Proc. 1996 IEEE Intl. Conf. on Robotics and Automation, pp. 1525-1532

A Survey of Constraints in Automated Assembly Planning*

Rondall E. Jones Randall H. Wilson

Intelligent Systems and Robotics Center
Sandia National Laboratories
Albuquerque, NM 87185-0951

Abstract

This paper is a survey of constraints used in automated assembly planning, where "constraints" are interpreted broadly to include requirements, preferences, and suggestions to the planner. We list a large number of constraints that have appeared in the assembly planning literature and categorize them along several dimensions. These dimensions include strategic *vs.* tactical constraints; requirements *vs.* criteria to optimize *vs.* suggestions; and constraints encoding manufacturing requirements *vs.* those imposed by the automated planning process itself. We also consider the information needed to assess each constraint, particularly as it affects the efficiency of planning with the constraint. Finally, we present our planned framework for incorporating many of the surveyed constraints into an interactive version of the Archimedes assembly planner.

1 Introduction

Useful assembly plans must satisfy a large number and variety of constraints. Representing these constraints and finding assembly plans that satisfy them are the primary capabilities required to create automated assembly planners. Many different constraints have appeared in the literature on automated assembly planning or been implemented in experimental systems. However, the constraints often occur stated in different ways and in a variety of forms and contexts.

This paper surveys the assembly planning literature to identify, synthesize, and organize common constraints on assembly plans. We focus on *what* constraints appear in the literature, rather than *how* they have been implemented. Implicit in our survey is the assumption that if a constraint 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 constraints and categorize them along several dimensions, such as the origin of the constraint and whether it is absolute or a criterion to optimize. We also consider the information needed to assess each constraint, as it affects the efficiency of planning with the constraint. Finally, we describe our planned framework for incorporating many of the surveyed constraints into an interactive assembly planning system.

We should comment on our use of the term "constraints". While there will be no confusion with calling requirements and prohibitions "constraints", some may think the term is inappropriate when applied to optimizations or suggestions (see Section 6.2). But from the viewpoint of an automated planning tool all of these are user-given directives which *constrain* the final choice of plan. Thus, for the lack of a better term we will refer to all the requirements, prohibitions, requested optimizations, and user suggestions as "constraints".

Our approach has the following additional motivations and assumptions:

Inclusion: We include all constraints mentioned in the assembly planning literature with which we are familiar. In addition, we have included constraints 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 constraints is new or novel.

Exclusion: On the other hand, this is not an attempt to catalog all *possible* constraints. 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* use of a cluster makes sense. Thus, the logically possible constraint of *prohibiting* use of a *cluster* does not seem useful, and is not included.

In the literature, constraints are often discussed to-

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

gether with features of the computer programs which implement them. Such software features may superficially seem like constraints, when they are actually just "housekeeping" functions. For example, a feature to DELETE A STATE represents a constraint [1]. The related software feature of UNDELETE A STATE just clears a previously selected constraint, and thus would not be included here. This stands in contrast to a software feature to RECOMMEND A STATE, whose intent is to guide the planning algorithm to a solution, and would be included here.

Grouping: Often very similar ideas can be expressed quite differently. For example, prohibiting multiple simultaneous subassemblies (as in [1]) is the same as requiring linearity [14] (see definitions below), so we arbitrarily include only the latter. Similarly, a constraint to minimize parallelism in a plan is very close to the concept of maximizing linearity, so we include only the latter. No doubt we have oversimplified in some cases, and each decision we have made to combine constraints 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 constraints: Requirement (and prohibition) constraints can be combined using logical conjunctions, disjunctions, and negations. Optimization criteria may be combined using a scalar function. However, we do not delineate which constraints may be combined with which others. Instead, we have attempted to identify the basic constraints that might be the terms in such composite constraints. Of course, combinations of constraints must be considered carefully when implementing an assembly planning system.

2 Overview

The remainder of this paper is organized as follows: In Section 3 we introduce categorizations which attempt to capture important qualities of assembly constraints. Some categorizations (namely *obligation* and *information required*) turn out to be quite useful for planning implementation of constraints, while others are not directly useful, though they may be interesting.

In Section 4 we define terms and abbreviations we use in the later sections.

Section 5 is the core of the paper, in which we give a short systematic name to each constraint and briefly define it, with references to previous research that used the constraint.

In Section 6 we organize the constraints using various categorizations. We discuss examples and subcategorizations for the benefit of the reader's understanding of the various ways of viewing the constraints.

Finally, in Section 7 we present an approach to implementing constraints that we plan to use in an interactive extension to the Archimedes 2 assembly planner [9]. This approach crystallized during the process of organizing these constraints.

3 Categorizations

In surveying a large number of constraints and attempting to organize them, several categorizations of assembly planning constraints become apparent. We present them here briefly; they will be used and discussed further in Section 6.

3.1 Origin

First, it is natural to consider the source of the constraint. We use just two categories:

Assembly issues per se: The largest category of constraints is those arising directly from a physical constraint on assembly, such as fixturing constraints, manipulability constraints, or issues of assembly line layout. Closely related are constraints arising from a need to optimize one of the three basic aspects of an assembly project: assembly cost, assembly time, or assembly performance/reliability.

Planning process issues: A number of constraints arise from the assembly planning process itself, such as simplifying assumptions or human guidance of a search.

3.2 Obligation

A second, and more useful, categorization is the degree of obligation of the constraint. Some constraints are absolute, either *requiring* or *prohibiting* certain features of assembly plans. (These are abbreviated as REQ and PRH, respectively, in the constraint names below, and are referred to as a group as requirements.) Others are preferences, which we further subdivide into optimizations and suggestions. *Optimization* constraints select assembly plans which, at least conceptually, either maximize or minimize a scalar function (abbreviated MAX and MIN). *Suggestions* (abbreviated SUG below) appear often in the literature, but their meaning is a little less clear. The planner is free to obey a suggestion or not, depending on other constraints. In some systems suggestions are used to help guide the planner to a better plan or a quicker result. In others they seem to be an informal type of optimization criterion, so that plans that obey suggestions tend to be selected. Finally, suggestions are sometimes used to override an algorithm's normal treatment of a particular situation, as with a flexible part that appears unassemblable due to rigidity assumptions [9].

3.3 Scope

Regardless of the level of obligation a constraint carries, its scope of relevance to the plan or the planning program may be *strategic* or *tactical*. Strategic constraints are applicable during the whole assembly planning process, or to all or most of the assembly actions. Tactical constraints address a physically local situation or apply to a relatively small portion of the assembly planning process.

3.4 Information Required

An important factor when implementing constraints in a program is what information is needed to evaluate whether a given constraint is met. As we will discuss, almost all constraints fall into one of two classes: those that require an entire plan or set of plans to assess compliance, and those that require only very local information such as single assembly states or actions.

4 Terms and Abbreviations

The following terms and their abbreviations are used in the later sections of this paper. Most of these are commonly used in the literature. For references, see the next section.

- *AND/OR graph*: a commonly used representation of a set of assembly sequences, listing subassemblies and actions that create larger subassemblies from smaller ones.
- *assembly*: a set of parts, in given geometric relation to one another
- *assembly action*: (abbreviated as ACTION) any single action of bringing together parts or subassemblies, or of moving parts or subassemblies. It is a more general term than "insertion".
- *awkwardness*: a usually subjective judgement of the relative difficulty of a task.
- *cluster*: a group of parts to be assembled in uninterrupted sequence, but for which the sequence is not specified. 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.
- *fixture*: (abbreviated as FIXT) a device to hold a part, assembly, or subassembly.
- *insertion*: the adding of a single part or subassembly to an evolving assembly.
- *liaison*: the geometric relationship between two parts which are touching or effectively touching, whether physically attached or not.
- *linear*: a property of an assembly plan in which all parts are added to the assembly one at a time.
- *monotone*: a property of an assembly plan in which each part is inserted immediately into its final position relative to the remainder of the assembly. Thus, an n-part assembly is executed in 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 the assembly plan.
- *state transition diagram*: (abbreviated STD) a commonly used representation of a set of assembly sequences, listing states and the feasible transitions between them.
- *subassembly*: (abbreviated SUBASSY) a subset of parts of an assembly; this over-used term may mean either (1) a connected set which is treated much as if it were a part, or (2) a stage in an evolving assembly.

5 Constraint List

The constraints we have identified are named and briefly defined in this section, in alphabetical order of the short name. Due to limited space, we reference at most one or two representative papers in which constraints appear.

MAX-LINEAR: Maximize some measure of the degree of linearity in the plan [13].

MAX-PARALLEL: Maximize some measure of the degree of parallelism in the plan [7, 13]. This is closely related to maximizing flexibility at each stage of assembly.

MAX-STAB: Maximize some measure of the stability of states in a plan [11].

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.

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 fixturing parts and subassemblies.

MIN-DIREC: Minimize the "directionality" of the assembly plan [14]. 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 [2, 14, 16]: this is one of several possible ways to approximate minimization of fixturing cost.

MIN-REFIXT: Minimize the number of refixturings of the evolving assembly [1].

MIN-REORIENT: Minimize the number of assembly reorientations in the plan [1, 9]. MIN-COST-FIXT, MIN-REFIXT, MIN-DIREC, and MIN-REORIENT are closely related.

MIN-SIMUL-LIAISON: Minimize the use of simultaneous liaison creation [1]. This is related to awkwardness constraints, but is more specific. This constraint would be used in contexts in which more liaisons being established in a single assembly action make the action more difficult. (Note that in some contexts multiple liaison creation helps rather than hinders.)

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

MIN-TOOLCHANGE: Minimize the number of tool changes in an assembly plan [9].

PRH-ACTION: Prohibit a particular action, such as a particular simultaneous liaison creation or state transition [4, 1].

PRH-COLLISION: Require that each part not be penetrated by any other part during motions. This a fundamental constraint.

PRH-STATE: Remove a state from consideration. This is used frequently in STD methods [1].

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 [4] 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 [12].

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

REQ-CLUSTER: Require that a set of parts be added to the assembly without interruption by other parts [3].

REQ-CONNECT: Require that every subassembly in the plan be connected. This common constraint is implicit in cut-set methods such as [1, 8].

REQ-FASTENER: Require that certain parts be treated as fasteners for other parts [12].

REQ-LINEAR: Require that parts be inserted one at a time [1, 14]. This is a common constraint.

REQ-LINEAR-SUBSET: Same as REQ-LINEAR, but applying only to a subset of operations or a cluster of parts. This constraint would also cover the minimal case of requiring that a part be inserted into final position individually.

REQ-MONOTONE: Require that the plan be monotone [14]. Very common; [5] 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" [10], 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. This is a very common type of constraint, analyzed in [15].

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, [7] 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. For example, insertion might be allowed from any axial direction except vertically from below. Common special cases are *uniaxial* constraints (allowing motions in either direction along one axis) and *unidirectional* constraints [10].

REQ-PATHS-STR: Require that each assembly action be in a straight line, or a straight line with a screwing motion [13].

REQ-STAB-ACTION: Require that a part or parts be stably fixtured during some assembly action such as a movement or a machining operation.

REQ-STAB-STATE: Require that a part or subassembly be stably fixtured versus gravity [4].

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, 4].

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.

REQ-TWO-HANDED: Require that each assembly action involve two subassemblies [14]. This constraint is a very common assumption.

SUG-ACTION: Suggest that a particular assembly action or actions be included in the plan.

SUG-CLUSTER: Suggest to the planner that a certain cluster of parts be obeyed [3].

SUG-EARLY: Suggest that a particular part(s) be handled as early in the sequence as possible.

SUG-ORDER-GENERAL: Suggest a general order in which to add parts to the assembly. For example, rivets might need to be done approximately in order from one end of an airframe to another.

SUG-STATE: Suggest to the planner that a certain assembly state be used if possible (perhaps to speed the finding of a feasible plan) [1].

6 Organization of Constraints

In compiling the constraint list above, a certain amount of organization was imposed by the naming process. Next we want to categorize the list of constraints in ways that may provide insights as to their interrelations and possible approaches to implementation. We will use the categorizations introduced in Section 3. We restrict each constraint to being in one category in each categorization, which, again, may constitute oversimplification. All the categorizations below are combined into Table 1. In the table, constraints are sorted first by their entry in column two, then by their entry in column three, etc., and lastly by the constraint name.

6.1 Organization by origin

When we started this survey, we began with the *origin* categorization, but it turns out to perhaps have the least insight to offer. However, we think it is worthwhile to discuss here to help the reader gain some sense of the range of constraints in use and their purposes in practice. As outlined in Section 3, the majority of constraints mentioned in the literature are direct *assembly* issues. Examples include the need for an assembly to be stable with respect to a force during a subsequent operation (REQ-STAB-ACTION), and the need to perform all the assembly actions vertically from above (REQ-PATHS-AXIAL). Such constraints 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" constraints in the literature. We also include as assembly-origin constraints those that focus on the assembly process, such as MIN-COST-FIXT and MIN-TOOLCHANGE.

The remaining constraints arise primarily from the assembly *planning* process itself. Such constraints should be viewed differently than the others, in that an assembly planning system does not need to include them in order to be comprehensive, if the methods used in the planning system do not themselves require these constraints in order to be effective.

In the last column of Table 1 we have indicated the general origin of each constraint, and have added a subcategory comment regarding a more specific topic that constraint might be expected to address.

While not well shown in Table 1, some constraints 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).

6.2 Organization by function in planning process

Here we wish to organize the constraints according to their typical function in an assembly planning process. This function can be represented by the *obligation* and *scope* categorizations (see Section 3). In Table 1 we show in columns two and three all the six possible combinations of obligation and scope. (Column two is ordered by decreasing degree of obligation.) For example, REQ-MONOTONE is a *strategic requirement*, which is imposed to avoid the complication of moving a part into final position sometime after its initial insertion.[1] An example of a tactical requirement 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), or the requirement that a specific sequence of actions be used at some point in the assembly process because it is advantageous due to allowing simple fixturing, easy manipulation, etc. (REQ-SUBSEQ). Note that many tactical requirements, such as REQ-SUBSEQ, seem to arise because a person wishes to impose his/her insight into the assembly planning process.

Some optimization constraints, such as minimizing the cost of fixturing (MIN-COST-FIXT), are strategic,

[1] REQ-MONOTONE is implicitly implemented in most planning systems, because producing non-monotone plans is usually much more difficult than producing monotone plans.

CONSTRAINT NAME	OBLIGATION	SCOPE	INFO REQ'D	ORIGIN(SUBTOPIC)
REQ-LINEAR	requirement	strategic	each action	assembly(line layout)
REQ-PATHS-AXIAL	requirement	strategic	each action	assembly(simplicity)
REQ-TWO-HANDED	requirement	strategic	each action	assembly(simplicity)
REQ-MONOTONE	requirement	strategic	each action	planning(simplifying)
REQ-PATHS-STR	requirement	strategic	each action	planning(simplifying)
REQ-CONNECT	requirement	strategic	each state	assembly(fixturing)
PRH-COLLISION	requirement	strategic	each state	assembly(fundamental)
REQ-ACTION	requirement	tactical	a plan	assembly(various)
REQ-STATE	requirement	tactical	a plan	assembly(various)
REQ-STAB-STATE	requirement	tactical	a state	assembly(fixturing)
REQ-ACCESS-TEST	requirement	tactical	a state	assembly(manipulability)
REQ-ACCESS-TOOL	requirement	tactical	a state & action	assembly(manipulability)
REQ-SUBSEQ	requirement	tactical	a sub-plan	assembly(various)
PRH-SUBSEQ	requirement	tactical	a sub-plan	planning(reject a bad plan)
REQ-STAB-ACTION	requirement	tactical	each action	assembly(fixturing)
REQ-LINEAR-SUBSET	requirement	tactical	each action	assembly(line layout)
REQ-CLUSTER	requirement	tactical	each action	assembly(mfg. efficiency)
REQ-ORDER-FIRST	requirement	tactical	each action	assembly(mfg. efficiency)
PRH-ACTION	requirement	tactical	each action	assembly(various)
REQ-FASTENER	requirement	tactical	each action	assembly(various)
REQ-ORDER-LIAISON	requirement	tactical	each action	assembly(various)
REQ-ORDER-PART	requirement	tactical	each action	assembly(various)
REQ-PART-SPECIAL	requirement	tactical	each action	assembly(various)
REQ-SUBASSY	requirement	tactical	each action	assembly(various)
PRH-SUBASSY	requirement	tactical	each action	planning(reject a bad plan)
PRH-STATE	requirement	tactical	each state	assembly(various)
REQ-ORDER-SPECIAL	requirement	tactical	various	assembly(various)
MAX-LINEAR	optimization	strategic	a plan	assembly(line layout)
MIN-COST	optimization	strategic	a plan	assembly(cost)
MIN-DIREC	optimization	strategic	a plan	assembly(cost)
MAX-STAB	optimization	strategic	a plan	assembly(cost: fixturing)
MIN-COST-FIXT	optimization	strategic	a plan	assembly(cost: fixturing)
MIN-FIXT-COMPLEX	optimization	strategic	a plan	assembly(cost: fixturing)
MIN-REFIXT	optimization	strategic	a plan	assembly(cost or time)
MIN-REORIENT	optimization	strategic	a plan	assembly(cost or time)
MAX-PARALLEL	optimization	strategic	a plan	assembly(time)
MIN-TIME	optimization	strategic	a plan	assembly(time)
MIN-TOOLCHANGE	optimization	strategic	a plan	assembly(time)
MIN-SIMUL-LIAISON	optimization	strategic	a plan	assembly(reliability)
MIN-AWKWARD-ACTION	optimization	tactical	an action	assembly(manipulation)
MIN-AWKWARD-GRIP	optimization	tactical	an action	assembly(manipulation)
SUG-ORDER-GENERAL	suggestion	strategic	a plan	assembly(various)
SUG-EARLY	suggestion	tactical	a plan	assembly(mfg. efficiency)
SUG-ACTION	suggestion	tactical	a plan	planning(speeds the search)
SUG-CLUSTER	suggestion	tactical	a plan	planning(speeds the search)
SUG-STATE	suggestion	tactical	a plan	planning(speeds the search)

Table 1: Combined Table of Constraints

as they require that some numeric measure be summed over all the states or actions of an assembly plan, and are implemented as a search in the space of assembly plans. Others, such as minimizing the awkwardness of a particular action, are local, or tactical optimizations. However, note that a tactical optimization could become a strategic optimization if it is applied to all states or actions of a plan.

"Suggestion" type constraints, which are less common, also can be strategic, such as applying rivets generally from one end of an airframe to another (SUG-ORDER-GENERAL), or tactical, such as suggesting to a planner that a set of bolts be added as a subset if possible (SUG-CLUSTER).

6.3 Organization by information required

When implementing constraints in automated assembly planning software, a significant consideration is how much information is required to evaluate whether a constraint is being met. In Table 1 we have indicated in the fourth column what type of information is typically required to evaluate it. Most of the constraints fall into one of two categories: ones that can be implemented on the basis of local information, such as an action, and ones that require that an entire plan be derived before the constraint can be evaluated or a score obtained for it. For example, REQ-STAB-STATE could presumably 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. Or, to implement REQ-ORDER-PART constraints, each proposed assembly action can be compared to a list of part order constraints. On the other hand, all the strategic optimizations – and some others – require in general that an entire plan be proposed before a score can be obtained for optimization purposes. (We note that in some cases a score can be estimated from a partial plan.) Only a few constraints, such as REQ-SUBSEQ, fall into a category intermediate between these two extremes. We use the term *sub-plan* in Table 1 to refer to a sequence of actions smaller than a plan.

We should note that the information requirement listed for some of the tactical requirements may seem too localized. However, we have found that it is possible to implement a number of these constraints using only a description of each action, provided that that description includes a complete list of all parts in the subassemblies involved.

7 A General Approach to Constraint Implementation

This survey was performed to help create a plan for implementing constraints in an interactive extension of the Archimedes 2 system [9]. In this context, the obligation category seems the most important category of constraint for structuring a constraint implementation. Next most important is information required. We plan to structure our constraint implementation along these lines.

7.1 Requirements

We plan to implement nearly all of the tactical requirements, and some strategic requirements, as *filters* organized by information required. A filter will take a proposed state, action, or sub-plan as input, and return whether the input passes the filter's test.

It is somewhat surprising that all the strategic requirements, and most of the tactical requirements, need only local information to evaluate them. Thus, we can interact with the user in a coherent manner about a whole set of filterable constraints which are then implemented in a more or less integrated manner. In addition, the optimizing phase of the Archimedes 2 system will then accrue the benefit of this filtering function *a fortiori*.

The remaining strategic requirements are either implicit in the Archimedes 2 system (such as REQ-MONOTONE) or else can be implemented more efficiently as flags to the planner (such as REQ-LINEAR).

7.2 Optimizations

In Archimedes 2 we currently have implemented three degrees of optimization: a full state-space search, a search in a subspace of plans which use given subassemblies, and a non-optimized "first feasible plan" method. We plan to expand this set of search tools by providing a "beam" search, a greedy search, a K-lookahead greedy search, and variations of the subassembly-guided search. This will provide the user with a selectable level of computing effort, so he/she can trade off computing time with level of assurance of the optimality of the plan.

Once such a fairly general search strategy has been implemented in the space of assembly plans, a number of strategic optimization constraints could be offered to the user of an interactive planner by allowing the user to define a weighted sum of several metrics as an optimization criterion. We have experimented with this idea in Archimedes 2 using criteria such as "minimize the sum of tool changes plus subassembly inversions".

7.3 Suggestions

In this survey we found few "strategic suggestion" constraints, but we believe such constraints could be an effective tool for communicating between a human user and an interactive planning system to jointly find an appropriate plan for a large assembly. For example, it seems it would be useful to suggest to a planning system that the formal hierarchy of subassemblies defined within a CAD model be used as actual subassemblies when feasible, rather than requiring that they be obeyed, which might fail. Or, general preferences could be stated, such as a preference for assembling parts approximately from the assembly center outward. Such suggestions could efficiently guide the planning system toward finding an acceptable plan (or an optimal plan within a chosen subspace) in less time.

There is some question as to how to approach implementation of suggestions. We believe the right approach may be to treat the suggestion first as a requirement; then, if that fails to produce an acceptable plan, the requirement would be progressively relaxed in some manner until an acceptable plan is obtained. If an optimization is being performed the suggestion might be relaxed further if a better score is so obtained.

8 Conclusion

We have winnowed, formalized, and categorized a substantial body of constraints appearing in the assembly planning literature or of common knowledge. We hope this collection of constraint definitions and their categorizations will be useful to other assembly planning software efforts, or even be the beginning of a framework for more formal discussion of constraints in assembly planning. In any event, the perspective these categorizations provide has been of assistance in planning the implementation of constraints in the interactive version of Archimedes 2 which we are now developing.

References

[1] T. E. Abell, G. P. Amblard, D. F. Baldwin, T. L. D. Fazio, M.-C. M. Lui, and D. E. Whitney. Computer aids for finding, representing, choosing amonst, and evaluating the assembly sequences of mechanical products. In [6], pages 383–435.

[2] D. Baraff, R. Mattikalli, and P. Khosla. Minimal fixturing of frictionless assemblies: Complexity and algorithms. Technical Report CMU-RI-TR-94-08, Robotics Inst., CMU, 1994.

[3] 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*, pages 233–238. Pergamon Press, 1989.

[4] J. M. Henrioud and A. Bourjault. LEGA: a computer-aided generator of assembly plans. In [6], pages 191–215.

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

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

[7] 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.

[8] 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.

[9] S. G. Kaufman, A. L. Ames, T. L. Calton, R. E. Jones, C. A. Laguna, and R. H. Wilson. The Archimedes 2 assembly planning system: Implementation and performance. *Proc. IEEE Intl. Conf. on Robotics and Automation*, 1996.

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

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

[12] 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.

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

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

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

[16] 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.