY constraint with part set P will return `true` if and only if

$$[P \subseteq S_1] \ \vee \ [P \subseteq S_2] \ \vee \ [(S_1 \cup S_2) \subseteq P] \ \vee \ [(S_1 \cup S_2) \cap P = \emptyset]$$

In other words, the operation satisfies the constraint if it keeps the parts in P together, if only parts in P are involved, or if no parts in P are involved.

As a standard interface to all constraints, the filter function provides a number of benefits. First and foremost, it makes the implementation of each constraint type independent. Interactions between constraints need not be considered, and each constraint can be implemented in its most straightforward and efficient way. This becomes crucial as the number of constraint types grows. In addition, constraints can vary in the data associated with them, their instantiation routines, various debugging outputs, and so on.

For use in a standard state-space search method (such as generating an AND/OR graph for the assembly), it is important that the filter functions take as input single assembly operations, rather than more complex information such as a sequence of operations. In a state-space search, a given operation appears only once in the state graph, and is either present or not. Hence its feasibility cannot depend on operations that come before or after it. In our framework, a number of constraints (e.g. REQ-SUBASSY above) must be implemented in a less natural way to apply to single operations than would otherwise be necessary.

[2] An operation will typically have other specifications, such as a mating trajectory and perhaps an assembly orientation, but these are not relevant to REQ-SUBASSY.

Constraint Name	Scope
REQ-CONNECT*	strategic
REQ-LINEAR*	strategic
REQ-VERTICAL*	strategic
PRH-STATE	tactical
PRH-SUBASSY	tactical
REQ-CLUSTER	tactical
REQ-FASTENER*	tactical
REQ-LINEAR-CLUSTER	tactical
REQ-LINEAR-PARTS	tactical
REQ-ORDER-FIRST	tactical
REQ-ORDER-LAST*	tactical
REQ-ORDER-LIAISON	tactical
REQ-ORDER-PART	tactical
REQ-PART-SPECIAL	tactical
REQ-PATHS-AXIAL	tactical
REQ-STACK	tactical
REQ-STAT	tactical
REQ-SUBASSY*	tactical
REQ-SUBASSY-WHOLE*	tactical
REQ-SUBSEQ	tactical
REQ-TOOL*	tactical

Table 2: Constraints implemented in Archimedes. Those marked by * have special-purpose routines for efficiency.

Filter functions are flexible enough that we have been able to implement a large subset of the constraints identified in our survey, plus some additional ones that our users requested. The flexibility is further demonstrated by the REQ-TOOL constraint, which encodes tool accessibility constraints for various hand and robotic tools [26] within the framework. Table 2 lists the constraint types currently in the Archimedes system. Those that do not appear in the survey are defined in Subsection 6.3.

Although filter functions themselves are usually quite fast, the generate-and-test abstraction can sometimes lead to an inefficient planning process overall. This is particularly true when many dead-ends appear in the search space, or when a large number of assembly operations are generated but few satisfy the constraints. In many cases, special purpose routines can increase efficiency dramatically. The constraint types for which we have implemented such methods are indicated with an asterisk (*) in Table 2. Subsection 6.5 provides more details of these methods.

5.3 Interaction

In experiments with product designers and assembly process engineers, we have found that a high level of interactivity is critical to successful application of an assembly planner. Usually the designer cannot list all the constraints on assembly order at the start of the planning session. However, many of these constraints become "obvious" when the system graphically illustrates a plan that violates them. Seeing a violation, adding a constraint to remove it, and then replanning becomes the main cycle of interaction in the system. In this way, the assembly planner aids constraint discovery and management as well as plan generation and optimization.

Note, however, that placing a new constraint is very different from ruling out a certain operation, as performed in some previous systems such as [2]. Although a single operation demonstrates the need for a constraint, placing the constraint prohibits similar actions from occurring in many different operations, and hence limits the allowable plans far more than prohibiting a single operation. In the best case (and in many practical cases), the constraint encodes the manufacturing constraint exactly.

This plan–view–constrain–replan cycle requires that replanning be performed at interactive speeds. In the Archimedes system, a first assembly plan for a product can usually be found in a few minutes [15]. However, the most expensive part of planning is ensuring that part insertions are collision-free. By saving collision-detection information, replanning usually requires 10 or 20 seconds for assemblies of up to 100 parts.

There is of course no guarantee that all of the constraints the user has instantiated can be satisfied by a single plan. In this case, the planner fails and enters a "debug" mode that helps the user to determine the cause of the failure. If the constraints are all real, then a problem with the product design may be indicated. In most cases, some constraints can be adjusted to allow planning to succeed. When there are inaccuracies or inconsistencies in the product CAD data, planning can fail before the user has entered any constraints. The debug mode also supports finding such problems, and certain problems can be fixed within Archimedes. Subsection 6.4 provides more details.

Note that if constraints are entered inaccurately, they may over-constrain the choice of plan and rule out some plans incorrectly. However, if some plans still satisfy the entered constraints, then the error may go unnoticed. This is a difficult problem to solve in a system allowing user specification of constraints.

After all known manufacturing constraints have been entered, the user can then ask for an optimal plan, according to user-specified costs of certain standard operations. In some cases additional unstated constraints will be violated and discovered as the planner looks through a large space of plans to find the best. In this case the new constraints must be added and the cycle repeats.

6 A Detailed View

6.1 Assembly Planning Approach

The approach taken to assembly planning is obviously critical to the design, implementation, and performance of a user constraint system. It especially affects special-purpose routines for efficiency. Our constraint system was added to the Archimedes mechanical assembly planner [15]. The system

consists of four main elements: a user interface, a constraint system, a search engine, and an animation module.

Archimedes generates two-handed monotone assembly sequences in reverse, starting from the more highly constrained, fully assembled state. This a standard technique in assembly planning. The search space is an AND/OR graph of subassembly states and operations to construct them from smaller subassemblies. The planner uses an NDBG of each subassembly [28] to efficiently determine assembly operations that might be performed to construct that subassembly, then checks these operations for geometric collisions, which is essentially a built-in filter. Operations are then checked against the list of user constraints.

The search strategy is carefully tuned to generate a first plan as quickly as possible in the domain of mechanical assembly. This is critical to achieve the desired view–constrain–replan cycle of interaction. The search algorithm is not the focus of this paper, and space allows only a cursory description. An AND/OR version of iterative sampling [18] is performed: during each pass of the algorithm, a single assembly sequence is generated, making random choices of operations to construct each subassembly. The first time any subassembly is visited, only a single operation is generated to construct it, and the known subassemblies of that operation are then visited. Bounds on quality measures for each subassembly and operation are stored and propagated in the AND/OR graph as they are generated. This allows useless search paths to be identified and pruned and an optimal plan to be identified when it is known. The strategy is designed to quickly reach a first solution, like a depth-first search, but to avoid getting caught by bad early decisions as a depth-first search would. The same algorithm functions as an any-time algorithm to optimize the assembly sequence when the user requests.

6.2 User Interface

The user interface is critical to effectiveness and user acceptance of an interactive planning system. The constraints must be easy to understand, define, and manage. In this subsection we describe features of the Archimedes user interface that are important to the success of the constraint system. See [14] for a better understanding of the Archimedes constraint interface.

Figure 1 shows the main Archimedes user interface. The upper left window shows the program's current status, displays any diagnostic output, and allows pausing or aborting any computation. The upper right window provides graphic output and part/subassembly selection. The bottom window is the main control window.

After loading the CAD data for an assembly and perhaps making some initial adjustments to it (see Subsection 6.4), the user selects "Plan", which brings up the planning dialog shown in Figure 2. The top half of this dialog concerns global choices for the planner, and the bottom half provides management of the current set of constraints.

Constraints are added by clicking on the "Add" button at the bottom, which brings up a sequence of menus and questions that let the user pick a constraint type and specify the particulars of the desired constraint. Each constraint requires the user to select one or more parts of a "controlled" set in the graphic window, such as the parts making up a subassembly. For some constraints additional inputs must be provided, such as a second set of parts required by the constraint or the choice and placement of a tool to be used in assembly (see Figure 3). An auxiliary window provides a list of named subassemblies to facilitate selection of larger sets of parts. Each constraint can be

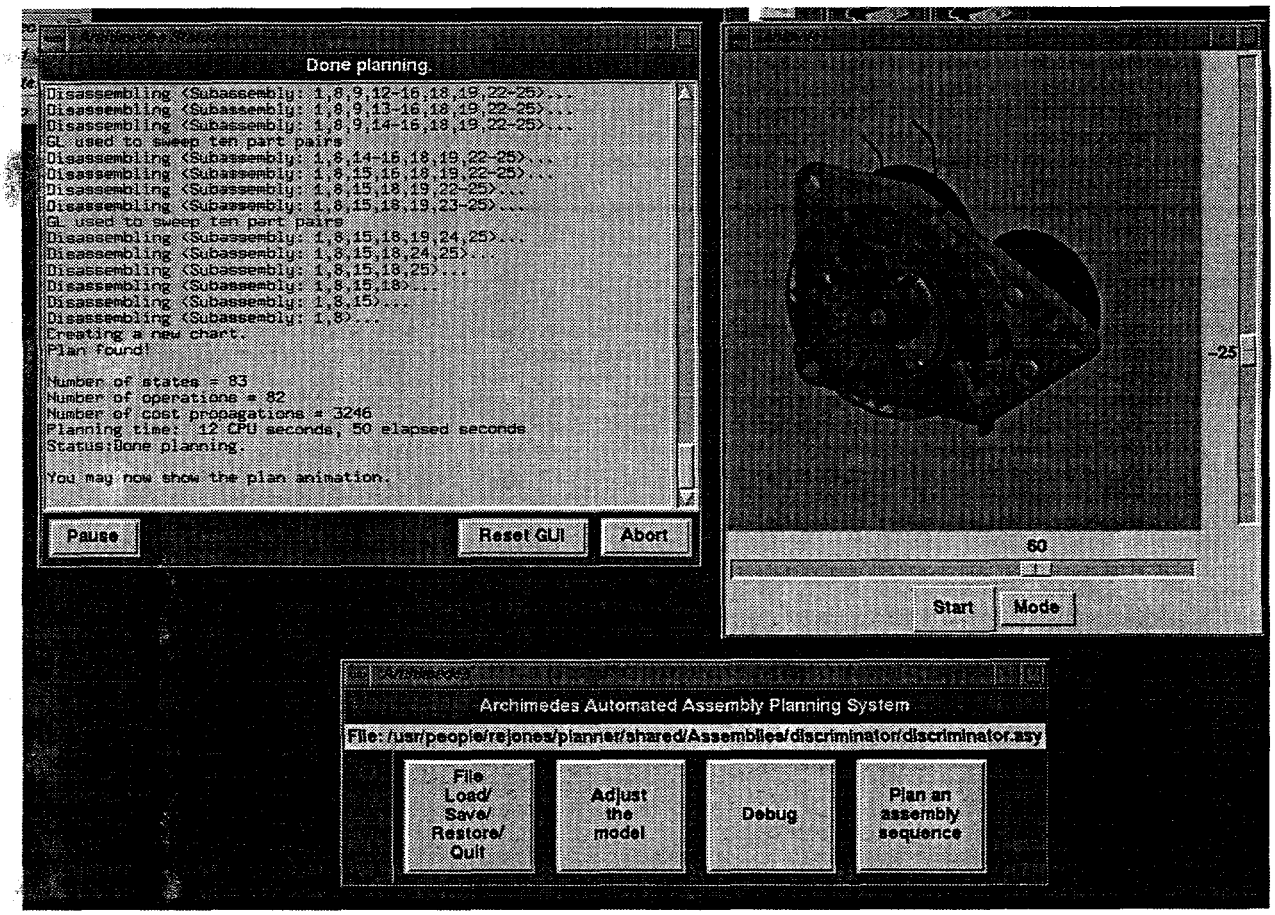

Figure 1: The Archimedes assembly planning system

given a name and descriptive comment by the user. Some simple checks are performed to catch certain obvious inconsistencies within and between constraints.

Once defined, constraints are listed in the planning dialog. They can be edited using a process very similar to initial definition. They can also be deleted, temporarily suspended, and re-activated. Constraint suspension is a very useful feature that allows the user to consider various scenarios for assembly. Constraints often embody assumptions about the product assembly scheme; by suspending some and replanning, the user can compare the cost of removing the assumption to the possible gains in assembly sequence efficiency that result.

6.3 Additional Constraints

In experience with our constraint framework, we implemented several additional constraint types that we did not find in our survey. Some constraints were requested by users explicitly, while others were identified by members of our team. We describe them below. These constraint types do not

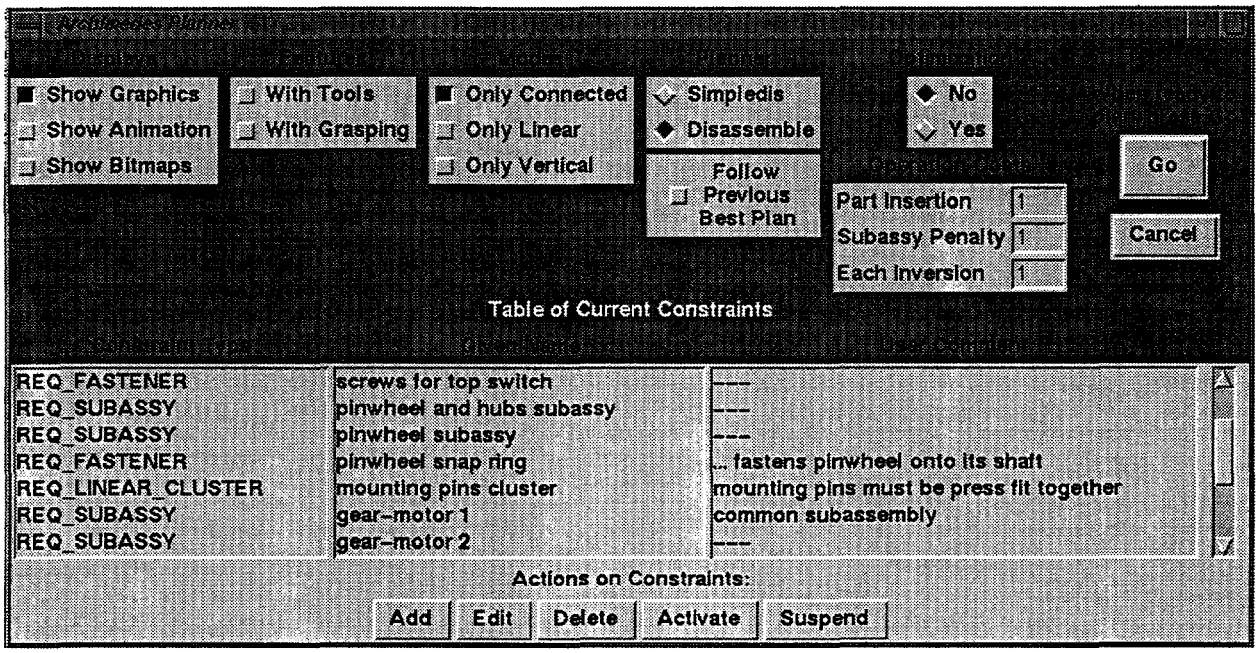

Figure 2: Planning dialog, showing the list of current constraints

appear (in the same form) in Section 3 or Table 1, but do appear in Table 2.

REQ-VERTICAL: Requires that all parts be placed directly downward, but unlike REQ-PATHS-AXIAL, allows the assembly to be reoriented so that any given action is considered "downward". This constraint is very useful when planning assembly with standard SCARA-type robots, where insertions must be vertical but reorientations should be minimized.

REQ-LINEAR-CLUSTER: A combination of REQ-CLUSTER and REQ-LINEAR-SUBSET.

REQ-LINEAR-PARTS: Requires that a given set of parts be assembled linearly when they are being mated with any of another set of parts.

REQ-ORDER-LAST: Requires that a certain part of set of parts be placed last in the assembly plan.

REQ-PART-SPECIAL: Any action involving the part is shown to the user for manual confirmation of its feasibility. This is very time-consuming, but in some cases it is required.

REQ-STACK: Specifies a set of parts to be assembled one at a time in a given direction.

REQ-STAT: Requires a set of parts to be in the stationary subassembly when mated with any of another set.

REQ-SUBASSY-WHOLE: The same as REQ-SUBASSY but tells the planner in addition not to generate a plan to construct the subassembly.

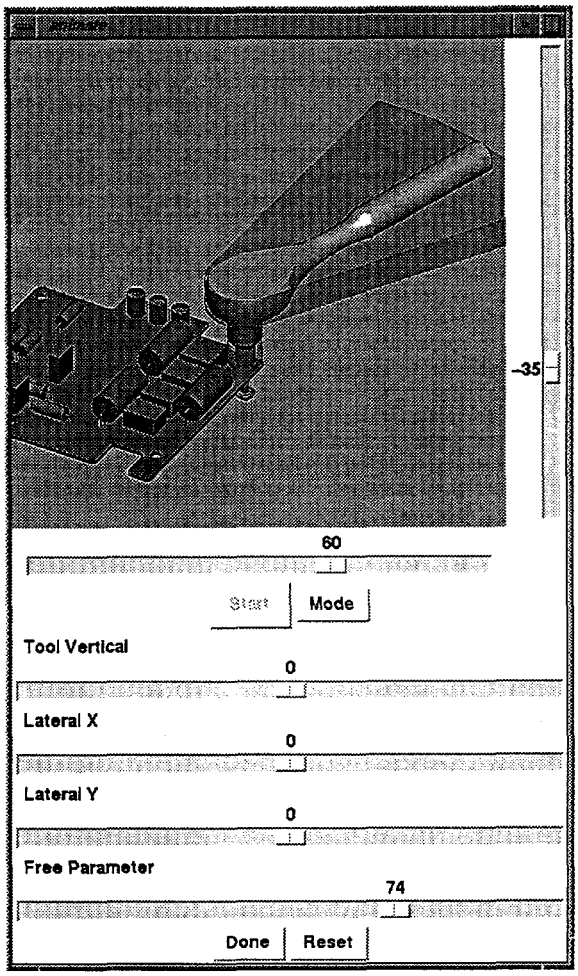

Figure 3: Defining the placement of a socket wrench for a REQ-TOOL constraint. The constraint requires the translucent volume to be free when the tool is applied [26].

REQ-TOOL: Requires that a collision-free placement of a given tool use volume must exist in the assembly during a certain operation. See [26] for more details.

6.4 When Planning Fails

When the product cannot be assembled according to the current set of constraints, the planner fails and enters a "debug" mode that helps the user to determine the cause of the failure. For example, one can request that the planner try to remove a particular part or subassembly (from the subset of parts remaining when the planner failed) in a direction that appears possible. Collisions or constraints that disallow the operation are then posted in the status window. Other options include trying to disassemble every pair of parts in the offending subassembly, or trying to remove any parts along a given trajectory. This constraint debugging capability is very important and useful, since

in some cases it may not be apparent to the user why the planner cannot find a feasible plan.

Often, the planner can fail without any user-defined constraints. This is sometimes due to limitations in the planner's algorithms, such as an inability to reason about flexible parts such as snapfits and springs. Other times, inaccuracies or inconsistencies in the product CAD data cause the planner to fail. Examples include pressfit parts and threaded parts that are modeled as cylinders too large for their holes. Archimedes includes a set of model adjustment features, or *overrides*, which can be used to correct such problems. These include a function to effectively add threads to cylindrical contacts between parts; to specify that certain part insertions are in fact possible, even though collisions occur between the parts; and to delete a part temporarily.

6.5 Efficiency

As mentioned above, the generate-and-test abstraction can sometimes lead to an inefficient planning process where a large number of operations are generated, very few of which pass through the filters. In such cases supplemental routines can improve planning efficiency greatly. These routines are very dependent on the internal algorithms of the planner. Because Archimedes plans backward from the assembled state to individual parts, the supplemental routines must be implemented in reverse.

For instance, if the user has created a REQ-SUBASSY constraint with part set P, and parts not in P are present, then P cannot be "split" at that point in the plan. To implement this constraint efficiently, a supplemental routine binds the parts of P together for that stage, not considering any operations that split them. This is accomplished by placing bidirectional arcs between every pair of parts in P in every blocking graph of the subassembly [28].

Supplemental routines must be considered carefully, trading off the added speed against the increase in planner complexity. Three characteristics identify candidates for supplemental routines:

1. the constraint either leads to many dead-ends in the search space or rules out a very large proportion of generated operations,

2. an efficient method exists to implement the preprocessing, and

3. the constraint is used often.

The REQ-FASTENER constraint type is another instructive example. Fasteners are very common in mechanical assemblies. The REQ-FASTENER constraint requires that as soon as one set of parts is joined (the *fastened parts*), then a set of *fastener parts* must immediately be placed. In reverse, this constraint means that as soon as a single fastener part is removed, then all other fasteners must be immediately removed, followed by at least one of the fastened parts. If any of the fasteners cannot be removed, a dead end appears in the search space (in fact many can appear).

The filter function for REQ-FASTENER is straightforward, but its supplemental routines are the most complex we have implemented. The fastener parts are removed from the assembly representation and considered *secondary parts*, that implicitly must be added when the fastened parts are mated. Before generating operations to construct a certain subassembly, the planner determines which fasteners could be placed into the subassembly. For each fastener that cannot be placed, the corresponding fastened parts are bound together as for REQ-SUBASSY. Fasteners are placed back in subassemblies for collision checking and other calculations.

Assembly	Parts	Overrides	Constraints
Sprytron	18	0	5
Door latch	23	28	8
Discriminator	42	18	15
Hughes	472	26	144

Table 3: Example assemblies planned with Archimedes

Note that when a constraint has supplemental routines, the planner still calls the constraint's filter function, which should never return `false`. This double check is very useful to ensure correctness, because the supplemental routines are complex and interact with each other. It is conceivable that a supplemental routine would only *reduce* the number of operations rejected by the filter function, but we have not found such a case.

In almost all cases, adding constraints reduces planning time. The reduced size of the search space usually outweighs the extra time required to compute the filter functions. In fact, constraints can be used to guide the planner to a correct plan for assemblies that would otherwise have intractably large search spaces.

6.6 Implementation

Archimedes is implemented in C++, with Tcl/Tk used for the graphic interface and OpenGL™ for 3D graphics and animation. The constraints are implemented as a hierarchy of C++ derived classes. Each type of constraint simply overrides the filter function from a base class, along with methods to define the type, name, etc. of the constraint. Each constraint type also has its own data members, such as part sets, tool choices and points of application, and so on. Some of the supplemental routines are implemented as constraint class methods; however, most cannot be separated from the planner's algorithms, and are woven directly into the planner implementation.

In our survey above, we differentiated between strategic and tactical constraints. Archimedes currently implements strategic constraints as flags for the planner. However, in implementing our framework it has become clear that in theory there is no real difference between strategic and tactical constraints. Tactical constraints can usually be applied to the entire assembly, whereas strategic constraints can always be limited to a subset of the parts. For instance, the REQ-PATHS-AXIAL constraint, identified in Table 1 as strategic, is also very useful applied to a subset of the assembly. Implementing it as a tactical constraint allowed both uses, was simpler to implement, and caused no loss in efficiency. We plan to replace all strategic constraints in Archimedes with tactical constraints.

7 Examples

We have applied the Archimedes planner, extended with the constraint system, to a number of actual assemblies from sources in government and industry. Example results are summarized in Table 3.

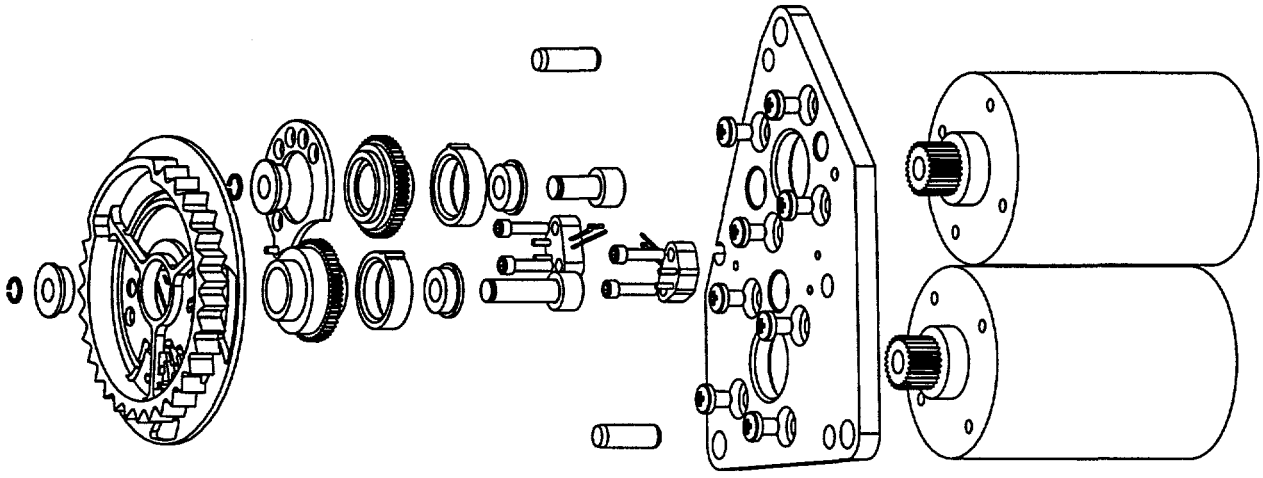

Figure 4: An exploded view of the discriminator

The **sprytron** is an eighteen part diode-like device, including a surrounding assembly fixture. Most of the parts are symmetric or nearly symmetric about a central axis. The CAD data required no overrides to produce a first plan. However, in the resulting plan some parts that would more naturally be inserted along the axis were inserted from other directions. Adding a REQ-PATHS-AXIAL constraint removed all these unwanted directions, and adding a REQ-ORDER-FIRST placing the fixture first made the plan even better. Three REQ-ORDER-LIAISON constraints were then added to fine tune the assembly plan to specific manufacturing constraints.

The **door latch** mechanism involves 23 parts, some of which are very complex. The presence of snapfit and riveted fasteners, plus many inaccuracies in the CAD data, required 28 overrides. A good plan was obtained after seven REQ-SUBASSY constraints were added, and one REQ-ORDER-LAST was used.

The **discriminator** is a 42-part clockwork-like mechanism used as a safety device. It is the object partially shown in the animation window in Figure 1, and Figure 4 shows an exploded view. Several parts overlapped in the CAD data, including 12 screws which were modeled larger than their corresponding holes, resulting in 18 model overrides. A plan for the resulting adjusted model was then found with no need for constraints. To make Archimedes place all fasteners in appropriate groups seven REQ-FASTENER constraints were needed, and to match the sub-assemblies intended by the designer six REQ-SUBASSY constraints were used. In addition, one REQ-LINEAR-CLUSTER was used, and the chassis was requested to be placed first with a REQ-ORDER-FIRST.

The **Hughes assembly** in Figure 5 is an initial design of the guidance section of a Hughes Aircraft AIM-9X air-to-air missile. With 472 parts described by 55Mb of ACIS™ data (translated from Pro/ENGINEER™) and over 800,000 facets, the Hughes assembly is to our knowledge the most complex assembly that has been processed by any automated assembly planning system. Since Archimedes plans only for straight line assembly motions, and this assembly contained a number of flexible parts (such as cables) that could not use straight line insertions, we overrode

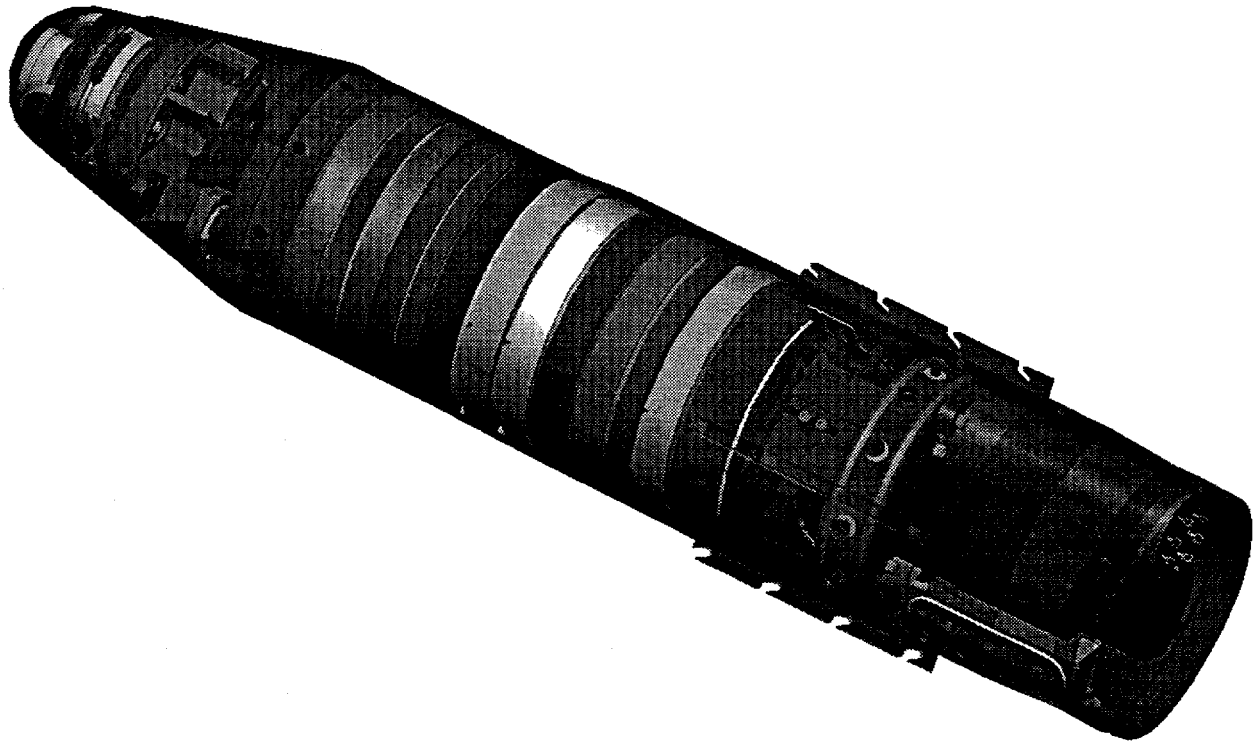

Figure 5: The guidance section of a Hughes Aircraft AIM-9X air-to-air missile. Figure used by permission of Hughes Aircraft and the U.S. Naval Air Command.

22 part mating situations with specific matings. Four other overrides clarified contacts between parts. A large number of REQ-SUBASSY-WHOLE constraints were used for subassemblies that our Hughes customer was not interested in sequencing. A breakdown of all the constraints used to produce a plan useful to the customer is in Table 4. After loading and facetting all the parts in the Hughes assembly, Archimedes requires approximately 22 minutes to find all contacts and produce an assembly plan. After modifying constraints, replanning is usually performed in a few minutes.

The reader may note an approximate 3-to-1 ratio of parts to constraints in Table 3. Although our data is clearly insufficient to draw any conclusions, this may be indicative of what we should expect for typical applications.

8 Conclusion

We have surveyed, formalized, and categorized a rich set of assembly criteria appearing in the assembly planning literature or of common knowledge. We hope this collection of criterion definitions and categorizations will be useful to other assembly planning software and research efforts. The perspective provided by the survey influenced to a great extent our development of an interactive user constraint system for the Archimedes computer-aided assembly planning system.

Constraint	Count
REQ-SUBASSY-WHOLE	70
REQ-ORDER-LIAISON	48
REQ-CLUSTER	11
REQ-SUBASSY	7
REQ-FASTENER	3
REQ-ORDER-FIRST	2
REQ-ORDER-LAST	2
REQ-PATHS-AXIAL	1

Table 4: Constraints for the Hughes Assembly

In our experience, constraint-based interaction has proven to be a powerful and intuitive paradigm for interactive assembly planning. In addition, we have been very pleased with the concept and implementation of constraints as filters. It keeps the code simple, maintainable, and efficient, especially when supplemented with special-purpose routines for certain constraints. A rich variety of useful constraints can be so represented, and we have easily added constraint types when required. The interactive mode of user-guided planning that results from the system has been very effective in our experience.

At present, our framework only addresses the constraints from our survey. We are working to expand the scope of our algorithms to also include quality measures and suggestions. A relatively simple approach to user-specified quality measures would have a library of measure types available, parallel to our current library of constraints. Each measure would express a certain condition; but rather than requiring or prohibiting conditions like constraints, the measure would simply express a positive or negative value that is incurred when the condition happens in a plan. In this view, constraints could be seen as measures with infinite values, although planners can be more efficient with the direct constraint representation we currently have. However, it remains to be seen whether such a straightforward framework can be as successful in representing user quality measures as our constraint framework has been in representing user process constraints.

Acknowledgments

We would like to thank Glenn Laguna and Arlo Ames for useful discussions and their work on the Archimedes system. We are also grateful to several anonymous reviewers for their comments.

This work was supported by Sandia National Laboratories under DOE contract DE-AC04-94AL85000.

ACIS™ and Pro/ENGINEER™ are registered trademarks of Spatial Technology Inc. and Parametric Technology Corp., respectively. OpenGL™ is a trademark of Silicon Graphics, Inc.

References

[1] T. E. Abell, G. P. Amblard, D. F. Baldwin, T. L. De Fazio, M.-C. M. Lui, and D. E. Whit-

ney. Computer aids for finding, representing, choosing amongst, and evaluating the assembly sequences of mechanical products. In [9], pages 383–435.

[2] 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.

[3] D. Baraff, R. Mattikalli, and P. Khosla. Minimal fixturing of frictionless assemblies: Complexity and algorithms. *Algorithmica*, 19(1/2):4–39, 1997.

[4] N. Boneschanscher and C. J. M. Heemskerk. Grouping parts to reduce the complexity of assembly sequence planning. In E. A. Puente and L. Nemes, editors, *Information Control Problems in Manufacturing Technology 1989: Selected Papers from the 6th IFAC/IFIP/IFORS/IMACS Symposium*, pages 233–238. Pergamon Press, 1989.

[5] A. Bourjault. *Contribution à une approche méthodologique de l'assemblage automatisé: élaboration automatique des séquences opératoires.* PhD thesis, Faculté des Sciences et des Techniques de l'Université de Franche-Comté, 1984.

[6] T. L. De Fazio and D. E. Whitney. Simplified generation of all mechanical assembly sequences. *IEEE Journal of Robotics and Automation*, RA-3(6):640–658, 1987. Errata in RA-4(6):705-708.

[7] J. M. Henrioud and A. Bourjault. LEGA: a computer-aided generator of assembly plans. In [9], pages 191–215.

[8] R. L. Hoffman. Automated assembly in a CSG domain. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 210–215, 1989.

[9] L. S. Homem de Mello and S. Lee, editors. *Computer-Aided Mechanical Assembly Planning.* Kluwer, 1991.

[10] L. S. Homem de Mello and A. C. Sanderson. Evaluation and selection of assembly plans. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 1588–1593, 1990.

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

[12] L. S. Homem de Mello and A. C. Sanderson. Representations of mechanical assembly sequences. *IEEE Trans. on Robotics and Automation*, 7(2):211–227, 1991.

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

[14] R. E. Jones and R. H. Wilson. An interactive assembly planning system. In *Video Proc. IEEE Intl. Conf. on Robotics and Automation*, 1997.

[15] S. G. Kaufman, R. H. Wilson, R. E. Jones, T. L. Calton, and A. L. Ames. The Archimedes 2 mechanical assembly planning system. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 3361–8, 1996.

[16] H. Ko and K. Lee. Automatic assembling procedure generation from mating conditions. *Computer Aided Design*, 19(1):3–10, 1987.

[17] E. Kroll, E. Lenz, and J. R. Wolberg. Rule-based generation of exploded-views and assembly sequences. *Artificial Intelligence in Engineering, Design, and Manufacturing*, 3(3):143–155, 1989.

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

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

[20] R. Mattikalli, D. Baraff, and P. Khosla. Finding all stable orientations of assemblies with friction. *IEEE Trans. on Robotics and Automation*, 12(2):290–301, 1996.

[21] J. M. Miller and R. L. Hoffman. Automatic assembly planning with fasteners. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 69–74, 1989.

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

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

[24] B. Romney. Atlas: An automatic assembly sequencing and fixturing system. In W. Strasser, R. Klein, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, pages 397–415. Springer-Verlag, 1997.

[25] R. H. Wilson. Minimizing user queries in interactive assembly planning. *IEEE Trans. on Robotics and Automation*, 11(2):308–312, 1995.

[26] R. H. Wilson. Geometric reasoning about assembly tools. Technical Report SAND95–2423, Sandia National Labs, 1996.

[27] R. H. Wilson, L. Kavraki, T. Lozano-Perez, and J.-C. Latombe. Two-handed assembly sequencing. *Intl. J. of Robotics Research*, 14(4):335–350, 1995.

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

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

[30] J. D. Wolter, S. Chakrabarty, and J. Tsao. Mating constraint languages for assembly sequence planning. *IEEE Trans. on Robotics and Automation.* To appear.

[31] J. D. Wolter and J. C. Trinkle. Automatic selection of fixture points for frictionless assemblies. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 528–534, 1994.