Robotica (1998) volume 16, pp. 265–283. Printed in the United Kingdom © 1998 Cambridge University Press

Design of an experience-based assembly sequence planner for mechanical assemblies

Arun Swaminathan, Saghir A. Shaikh and K. Suzanne Barber

The Laboratory for Intelligent Processes and Systems, Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas 78712 (USA)

SUMMARY

This paper presents a design of an assembly sequence planner based on a "plan reuse" philosophy. Most of assembly planning research in the past has attempted to completely plan each problem from scratch. This research shows that stored cases of basic assembly configurations can be applied to a given assembly problem. It is observed that the number of such basic assembly configurations is quite small. The planner divides the assembly into a number of constituent configurations, which are called "loops". These act as subgoals in its search for solutions. Plans retrieved for all subgoals are fused into a set of plans that are consistent with the constraints implied by each plan. Application specific constraints on the assembly are explicitly handled in the second phase of planning. Mechanisms for assembly representation and implementation details of the planner are also presented.

KEYWORDS: Assembly planning; Case-based reasoning; Subgoals; Computational complexity; Precedence constraints; Precedence graphs.

I. INTRODUCTION

Assembly planning plays a major role in aiding shop floor control, production system design and scheduling activities. The assembly plan provides vital manufacturing information, and imposes constraints in the selection of production equipment and alternative routings. The planning process followed by a human planner is iterative. Humans examine the part and assembly drawings to create a tentative plan based on intuitive geometric reasoning. They subsequently revise the plans based on mechanical properties, assembly practices, managerial considerations, design notes, etc.[1] The approach commonly used in planning is to subdivide the task into smaller and smaller problems until the simplest problems are obtained. More information on the modeling and reasoning issues in assembly planning can be found in literature.[1–11] An insight gained from our research is that planning in the assembly domain is best served by considering many of the constraints at the local level before the global level. That is, combinations of locally feasible solutions are combined to give globally feasible solutions. The advantage is that the combinatorial explosion of plan options is controlled at a lower level without allowing them to propagate to higher levels. This task needs to be supported by an appropriate identification of "localities". Localities are a suitable subdivision of the problem in which planning for each subgoal can be performed relatively independent of each other.

In this paper, the Assembly Planner using Experience (APE) is proposed. APE demonstrates that the Case-Based Reasoning (CBR) paradigm can be quite effective in the assembly domain. APE utilizes case-based planning to store, retrieve, and modify existing cases (experience) to develop assembly sequence plans. This paradigm is very powerful for complex domains but has not been applied to the assembly domain in a significant way. The focus of this research is to explore application of a planning paradigm, to combine both geometric and application constraints and to leverage past experience to resolve these constraints. The contributions of this research are the following:

- To demonstrate some aspects of an efficient representation of assembly information to support the re-use of past experience.
- To demonstrate how a large variety of possible assembly parts configurations can be effectively reduced to make the search and retrieval problem of past experiences (cases) computationally feasible for practical sized assemblies.
- To demonstrate the ability of APE to plan directly when the need arises, thereby providing a hybrid methodology which allows; (i) re-use of stored cases if they are found in the database, and (ii) for situations not found in the database, "primitive connections"* between parts are employed to form larger plans.

Assembly modeling, which involves the representation of both the assembly and the plan, is an important aspect of the planner design. In APE, graphs are used to model the assembly. Graphs provide a formal, efficient and flexible representation. The assembly is represented by three kinds of graphs: Connection Graph, Mating Direction Graph, and Obstacle Fact Graph. The plan is represented by an Assembly Precedence Graph. Formal definitions of these graphs and their properties are defined in this paper.

This paper is organized in six sections. The introduction is followed by Section II, which discusses the recent work in assembly planning and the case-based model of reasoning. This section also explains why CBR is useful for assembly planning. Section III describes the assembly modeling using graphs and the concept of "goals" in assembly. Section IV

* A "primitive connection" simply shows how two parts may be connected (i.e. the first part could be placed in the assembly and then the second could be connected to the first or *vice versa*).

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

presents the actual reasoning process, combining graph theory and CBR in a search for feasible solutions. Section V addresses the implementation issues of the APE system. A summary of the research is presented in Section VI.

II. Related work

This section reviews some previous work addressing modeling assemblies, reasoning approaches and application of case-based paradigm in planning.

A. Modeling for assembly

The information found in the assembly model is an important part of the input to the assembly. In order to represent and reason about the assembly, this knowledge must be in a compact and accessible form. The design of assemblies for manufacturability and maintainability is a difficult problem. According to Foundyller, "Engineering time associated with the designing, debugging and documentation of assemblies, as opposed to the time spent on individual components, often consumes most of the engineering effort that goes into the development of a new product. . .".[12] In order to help designers visualize the design, expensive prototypes and clay models are often used.

A detailed assembly model should consist of detailed geometric information, information about the components, information about the different types of contacts and information about the production environment of the product. Sanderson et. al.[3] used a model which includes three types of entities: parts, contacts and attachments (the contact types). It starts with the detailed part geometry and assembly configuration from the design stage. In addition to this information, they also explicitly represent of the attachment that hold the parts together. This attachment information is added interactively to the relational model. Thus, there are three levels of the product model: (i) the CAD model of assembly, (ii) a relational model which indicates the contacts between parts and (iii) a graph with the attachments added that describes the constraints on the degrees of freedom within the structure.

Henrioud and Bourjault represent the assembly information in a 5-tuple that comprises the components, connections, fasteners, set of transformations, and a mapping from each attachment to the set of components or functional features.[5] Santochi and Dini use a "table of contact" formalism in their FLAP (FLexible Assembly Planning) system.[13] Contacts between parts, and the feasible disassembly directions are determined by simulation of movements using a CAD system and detecting collisions. Lin and Chang derive graphs of mating directions and spatial constraints by analyzing the geometrical model built in the TWIN modeler.[1] In addition to the geometric data, non-geometric data is stored in a "frame based" symbolic representation. The three major types of non-geometric information are standardized machine elements, mechanical fasteners and assembly design intents. Delchambre suggests a structured model of assembly containing almost all the required information.[2] In his model, the geometrical information specifies the shape and dimensions of the parts as well as their relative positions within the final assembly. Component information includes the features of the components and their roles in the assembly. This information is usually obtained from user interaction. There is also the topological information that indicates the type of contacts between the parts in the assembly. In addition to these, there is the final assembly information that defines the features of the whole assembly (e.g. product functionality) and the technological information about the manufacturing facilities (e.g. which fixtures are available).

Anantha describes a system for feature based assembly modeling that enables a user to design mechanical parts based on features and explicitly specify the spatial relationships between the part features.[14] It also enables the user to characterize the remaining degrees of freedom in terms of kinetmatic joints that capture some of the functionality of the assembly. This allows a treatment of over-, under- and fully-constrained assemblies. The system employs symbolic reasoning about the geometric structure of the parts to satisfy the spatial constraints between them rather than the algebraic methods used by other researchers. Shah and Tadepalli also discuss feature-based assembly modeling.[15] Venuvinod describes an algorithm that identifies the principal disassembly directions by an iterative process of analyzing the contact planes.[16] These define the visual region of the part. A part is said to be nominally disassembleable if its visual region is unbounded. For confirmation, a second stage is used to check the global translation freedom of each component. Instead of checking the translational freedom in all the directions, he proves that it is sufficient to check along the central disassembly direction and two orthogonal directions. Wolter and his colleagues have analyzed mating constraint languages which are the most commonly used constraint languages for assembly planning. They consider relations of the type "equal/not equal"(==, =/=), "less-than" (<), and "less-than-or-equal" (=<). They have proven that it is possible to translate any constraint expession comprised of the above relation types to one comprised of "<" or "<=" relations only.[17]

An important stage in the planning process is the generation of the precedence relations. All the different types of constraints have to be converted to precedence diagrams by the planner, from which the assembly plans can be generated automatically. Some researchers use the notion of "preconditions" and "post-conditions" directly.[17] A precondition is a set of states after which it is possible to perform an action. A post-condition is the set of states after which it is impossible to perform the operation. More frequently, the precedence constraints are used to implicitly define the pre- and post-conditions. If a sequence is legitimate, each operation in the sequence must be possible. Therefore, no operation must be prevented by past actions nor should it obstruct later actions. In other words, if the actions in a sequence satisfy the pre- and post-conditions, then the operations in the sequence can be performed in order.

In order to represent the precedences, different precedence operators have been suggested: Lin and Chang use "must_follow" and "must_immediately_follow"[1] whereas Delchambre suggests the use of "must_precede" and must_

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

strictly_precede".[2] In reference 1, a generalized scheme which includes all the fundamental constraint relations and from which complex ones can be built by boolean operations, is proposed. These may denote constraints between two parts (P1 > P2), a joint and a part (P1-P2 > P3), which means that the connection P1-P2 needs to be created before part P3 is placed in the assembly), or between two joints (P1-P2 > P3-P4). In reference 2, constraints are classified into hard constraints and soft constraints. Hard constraints are created due to the intrinsic property of the assembly (i.e. geometric feasibility, mechanical constraints (due to attachements) and component constraints). Soft constraints are further divided into stacking constraints and technological constraints.

Henrioud and Bourjault have considered two types of constraint: (i) operative constraints from geometry, material handling requirement and stability and (ii) strategic constraints which they obtain from imposed subassemblies, a linear assembly tree requirement, etc.[5] In Sanderson's work, the soft constraints are referred to as "state feasibility conditions" based on rigidity and stability whereas the former (hard constraints) are called "task feasibility conditions".[3] Sanderson and colleagues also use constraints of the type "connection to connection" and "connection to state". The precedence inference methods described above can be classified as either "query-and-answer" methods or "geometric reasoning methods". There is a further class of inference approaches called "knowledge based" that use rules for grouping parts.[9, 18, 19] It is claimed that these methods are superior for axial assemblies or orthogonal assembly directions since they avoid the extensive computation due to redundant geometric data.

B. Reasoning for assembly

The approach commonly used in planning is to subdivide the task into smaller and smaller problems until the simplest problems are obtained. Sanderson et al.[20] abstract the task decomposition problem into a search for cut-sets* of a graph of connections. They suggest the AND/OR formalism for the representation of selected assembly sequence. They also discuss evaluation functions for assembly sequence plans based on minimizing complexity, number of alternative sequences and the depth of the assembly tree. Sanderson and Homem deMello also discuss a formal algorithm that is correct and complete for finding assembly sequences.[4] However the utility of this algorithm is limited by the fact that it is mainly an exhaustive graph decomposition algorithm. Thus it allows the combinatorial explosion of solutions before constraints are applied and the solution set is pruned.

Wilson and Schweikard have found that give an assembly of k polyhedra with a total of n vertices, a valid translation and removable subassembly is found in $O\langle k^2 \cdot n^4 \rangle$ steps.[21] Based on this finding, they derive a polynomial time algorithm for determining feasible assembly sequences consisting of single translations. Santochi and Dini describe an assembly system that is able to plan for assemblies where Design for Assembly (DFA) principles have been applied.[13] Their primary approach is to identify subgroups within the assembly and generate all possible sequences for the subgroups and the entire product. The number of sequences is then reduced by considering precedence, accessibility and stability conditions. Browne et al. describe a two-stage planning approach where the first stage analyzes the assembly for compliance with DFA principles before planning.[22] The planning engine is rule-based and implemented in the OPS5 language. The rules cover situations where a part (or subassembly) is placed in a fixture, two subassemblies are joined, the master part is removed from assembly and so on.

The more sophisticated assembly systems for 3-D parts are described in references 1 and 2. Lin anc Chang describe a powerful method for performing the assembly sequence generation for 3-D mechanical parts.[1] They develop graphs of mating directions and spatial constraints from analysis of a boundary representation (B-Rep) geometric model. A three level planning strategy is proposed that analyzes the part connectivity relationships and plans for collision free insertion of individual components. The non-geometric data is also converted into precedence relations and rules are used to prune the graph obtained from the first three stages. They consider assemblies that have one base or main part. Delchambre develops an automated assembly planning system which is synthesis based.[2] Extending the algorithm by Sanderson,[4] Delchambre developed a synthesis based assembly planning system using precedence constraints at the start of planning. The general criteria applied to select sequences, such as the number of product reorientations, number of fixtures, number of unstable assemblies formed, and degree of parallelism are also discussed.

Cao has applied reasoning based on Petri-net representations to situations where intermediate processes during assembly may modify the properties of the part and create additional constraints that did not exist at the start of planning.[23] Arai and Iwata extend the scope of the assembly problem usually dealt with by researchers to include mechanism.[6] They address the problem of kinematic simulation and evaluation functions needed when designing mechanisms, using disassembly approach. Each part is studied to determine if it can be removed from the assembly. When there is more than one way to disassemble a component, the shortest path is taken. In calculating the movements, they also consider errors that appear due to representation of cylindrical and spherical components in a CAD system. Their system also supports disassembly in more than one step.

C. Case-based reasoning

Case-based reasoning (CBR) is an approach to solving problems in complex domains by reusing and/or adapting solutions from past experience. Case-based planning differs from rule-based planning in that rule-based planners require a large number of rules to account for all options. Rule-based planners are constrained to small, well-defined domains. Rules must account for each and every condition of uncertainty. In complex domains, rule-based systems quickly become cumbersome due to the number of rules to

* A cut-set is a set of edges which when removed, breaks up a graph into two or more partitions.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

be defined and managed. In addition, most real world domains are so complex that providing a complete set of rules is impractical or impossible. This is especially true of the interactions between actions in the assembly domain.

Case-based planning manages to control complexity by focusing on the required knowledge based on the context. All the information available is not stored. Only previously developed plans and information required to retrieve and adapt old plans are stored. For assembly planning, the graph description of the assembly and constraints on the assembly are stored. Adaptation of experience is employed as a learning mechanism. Typical rule-based systems do not have the ability to create new rules or change old ones depending upon their planning experience.

Case-based reasoning is derived from cognitive studies based on the nature of information storage in the human brain and the reasoning process. Human expertise is rooted in "stories" or "episodes" that are rich in detail.[24] Knowledge acquisition studies show that people find it easier to describe a series of events or a case rather than identify rules that made them decide on an action.[25, 26] The basic idea in CBR is simple. A case-based reasoner solves new problems by adapting solutions that were used for similar problems in the past.[24] Case-based planning is driven by the understanding that most problems in the real world are complex and planning them from scratch is quite tediouis. However, problems within a domain have similar characteristics. Thus, we are able to reuse old solutions. A case-based planner finds those cases in memory that are solutions to problems similar to the current problem. The planner then retrieves the old case, either reuses it directly or adapts it to suit the new problem.

Most case-based planning work addresses areas other than engineering. CYRUS is a story understanding program developed by Kolodner.[27] Stories about the diplomatic travels of a person are studied. Questions are asked to determine the system's understanding or comprehension of stories. If it does not have an answer directly, the system generates a series of sub-questions. The generation of answers to these sub-questions helps to answer the main query. IPP (Integrated Partial Parser) is another text understanding program.[28] IPP reads texts about terrorist activities and makes generalizations. IPP uses these generalizations to guide its future interpretation of news stories. The JUDGE program models a judge who is deciding sentences for convicted criminals.[29] JUDGE uses information such as the charge made, the events that occurred and legal statutes for that crime to reach its decision. This domain is a subjective domain unlike the assembly domain. Consistency is an important feature of JUDGE, independent of whether the decisions were "right" or "wrong". CHEF, an innovative Chinese cooking program built by Hammond at Yale, was an important influence on the development of the APE system.[30] CHEF generates Chinese dishes starting from an initial base of 20 recipes. The goals specified are usually the ingredients that one wants to be included in the dish. CHEF has a memory organization that promotes learning from its failures.

A case-based reasoner developed by Rechsberger and Pu exists in the assembly planning arena.[5] Their observation is that "If assembly sequence generation problems are costly, then not all of them should be solved from scratch." The system uses a vector of features such as the part's mating direction and overlap of its projected area with the projected area of other parts as the index key for each case. The causality evaluation of Rechsberger and Pu's system was limited because it tried to trace the problem to a single step. Since it only looks at single actions, the system does not have a sense of global interactions between components or subassemblies. Also, it provides only one plan and if a new constraint appeared the system replans from the beginning. A more recent work by Pu and Purvis (CORINTH) extends their previous work and adds a formal procedure for case adaptation.[31] This system is able to generate multiple plans; however these plans are generated one by one without capturing common portions. This again prevents identification of potentially parallel actions. It also seems that derivation of "require not" predicates would be difficult to automate. On the other hand, CORINTH's constraints-based matching algorithm is well suited for making approximate matches. Overall, APE and CORINTH seem to have complementary abilities. CORINTH uses spatial predicates such as "(press-fit 2 1)" or "(overlap 1 3)" in describing a case. These are very specific to that assembly. Thus, quick matches for very similar assemblies can be made. However, if the given problem is somewhat different (but topologically quite similar), it may result in imperfect matches and require repair procedures. APE, on the other hand, was designed to work at a higher level of abstraction. Thus, it does not use such predicates which are assembly specific. Instead, it uses the "loop" primitive to capture the essential (or generic) aspects of the assembly. The advantage is that APE is able to handle a wider variety of assemblies. The drawback is that in some cases, it will spend more effort in obtaining matching cases.

Chakraborty and Wolter[32] have presented a generalized reuse oriented solution to assembly sequence planning. They use a small set of primitive structures repeatedly to build up hierarchical structures of the assembly. However, the user is required to identify the different substructures in an assembly. This would be quite tedious for the user in the case of large assemblies. Stored plans for the substructures are combined to form a generalized plan. Constraints arising due to specific geometries of parts as well as user-defined constraints are not taken into account. Their paper proves two useful theorems. The first shows that plans obtained by systematic merging of correct primitive plans are also correct. The second theorem shows that if the sets of primitive plans are also complete, systematic merging of the plans for primitive structures comprising the assembly produces a complete set of plans for the assembly. Both of these theorems apply to APE. Thereby, the systematic approach to merging correct primitive assembly plans in APE is shown to be correct and complete under the assumption that all relevant constraints have been modeled.

This research demonstrates that CBR paradigm can play a much more effective role in assembly planning. An important requirement for case-based planning is the appropriate definition of "goals". This is crucial since it is with these goals that the planner is able to index its memory.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

It should be noted that in all the planners discussed the goal identification is relatively simple. Even in the assembly planner of Rechsberger and Pu, the goal is simply to find a part that can be placed next. However, to enable an overall view of the assembly planning problem, this goal definition is not complete. Our treatment of this issue is discussed in Sections III and IV.

III. A GRAPH-BASED ASSEMBLY MODEL

Modeling of the assembly problem is an important aspect of the planner design. The assembly model in general needs to accommodate both geometric and non-geometric information. The geometric information consists of the solid model representations of the parts, their positions and the information about the connections that comprise the assembly. Though the assembly problem is essentially a geometrical one, the non-geometrical information is important. The inclusion of non-geometrical information helps to reduce the explosion of possible solutions to a set that embody the limitations placed by the design intents, special properties of components and fasteners, and technical constraints. The planner, presented here, assumes: (i) that all the parts of the assembly are rigid polyhedral, (ii) there is no default main or base part, (iii) almost any part can be a base part, (iv) all the parts are moved from infinity to their final positions in the assembly in one translation (no rotational aspects), (v) there are no internal forces in the assembly, once part is placed it is not moved, and (vi) partial assemblies are stable. The assembly is assumed to be connected, i.e. there is no loose part or a disconnected subassembly. Only one kind of constraints, i.e. of the form "part-->part" are used.

A. Assembly representation

By preprocessing the geometrical model, it is possible to abstract the information contained in the CAD files into a set of graphs which can then be conveniently and efficiently manipulated without repeatedly accessing the CAD design file.[1, 10] The APE system begins planning from this graph data. Preprocessing of CAD files is not limited to a specific method; either geometric reasoning or knowledge based approaches can be used, provided they produce output in the form that APE requires. The knowledge based approaches may be better in situations where a designer makes a small local change in the assembly to minimize reprocessing. The information needed for assembly planning can arise from different sources. As described earlier, these could be geometric, part information, technological information, and so on. This research deals essentially with the geometrical aspect of the information. However, the proposed model is scalable to represent constraints (i.e. cost or priority associated to certain subassembly) arising from other sources. The other sources can be some problem-specific databases which provide this information as additional constraints. An example of a constraint that arises due to the tooling and the subsequent required modification is illustrated in Figure 1. While the first plan shows that parts *A* and *B* could be placed into the assembly independent of each other, the second plan shows that *A* needs to be placed after *B* due to the constraint. When these types of constraints are applied to the graph originally obtained from the geometrical planning, the graph reduces significantly.

To explain the geometrical model more clearly, a simple example is shown in Figure 2 with all the derived graphs. This pen assembly example was used in Pu.[33] By using the same example an informal comparison between APE and Pu's system is established. Basic characteristics of an assembly. $\mathcal{A}$ consisting of *n* parts are:

- Set of Parts $(P)=\{P1, P2, P3, \ldots, Pn\}$ of the assembly.
- Set of Connections $(C)=\{insert, attach, \ldots\}$ made between the parts.

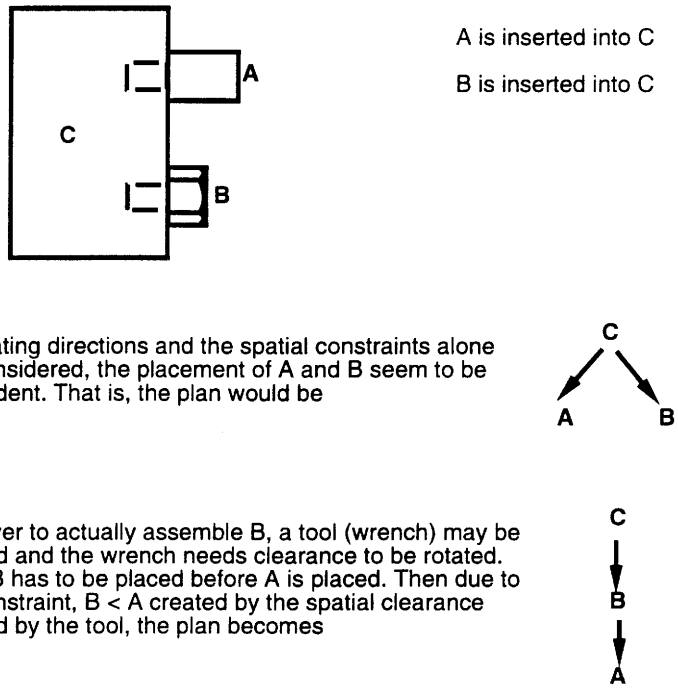


Fig. 1. An Example of tooling constraint.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

- Set of Directions $(D)=\{+X,\ 2\ X,\ +Y,\ 2\ Y,\ +Z,\ 2\ Z\}$ in which parts can be moved from infinity to the assembly.
- Set of Rules $(R)=\{rule\#1, rule\#2, rule\#3, \ldots\}$ to define the constraints under which a particular part can not be placed in it required position at required instance in assembly.

The geometrical information is essentially composed of three types of facts: (i) all the different connections that the parts make with each other, (ii) the different directions in which mating occurs between parts, and (iii) the conditions under which a part cannot be placed in its required position due to other parts obstructing its path. This form of a model for abstracting the assembly information is suggested by Line.[1] This information is represented by graphs. Graphs provide a formal, efficient and flexible means of representation. For any specific assembly, $\mathcal{A}$, a set of graphs are defined as follows using the above basic characteristics of an assembly:

(i) For an assembly, $\mathcal{A}$, there exists a single *Connection Graph CG(X, E)*, which represents inter-connections made between various parts. A node in *CG* represents the part and an edge between two nodes represents the connections made between the corresponding parts. Here $X=\{\ ;\ x_i|x_i=p:p\langle\langle P\}$ and $E=\{x_i|\ x_i=c:c\langle\langle C\}$.

A *CG* is an undirected graph consisting of n nodes with degree $d(x_i)<=1$. Figure 2(a) presents a *CG* for a pen assembly consisting of five parts; body (*B*), cap (*C*), tube (*T*), button (*Bu*) and head (*H*). The only type of connection in this example is *insertion*.

(ii) For an assembly, $\mathcal{A}$, there exists a single *Mating Directions Graph*, *MDG*(*X*, *E*), to identify the directions that are available for each part to connect with its mating parts. Here $X=\{\ ;\ x_i|x_i=p:p\langle\langle P\}$ and $E=\{x_i|x_i=d:d\langle\langle D\}$.

MDG is a directed graph with n number of nodes. Each directed arc $(x_i,\ x_j)$ represents the direction in which the source node x_i can mate with the destination node x_j. Thus, the *correspondence* for a node x_i, $G(x_i)$ indicates all its mating directions and *inverse correspondence*, $G^{2i}(x_i)$ shows the directions in which other parts can connect to it. In the planning process one needs both sets of arcs due to nature of the algorithms used and to resolve any ambiguity. For example, in Figure 2(b) the pen assembly. *H* mates with *T* by moving in the +Y direction, whereas *H* mates with the part *C* by moving in the 2 Y direction. *MDG* is the most typical representation of the assembly. It provides the spatial description of all possible solutions from which all the infeasible solutions are removed using the constraints

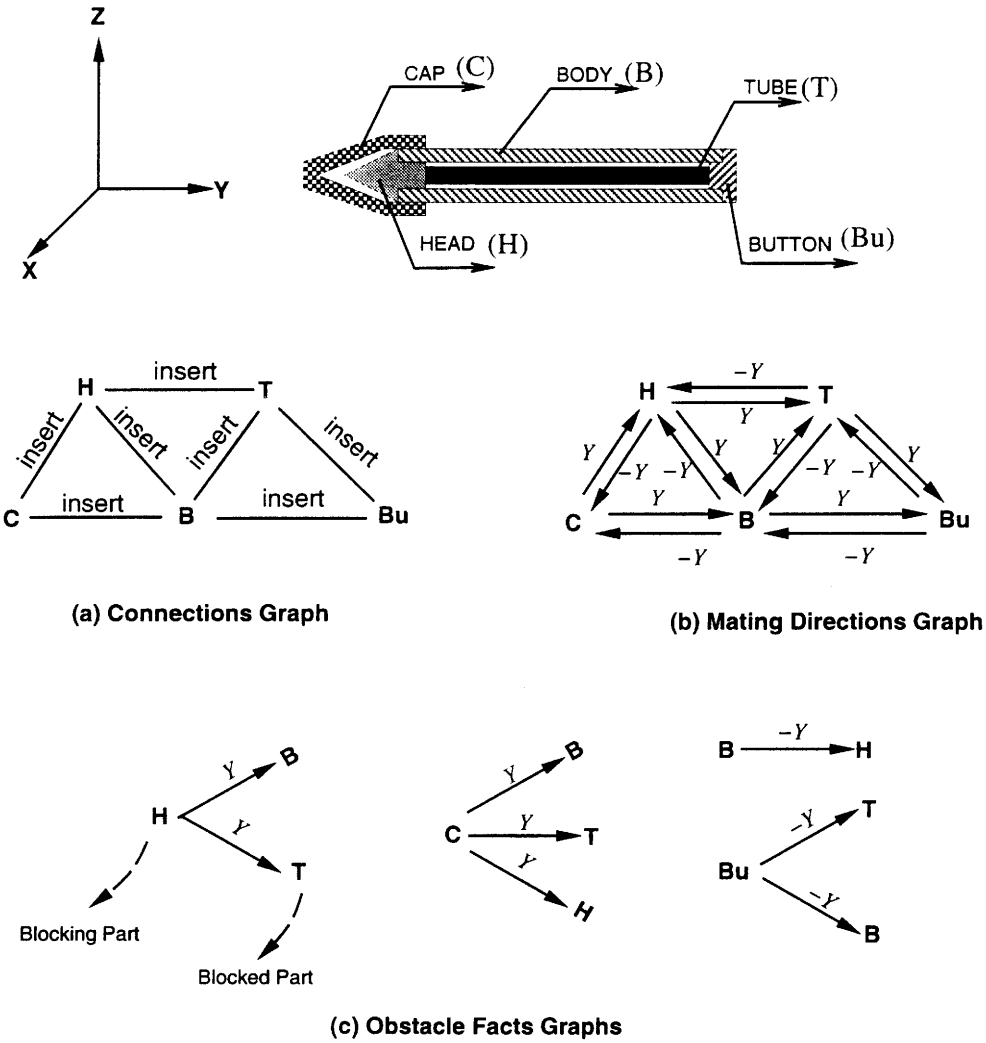


Fig. 2. Pen assembly example.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

specific to that problem.

(iii) For an assembly, $\mathcal{A}$, there exists from zero to n number of *Obstacle Fact Graphs*, $OFG(X, E)$. Each *OFG* corresponds to a part which obstructs the mating of other parts along certain mating directions if that blocking part is placed earlier in the sequence than the other parts. Here $X=\{r, x_i \mid x_i=p : r, p \ll P\}$ and $E=\{x_i \mid x_i=d : d \ll D\}$.

Each *OFG* has one source node r, denoting the obstructing part, and one or many nodes x_i, representing the parts that are being obstructed. The arcs between the root and other nodes denote the direction in which the obstruction occurs. *OFG*s are directed graphs and are constructed using the Set of Rules (R). In Figure 2, the T blocks the mating of parts *Bu*, *C* and *H* in the respective directions shown. The *OFG* differs from the Spatial Constraints Graph (*SCG*) used in reference 1. In *SCG*, there is a graph for each part showing the parts that obstruct it. While in *OFG*s, there is a graph for each part that obstructs other parts. This is more suitable for our algorithms since these algorithms are based on depth first search technique. Each node can be processed only by taking into account the nodes that are in its subtree. While in the *SCG* structure, all the *SCG*'s have to be scanned to obtain the parts that are obstructed by one part.

B. Plan representation

The representation problem for assembly plans has received some attention due to the requirement of less storage and easy user understanding. The storage requirement is understandable since in assembly, unlike other domains, the number of sequences(plans) that are produced can be quite large. Assembly plans have been represented by diamond graphs,[8] AND/OR graphs,[3,4] assembly trees[2] and assembly precedence graphs.[1] In the "Diamond graph" representation, the state of the assembly is shown by a rectangle filled with smaller rectangles. Each smaller rectangle represents the formation of a connection. In the initial state, the large rectangle is empty and in the goal state is completely filled. This representation requires less space but is quite difficult for a viewer to understand. AND/OR graphs have been used by Sanderson and deMello.[3] Successors of a node are grouped into pairs where each pair shows one way in which the parent could be divided. Each pair is combined by an "AND". The pairs are related to each other by "OR". This representation supports Sanderson and deMello's disassembly based planning algorithm.[4] However, the AND/OR representation is very difficult to interpret. Delchambre suggests the use of assembly trees.[2] The nodes represent subassemblies and branches show the actions. This representation shows the assembly starting from the base part and not as a disassembly graph (as in the AND/OR graph). When a part or subassembly is added, it is shown by the addition of an arrow. The links indicating the parts to receive these components do not have arrow heads. The intersection of an arc without an arrow and an arc with an arrow shows the formation of connections between the nodes they represent.

In this design, *assembly precedence graphs* (*APGs*) are employed for plan representation. In the APE system, additional information is provided by labeling the edges with the mating directions of parts. An *APG* is an acyclic directed graph (tree) with n number of vertices. The arcs between the nodes represent the order of precedence. For an *APG*, the following statements hold true:

(i) If x_i *precedes* x_j, it implies that part x_i is to placed before the part x_j.
(ii) If x_i *precedes* x_j, then x_j *succeeds* x_i.
(iii) If x_i *precedes* x_k and x_k *precedes* x_j, then *precedes* x_j.
(iv) If x_j is located in the subtree of x_j, then x_j is below x_i and x_i is *above* x_j.
(v) A part can be placed in the assembly only after all its predecessor parts have been placed in the assembly.

In the example shown in Figure 1(a) a part *C* is a predecessor to parts *B* and *A*. Similarly, *A* and *B* are successors to *C*. Thus, parts *B* and *A* will be placed after *C*. The parts can also be placed in parallel (assuming there is no other constraint). *APG* is capable of representing the parallelism in the precedence constraints and can efficiently be stored in less space.

C. Nature of assembly subgoals

The case representation within the case-base must be efficient. It must be efficient in size so that the potentially large number of cases can be represented, yet also powerful enough to effectively support the planning task. Different case representations are used by different planners. However, it appears that each domain has a different representation that would serve it most usefully. For

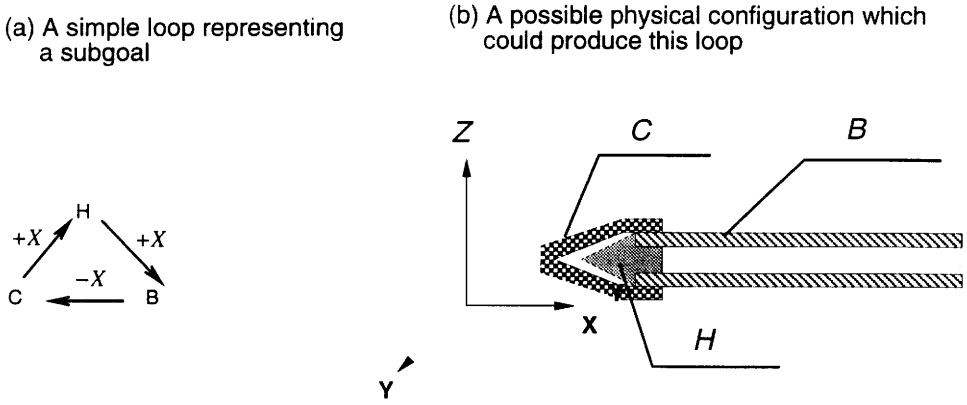


Fig. 3. An example of a loop representing a subgoal.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

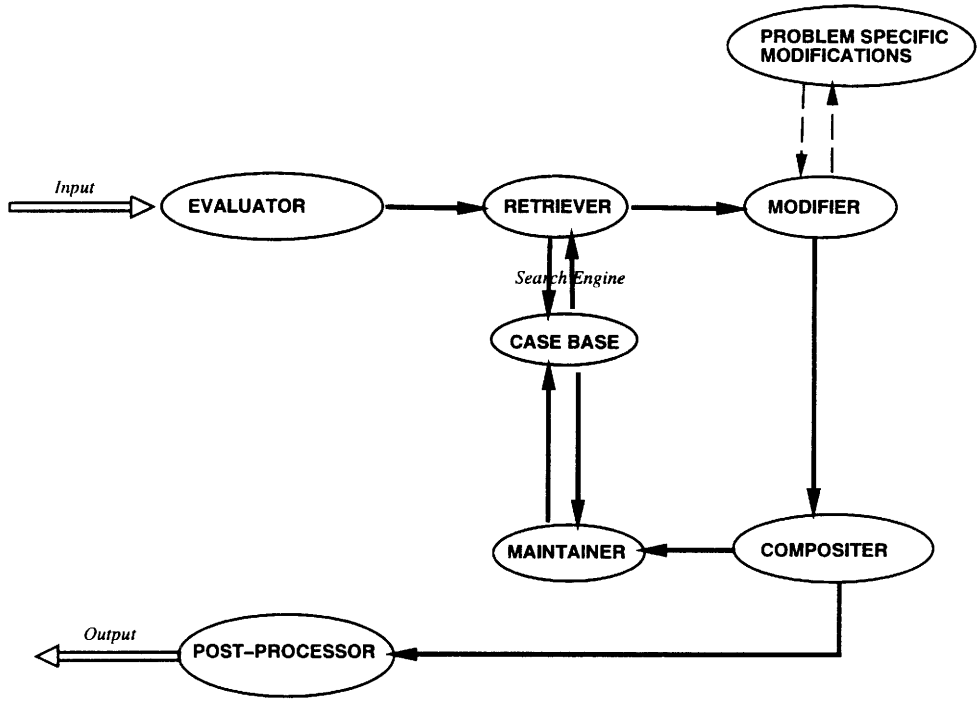


Fig. 4. Planning schematic of APE.

example, in CHEF,[30] the attributes of a dish are used to index into the case-base. The goals could be "to include a certain ingredient" or "to ensure the texture of a certain ingredient". There is also the notion that some goals are more important than the others. The search process is aware of these goal priorities. In the assembly domain, it is not clear that analogies of such goals exist. The goal is to ensure that all required connections are created in the proper directions. It is not obvious that some connections are more important than others. The overall picture is not localized but is dependent upon the mating directions and the spatial constraints. Thus, the APE system indexes and stored cases based on their connections, mating directions and spatial constraints.

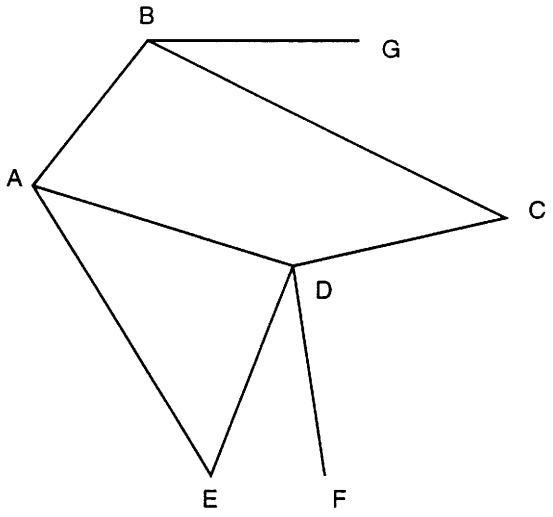


Fig. 5. Loop nodes and non-loop nodes.

The subgoals we have defined are called "loops". A loop consists of a sequence of mating parts including the directions in which parts mate with the part followng them in the loop. In Figure 3, for example, *C* can mate with *H* by moving along +X axis, *H* can mate with *B* along +X and *B* in turn can mate along 2 X with *C*. This situation gives rise to a loop shown in Figure 3. One should observe this kind of information about loops is already embedded in our assembly representation (i.e. *CG* and *MDG*). The degree of subgoal interaction is quite high in assembly planning. This is because each subgoal is comprised of parts that may also be included in other subgoals.

The part ordering derived from one subgoal may contradict a part ordering from another subgoal. This is not always the case for other case-based planners. Most of the goals in other domains are non-interacting and the plans for non-interacting subgoals are easy to combine. For the goals that are found to interact, certain repair rules based on causality are applied. In the assembly problems, every step has interactions. These planners must perform a "plan repair" task every time plans combination for different subgoals is attempted.[33]

Indices to the APE case representation consist of the set of assembly graphs. The constraints portion of the graphs can be modified later to expand the number of constraints. In addition, the case also contains the *APG* which represents the solutions to that problem. At the conclusion of planning, the *APG* implicitly satisfies all the precedence constraints that were considered during planning. The situation where two cases conflict does not occur here, as it is assumed that the set of constraints is consistent and each constraint is essential.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

Direction String	Family Number	Reorient Status	Swivel Status	Reverse Status	Cycle Status
+Z,–Z,–Z	1	2	0	1	0

Fig. 6. An example of loop parameters.

IV. PLANNING STRATEGY

The planning strategy developed for the APE system is shown in Figure 4. The utilization of experience represented in cases is fundamental to the approach. The concept of feedback in this strategy is inspired by the model developed by Hammond.[30] The planner is composed of the following modules:

(1) *EVALUATOR* module: The input to this module is the problem description. It extracts the different sub-goals from this description.
(2) *RETRIEVER* module: This module uses the subgoals generated by the evaluator to find "best" matches from the database of cases.
(3) *MODIFIER* module: The Modifier matches from the *RETRIEVER* and performs a series of adjustments to the case solution to adapt it to the problem at hand.
(4) *COMPOSITER* module: This module combines the retrieved plans to create a set of plans for the current problem. The *COMPOSITER* also has the ability to plan from first principles for subgoals not found in the retrieved plans.*
(5) *POST-PROCESSOR* module: This module takes the newly formed plans and applies the different constraints that it must satisfy. Constraints need not only be geometrical; application relevant constraints could also be considered. Plans that fail to satisfy the constraints are removed at this stage.
(6) *MAINTAINER* module: Before the *POST-PROCESSOR* stage, the *MAINTENANCE* unit decides whether the newly generated plans should be stored. The plans are stored after their composition but before the application of constraints to obtain a generic plan for that set of goals. This is more useful than storing a set of plans influenced by the effect of constraints that may be specific to that problem. The set of constraints to be satisfied are then applied to the plans. After applying the constraints, the remaining set of plans are provided to the user. The *MAINTENANCE* module also stores failure history for assembly designs that failed to generate even a single plan. These may be due to extrinsic constraints that are too restrictive or assembly designs that are physically impossible.

* A first principles approach would start with the basic plan, "part *A* is connected to part *B*". It then merges each of these "single step" plans to form a plan for the subgoal.

A. Goal identification

For a case-based planner, the most important function is to retrieve previously stored information. If the planner can find the correct past case, subsequent modifications are reduced. APE retrieves and plans using "micro cases". These "micro cases" or subgoals are obtained from the assembly representation. The initial step is to recognize that the *CG* is composed of nodes that are members of "cycles" or "loops" and nodes that are not members of cycles[1] (see

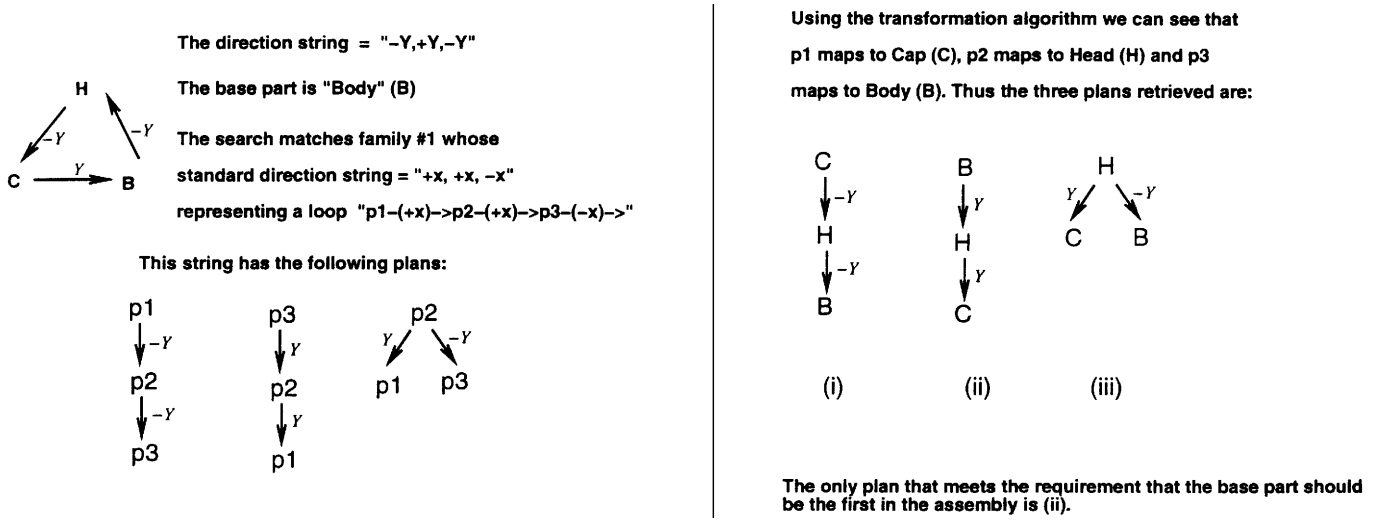


Fig. 7. Plan retrieval.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

Figure 5). These nodes are called “cyclic” nodes and “non-cyclic” nodes. It can be observed that in the assembly domain, a non-cyclic part can be placed in the assembly after its connecting part is placed. The part that mates with a non-cyclic part can be a cyclic or a non-cyclic part. There are also interactions between parts of one loop and another. This situation tells us that in a general sense the non-cyclic parts are responsible for much less complexity than the parts that form cycles.

For parts that comprise cycles, there are different sequences by which they can be assembled. This research determined that the loops of parts provide a convenient identification of *locality* (i.e. planning for subgoals which are independent of its adjacent parts) within an assembly. In addition to defining the loop, the proper mating directions of the parts are required. This is obtained from the *MDG*.

There are other ways of identifying locality of constraints. For example, one may focus on a node and identify the mating parts along each of its mating directions. However, the identification of loops is more useful. Focussing on individual nodes is not suitable since it does not provide a unique way to index a case. Moreover, it also does not show the constraints imposed by neighboring parts on the placement of a part. Therefore, subgoals are to locate the plans for each individual loop. Thus, after removing all the non-cyclic nodes, a graph is obtained that is composed of only nodes that form loops. The strategy is to first plan for the nodes that are in this reduced graph and then add the non-loop parts.

B. Problem evaluation

The first step in obtaining the subgoals for a certain assembly problem is to decompose its *CG* and *MDG* into a set of directed simple loops. The approach for this purpose is to create a spanning tree of the *CG* and to identify the loops by adding the missing edges one by one. The missing edges are the edges which were initially removed to form the spanning tree. Every addition of a missing edge provides an additional loop. For generating the spanning tree, any convenient algorithm may be employed since the algorithm for generating cycles does not require any special properties from the spanning tree. In this implementation, a depth first search (DFS) traversal algorithm of the graph produces a spanning tree.[34]

Along with the creation of the spanning tree, a list of the edges removed from the connections graph to form the spanning tree is also collected. This information, along with the *MDG* is used to generate the set of directed loops that form the assembly. It should be noted that the relationship between this set of loops to the assembly problem is not one-to-one. The set of loops produced from an assembly is unique, however a certain set of loops can be generated by

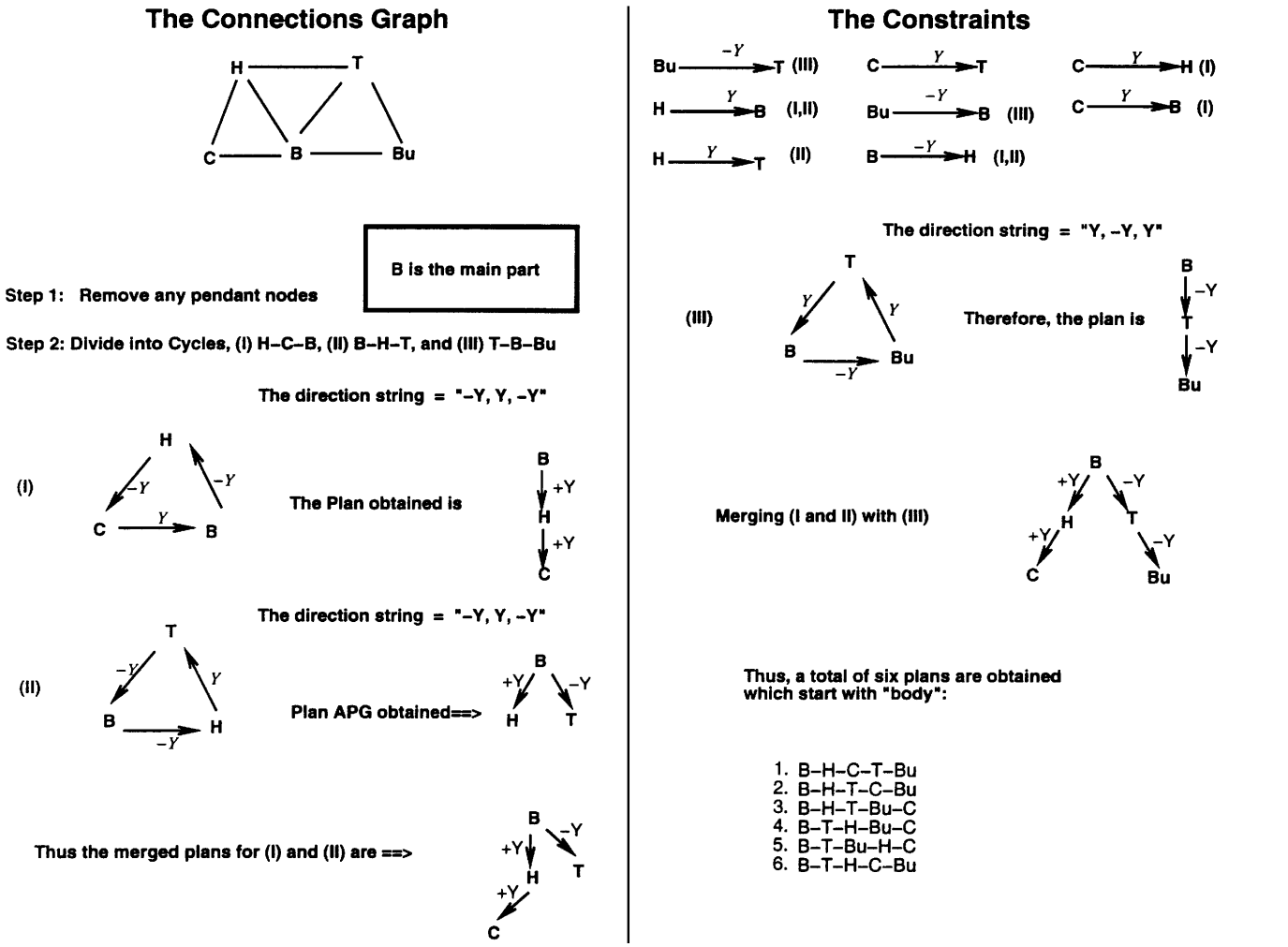


Fig. 8. Merging of plans.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

more than one assembly.

The *COMPOSITER* mechanism does not require that the loop set be *unique* (i.e. for a given assembly, one always gets particular set of loops). However, generating a set of goals uniquely from an assembly is still important for the planner extensions to define goals for "macro" cases. To ensure uniqueness, the algorithm first extracts all the loops of size 3 and only then proceeds to loops of size 4. Since loops of size 3 and 4 are the most common ones in assemblies, they are stored in the case base. Without attempting to show this formally, it is a common observation in most assemblies that useful apertures and shapes are formed by only three or four parts interacting with each other. For loops of larger sizes (which are formed infrequently), the planner can generate plans individually.

C. Plan Retrieval. This phase involves the searching of the database to find a match for the loops provided to the RETRIEVER module. Before proceeding further, a description of the databases is provided.

(1) Loop Database: The loop database consists of loops that are identified by the strings of mating directions. There are different databases for loops of different sizes. Actual part names are not included in the description, though their location is implied in the direction strings. Each entry also contains the loop's family number. A family of loops consists of loops that are of the same assembly configuration but different in spatial orientation and/or direction of traversal of the loop. An example of an entry in the loop database is shown in Figure 6. The important consequence of grouping into a Family is that all the loops have identical plans that only have to be adjusted for their differences in orientations. Each loop is thus identified by four parameters or status values in addition to the family number. These are:

- *Reorient:* The Reorient value locates the axis in which the primary direction points. The primary direction is the first direction in the direction string.
- *Swivel:* The Swivel value indicates the value (in steps of 90°) by which the loop is rotated about the primary axis.
- *Reverse:* The Reverse status indicates whether the loop was traversed in a reverse direction relative to

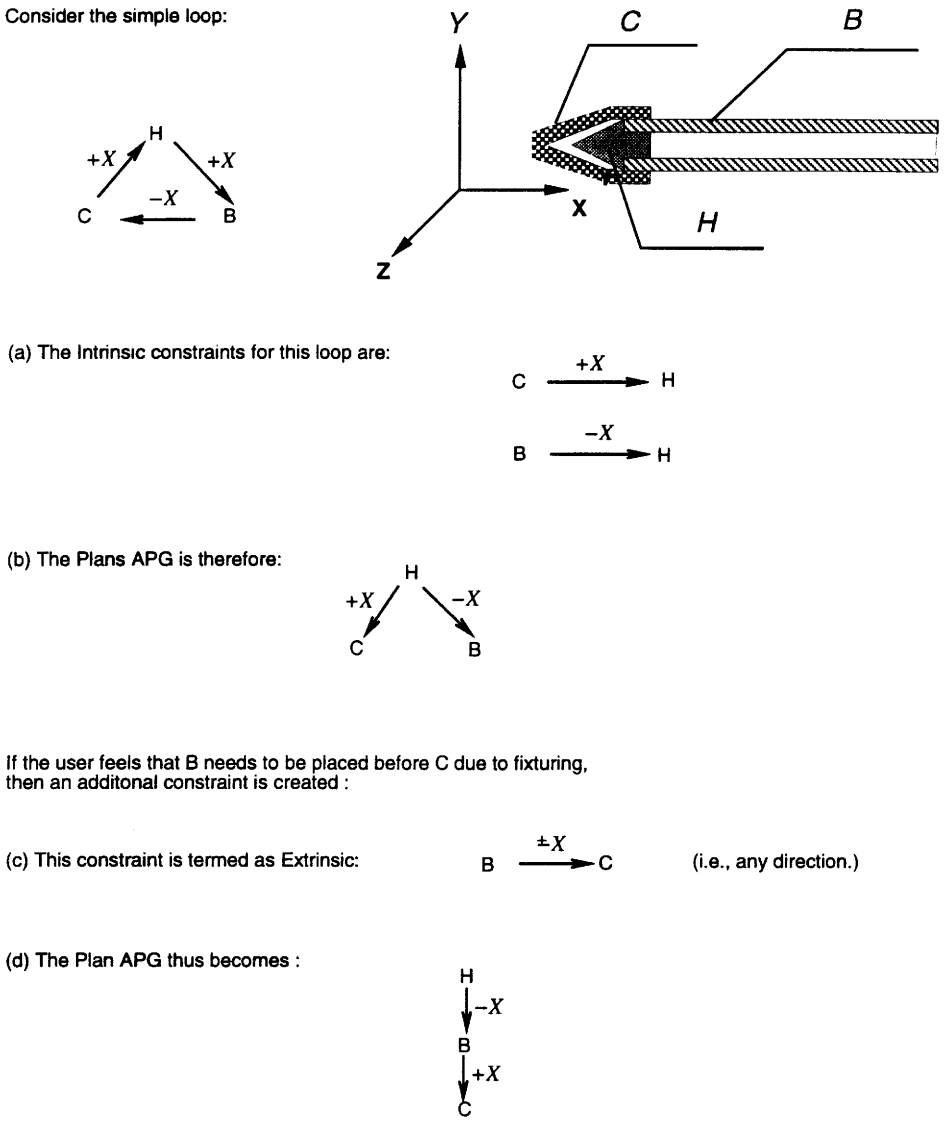


Fig. 9. Types of constraints.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

that of the *reference string* of the family. The reference string for a family is the string which has a zero value for all its status values.

- *Cyclic:* The Cyclic value denotes the fact that the loop has been rotated to the left by one unit. Thus, a size-3 loop can be cycled 3 times and a size-4 loop can be cycled 4 times.

(2) Plan Database: After retrieving the loop, the family number of this loop is used to index the Plan database which holds the plans for each family. The interesting fact about these families is that they are quite few in number*. Thus we have a very good opportunity for reuse of plans.

D. Modification of plans

The plans obtained from the database are expressed in terms of part names acting as placeholders for the actual parts. To make the plans useful, the dummy part names are converted to reflect the part names from the problem as well as change the directions labelled on the directed edges of the plan based on the reorientation. Both these functions are accomplished using the four status values.

The loop from the loop database is transformed using the status values so that its direction string now matches the direction string of the loop from the problem. At this stage, we can match the dummy part names with the part names from the problem. Each direction label of the plan is also subjected to the transformation routines using the status valus to obtain corresponding direction values. An example of this can be seen in Figure 7.

The reorient value indicates which axis toward which the assembly is oriented and the swivel value indicates the twist of the assembly about that axis. These affect the spatial positioning of the assembly. The reverse and cyclic values are used to identify the starting node and identify whether the loop has been traversed clockwise or counterclockwise. Thus, a set of plans that can be applied to the current situation is obtained and returned to the planner.

At the start of planning, the user is asked to provide the files which contain the assembly problem graphs. The user

* For this research, where we assume unidirectional mating, five families for loops of size 3 and 18 families for loops of size 4 have been identified as primary families. However, even among these families, there are only a few families which are physically feasible.

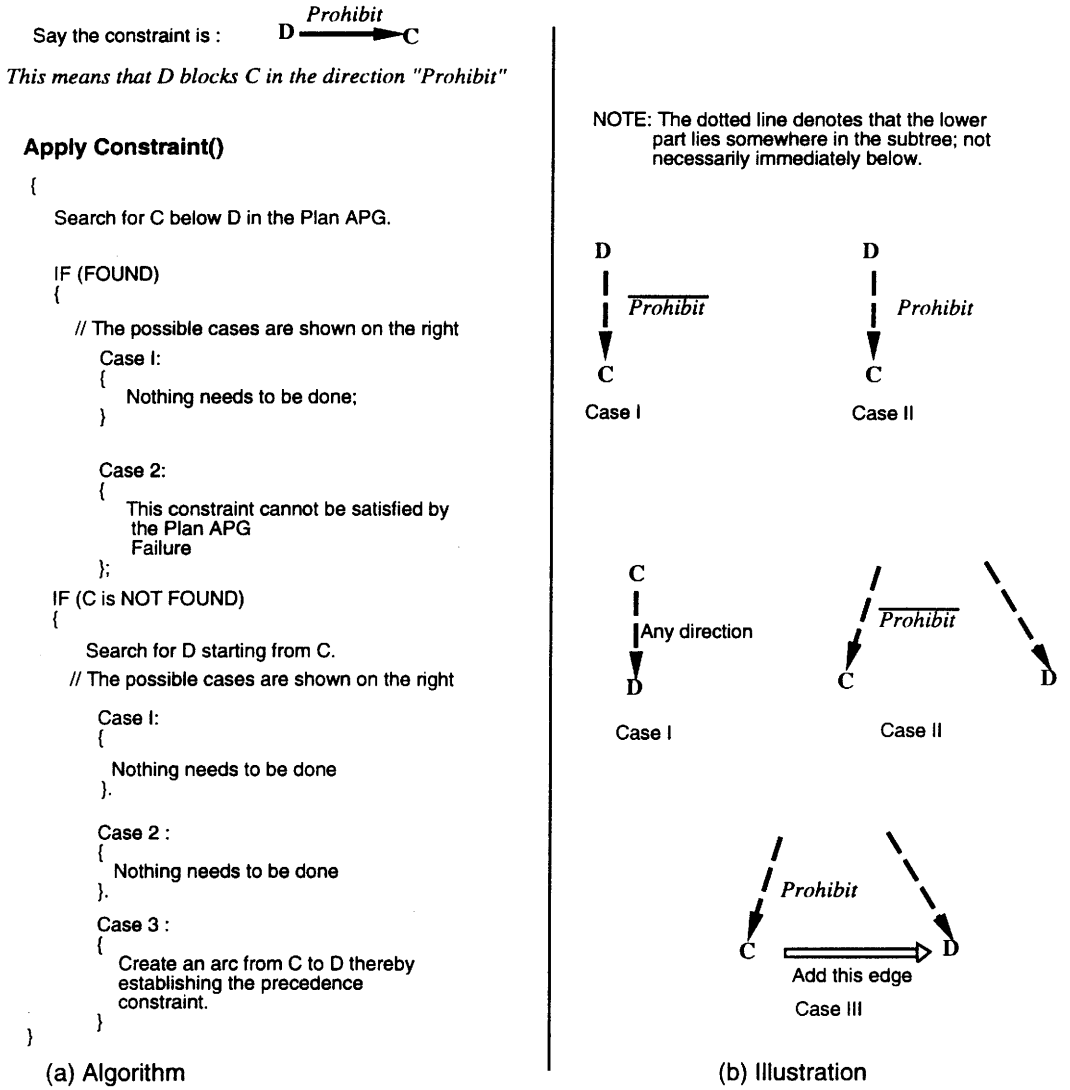


Fig. 10. Precedence algorithm.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

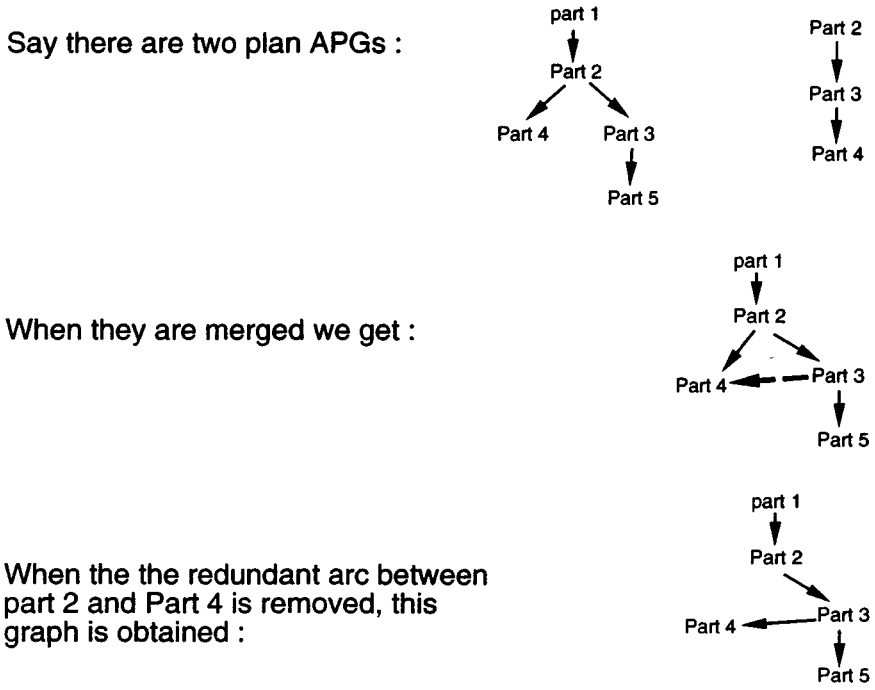


Fig. 11. Redundant precedence elimination.

is also asked to name the "*base part*" or "*main part*". This selection specifies the base part as the first part in the assembly (i.e. the part on which the other parts will be placed). This restriction helps to reduce the number of plans to be considered. Therefore, when the plans are retrieved, a check is applied to eliminate plans which do not have the base part as the first part in the assembly. Of course, for loops that do not contain the base part, this restriction is not meaningful. The base part restriction results in a different rule for loops having a part or parts which also form other loops inlcuding the base part. This rule applies to *plan graphs* (i.e. *APG*s) of such loops — a part which also forms loops with the base part can be the root node (or first part). This rule ensures that when the plans for loops without the base part are merged with the plans for loops containing the base part, the requirement that the base part be the root node will be satisfied for the merged *APG*.

E. Merging

When the plans for two loops are to be merged, plans for the respective loops are fused into one plan by matching corresponding nodes in the respective *APG*s. While doing so, the constraints implied in both graphs must be satisfied. In order to avoid attempts at merging plans that ae incompatible, a merge check is performed to evaluate the feasibility of the merge. If the initial set of constraints given were consistent (and a correct base part is chosen), then the intermediate *APG*s are always mergeable. (If none of the plans can be merged, this may indicate a wrong choice of base part).

The basic condition to successfully and correctly merge two plans is that the part ordering implicit in both the plans is consistent on the parts common to them. That is to say, if a part *B*, is *below* another part *A*, in an *APG* for the first loop, then *B strictly* cannot be *above A* in the second plan. The second condition is that the mating directions labelling the edges of the *APG* are consistent when merged. For example, if in one *APG* a part has a certain mating direction and in another *APG* it has another mating direction, then those two graphs cannot be merged. On the other hand, if the part can be placed using either of two mating directions when in one *APG* and it uses only one of those directions in another *APG*, then in the merged graph it can only use the *one common direction*. During the merge check, only the first of the conditions is checked. It is more convenient to ensure the second condition concurrently with the actual merging. The algorithm for checking the conditions specified above, first determines the common elements in both the graphs. For each common part, say PARENT, it finds the other common parts which are in its subtree (i.e. below it) in the *first* graph. They are its SUCCESSORS. The *second* graph is traversed starting at each of these SUCCESSOR nodes to search for this PARENT part in the subtree. If the PARENT part is found, a violation occurs and the plans cannot be merged. With respect to two merged plans referred to above, one of the merged plans is the plan composed of all the loops merged until now, and the other is the plan for a single loop that is to be merged with the remainder of the loops.

Once the check is performed the merging process itself is quite simple. In order to merge the two graphs, the incoming (i.e. new) plan is traversed in a depth first search (DFS) order. If a part on the incoming plan is a part not found in the existing plan, then a new entry is created to accommodate the new part. For parts common to the two plans, it is first checked to determine whether the mating directions of that part are consistent. This is the second condition for mergeability mentioned above. If this condition is satisfied, the part is located in the existing plan and its set of predecessors and set of successor parts is modified to include the information from the incoming plan. It is thus

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

ensured that the information in both graphs is systematically fused into one consistent plan. The merging process is demonsrated for the pen example in Figure 8. By iterating on all the plans in the existing plan set as well as the set of plans for the incoming loop, we are able to produce a set of plans that represent the feasible combinations.

F. Post-Processing

At the post-processing stage, user-defined and geometric precedences constraints are applied to plans. It should be noted however that the "intrinsic" constraints of the problem are automatically implied in the plans even at this stage. Therefore, they are redundantly applied at the next stage. The "intrinsic" constraints are a property of the loop's graph structure and will occur in any physical configuration containing the particular loop. The "extrinsic" constraints are the other constraints and may be due to the actual shape of the parts or some user imposed condition. Examples of "intrinsic" and "extrinsic" constraints are shown in Figure 9. The extrinsic constraints cannot be derived from an analysis of the loop alone. They can be derived only after including more details about the assembly. In our general view, constraints are also quite similar to plans. Constraints are in effect a "conditional plan" which also must be successfully merged into the existing plan for the assembly to be feasible. If the mating direction of a merged part, *C*, is the blocked direction with respect to a particular blocking part,

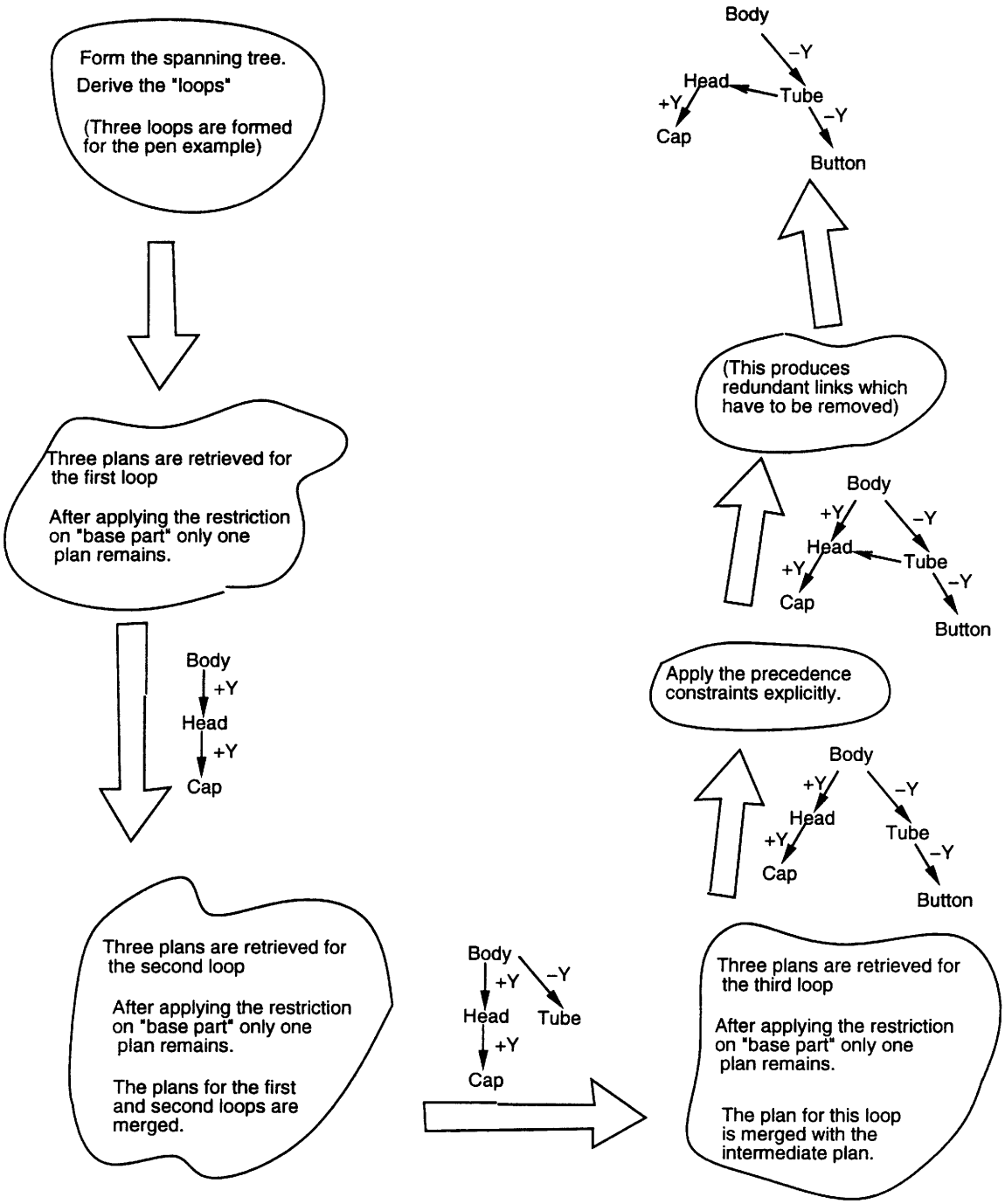


Fig. 12. Sequence of planning steps.

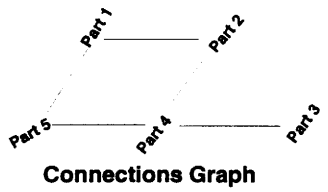
https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

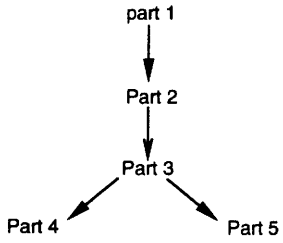
D, then *D* can be placed in assembly only after *C*. The different possible cases and the actions to be taken to handle those cases are shown in Figure 10.

An important function required after merging of plans as well as applying precedences is to "clean up" the plan graph. This procedure removes all the transitive links in the graph. The plan graph is not only directed and acyclic but must also be "non-redundant." By non-redundant, we mean that the precedence implied in the graph between two parts should only be derivable in one way. This is best illustrated by Figure 11. In such situations, the direct arc from the predecessor to the successor should be removed since the longer path between the two already implies that precedence. This direct arc must be removed to avoid extraction of a wrong plan while traversing the *APG*. For example, in Figure 11, if redundant arc between part2 and part4 is not removed a wrong plan (i.e. part1-part2-part4-part3-part5) may be extracted. This algorithm is a modification of the shortest path algorithm where a decision function maintains the maximum value of the distance between two points instead of the minimum distance. This algorithm calculates the longest distance between two nodes and compares it to the direct distance between the two nodes. If two nodes are directly connected in their adjacency graph and also have a maximum distance greater than one, then the short edge is removed. The complete sequence for the Pen example including the intermediate plans are shown in Figure 12.

V. IMPLEMENTATION ISSUES

The system was developed predominantly in C. For the different data structures, the system uses class libraries written in C++. The executables exist for both Sun (Solaris 2.2) and SGI Indigo (IRIX 4.0.5F) environments. The compilers used were Gnu g++ on the Suns and CC on the SGI.





(a) Adjacency Matrix of Unidirected Graph.

	P5	P1	P2	P4	P3
P5	0	1	0	1	0
P1	1	0	0	1	0
P2	0	1	0	1	1
P4	1	1	1	0	1
P3	0	0	1	1	0

(b) Adjacency Matrix of Directed Graph.

	P1	P2	P3	P4	P5
P1	0	1	0	0	0
P2	0	0	1	0	0
P3	0	0	0	1	1
P4	0	0	0	0	0
P5	0	0	0	0	0

Plan APG

(c) Adjacency List representation of Plan APG

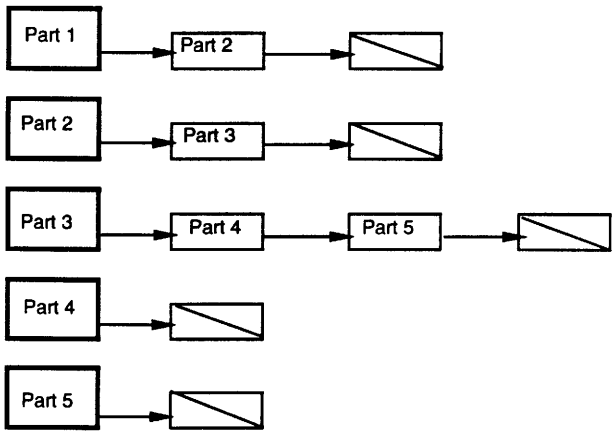


Fig. 13. Representation of graphs.

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

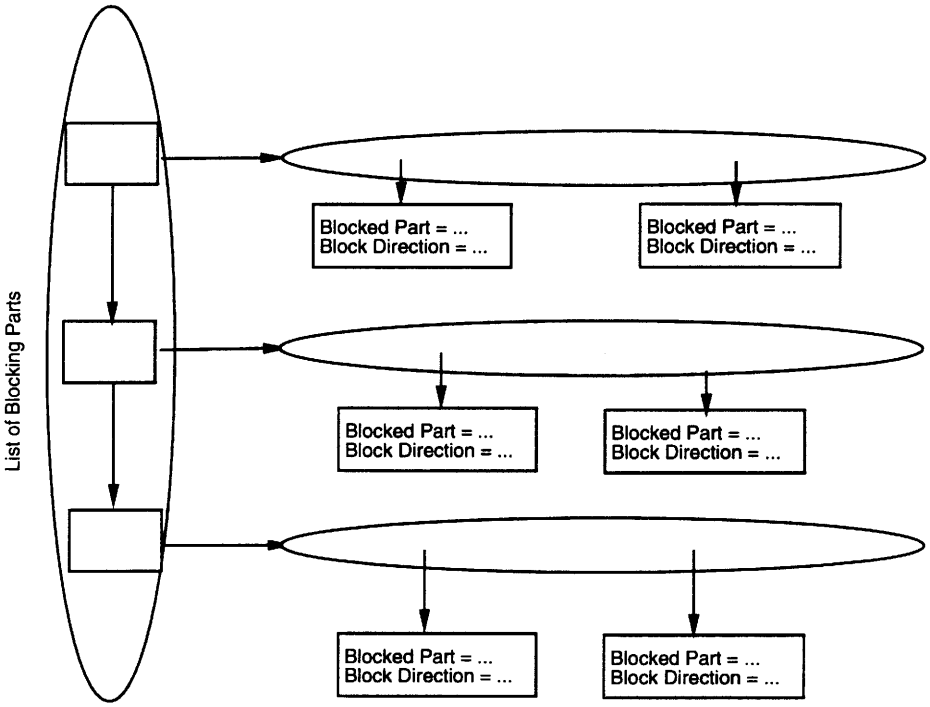


Fig. 14. Precedence constraints representation.

A. Data structures

Since the major tasks in this planner involve the manipulation of graphs, the representation of graphs is important. Both the directed and undirected graphs are used. The *CG* and a spanning tree are undirected graphs. The *MDG* and the plan graphs are directed graphs. However, the plan graphs are also acyclic. The assembly graphs *CG*, *MDG*, and *OFG* could be generated automatically in the future from geometric reasoning about part a CAD data.

The input graphs and the plan graphs stored for each family are stored in the adjacency matrix representation. This representation is explained by Figure 13. The *CG* is a symmetric matrix of ones and zeros. The matrix representing the *MDG* has direction strings in its elements. The directions are complementary about the diagonal. When checking mergeability as well as merging plans and applying precedences, it is necessary to traverse the graph from the top down systematically. The graph is traversed using the DFS algorithm. To support this algorithm, it is convenient to use an adjacency list representation of the plan graphs. The matrix representation is easier to use when the number of nodes in the graph is known. However, the adjacency list representation is much more flexible with respect to changing the number of graph nodes or representing graphs of unknown size. Other important structures are the "loops" which represent the search goals and the precedence constraints. Each "loop" is stored as a list of structures showing the source part, the destination part and the direction of mating. The precedence constraints are stored as a list of lists (see Figure 14).

B. Databases

The Loop database contains all the different direction strings that could be generated by a permutation of directions in all possible dimensions. These are generated off-line for loops of both sizes 3 and 4. As explained before, these are the most common loops and therefore it is useful to store them for case retrieval. The status values that lead to loop creation from the basic loop of each family are also stored. The effect of the different transformations on a direction string are shown in Figure 15. The loops are loaded into memory at run-time. The Plan database is also provided off-line.

C. Computational complexity

The existing implementation of the APE does not necessarily use the best algorithms in all of its stages. We present an estimate of the complexity of APE system assuming a typical situation where the algorithms with relatively better complexity are used. Assume there are n number of nodes, e number of edges, and L number of total loops. Also assume that w is an average number of nodes per loop (here w is either 3 or 4), then the complexity (C) can be estimated as:

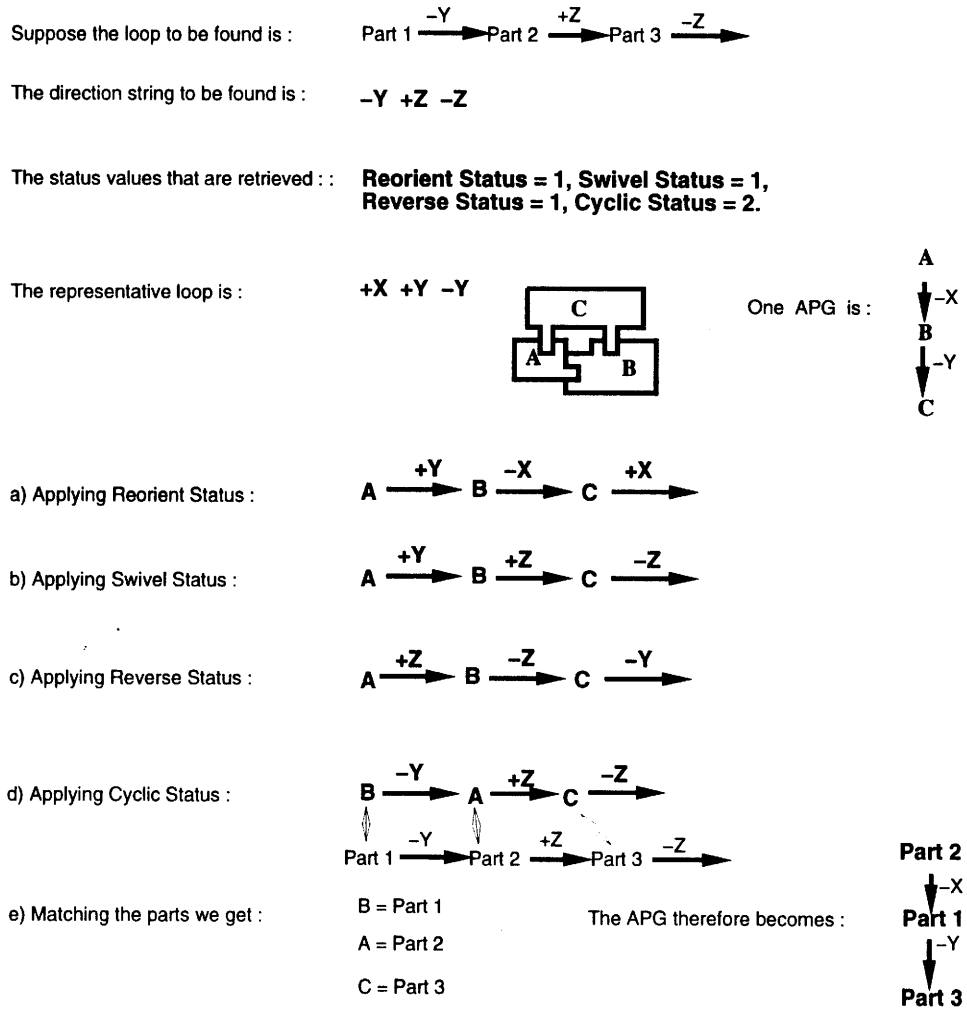
$$C = O(e^2) + O(L) + O(L) + O(Lw^2) + O(w^2)$$

The first term in the above equation is the complexity of the EVALUATOR to produce L loops. This stage involves the DFS (whose complexity is $O(e)$) and finding of circuits (or loops) of size 3 and 4. The second through fifth terms represent the complexities of RETRIEVER, MODIFIER, COMPOSITOR, and POSTPROCESSOR stages, respectively.

VI. CONCLUSIONS

The APE system design approaches the assembly sequence planning problem by leveraging notion that although assembly problems are complex arrangements of parts, these combinations are formed by combinations of a relatively small number of constituent configurations. This

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press



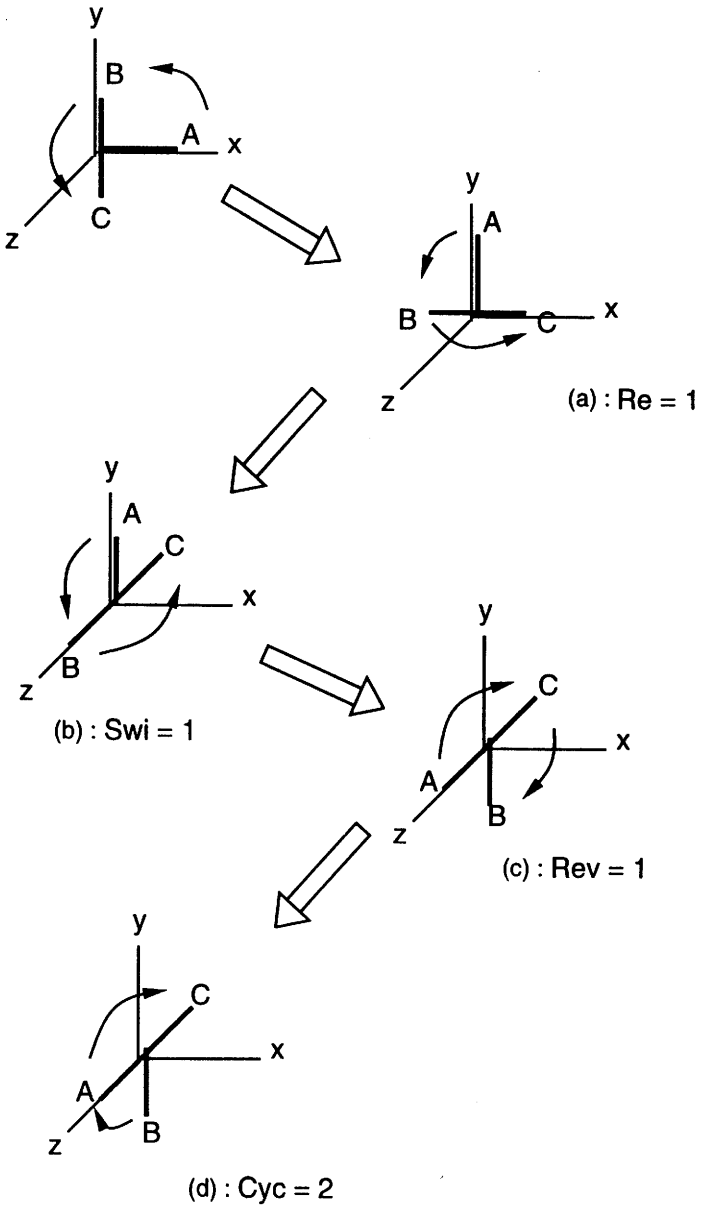


Fig. 15. Transformation of loop and APG.

realization assists in significantly reducing the complexity and amount of planning to determine the feasible sequences for the assembly. This planner is a "hybrid" planner since it reuses stored cases as well as create plans from the primitive connections between parts for situations that are not found in the case repository. For example, to creates plans for a loop of size five, each of the connections could be converted to a "micro-plan" that consists of just two parts and the direction in which the first mates with the second. From these micro-plans, the planner can use the same methods employed to merge larger plans. This capability allows planning for such infrequent cases. As a resulting finding, it is interesting that the number of possible assembly configurations is so small. When unidirectional mating is assumed, there were only five families for loops of size three and only eighteen families for loops of size four in the assemblies considered.

APE is the one of few planners that retrieves stored assembly cases and combines them in a systematic way to form a complete set of assembly plans. Idetification of similarity eases the difficult task of assembly planning by enabling plan reuse. APE systematically develops a number of feasible plans using the precedence constraints imposed on the assembly. This opens the door for applying various selection criteria on the plans such as least number of reorientations of a subassembly or least amount of movement required by an assembly robot. The compact plan representation employed by APE allows precedence constraints to be easily applied on all the plans. Constraints can be applied at any stage of the planning process or during adaptation of stored cases. APE plans efficiently by generating only feasible plans at each stage. APE is thus able to avoid the task of pruning large numbers of infeasible plans at the end of planning. Addition of a new constraint does not require complete replanning.

APE and CORINTH show that case-based planning is an effective planning method for assembly sequences. The potential challenges are to reduce the extent of training needed before a case-based planner becomes fully powerful, and the evaluation of whether this approach can consistently produce results superior to those planners using a fast "first-principles" approach.

References

1. A.C. Lin and T.C. Chang, "An integrated approach to automated assembly planning for three-dimensional mechanical products" *Int. J. Production Research* **31**, No. 5, 1201–1227 (1993).
2. A. Delchambre, *Computer Aided Assembly Planning* (Chapman and Hall Publishers, London, UK, 1992).
3. A.C. Sanderson, L. Homem deMello and H. Zhang, "Assembly sequence planning" *AI Magazine*, Spring, 62–81 (1990).
4. A.C. Sanderson and L. Homem deMello, "A correct and complete algorithm for generation of mechanical assembly sequences" *IEEE Transactions on Robotics and Automation* **7**, No. 2, 228–240 (1991).
5. J.M. Hernioud and A. Bourjault, "Computer aided assembly process planning" *Proceedings of Institution of Mechanical Engineers* **206**, 61–66 (1992).
6. E. Arai and K. Iwata, "CAD system with product assembly/disassembly planning function" *Robotics and Computer Integrated Manufacturing* **10**, No. 1/2, 41–48 (1993).
7. S. Lee, "Backward assembly planning with assembly cost analysis" Proceedings of the *IEEE International Conference on Robotics and Automation* (1992) pp. 2382–2391.
8. T.L. De Fazio and D.E. Whitney, "Simplified generation of all mechanical assembly sequences." *IEEE Journal of Robotics and Automation* 705–708 (1987).
9. D.Y. Cho and H.S. Cho, "Inference on robotic assembly precedence constraints using a part contact level graph" *Robotica* **11**, Pt. 2, 173–183 (1993).
10. R.S. Mattikalli and P.K. Khosla, "Motion constraints from contact geometry: representation and analysis" *Proceedings of the IEEE International Conference on Robotics and Automation* (1992) pp. 2178–2185.
11. A.Swaminathan, "APE: an experience-based assembly sequence planner for mechanical assemblies" *M.S. Thesis*, (The University of Texas at Austin, 1994).
12. C. Foundyller, "The right to assemble" *Computer-Aided Engineering*, (1994) **13**, 54 (1994).
13. M. Santochi and G. Dini, "Computer aided planning of assembly operations: the selection of assembly sequences" *Robotics and Computer Integrated Manufacturing* **9**, No. 6, 439–446 (1992).
14. R. Ananthan, "A feature based approach for representing and reasoning about mechanical assemblies" *M.S. Thesis* (The University of Texas at Austin, 1992).
15. J.J. Shah and R. Tadepalli, "Feature based assembly modeling" *Computers in Engineering* ASME, **1**, 253–260 (1992).
16. P.K. Venuvinod, "Automatic analysis of 3-D polyhedral assembly directions and sequences" *J. Manufacturing Systems* **12**, No. 3, 246–252 (1993).
17. J. Wolter, S. Chakraborty and J. Tsao, "Mating constraint languages for assembly sequence planning' *Proceedings of the IEEE International Conference on Robotics and Automation* (1992), pp. 2367–2374.
18. F. Akagi, H. Osaki and S. Kikuchi. "The method of analysis of assembly work based on the fastener method" *Bulletin of the JSME* **23**, No. 184, 1670–1675 (1980).
19. H. Sekiguchi and T. Kojima. "Study and automatic determination of assembly sequences" *Annals of the CIRP* **32**(1), 371–374 (1983).
20. L. Homem deMello and A.C. Sanderson, "AND/OR graph representation of assembly plans" *IEEE Transactions on Robotics and Automation* **6**, No. 2, 188–199 (1990).
21. R. Wilson and A. Schweikard, "Assembling polyhedra with single translations" *Proceedings of the IEEE International Conference on Robotics and Automation* (1992), pp. 2392–2397.
22. J. Browne, K. Tierney and M. Walsh, "A two-stage assembly process planning tool for robot-based flexible assembly systems" *Int. J. Production Research* **29**, No. 2, 247–266 (1991).
23. T. Cao and A.C. Sanderson, "Task sequence planning in a robot workcell using AND/OR nets" *IEEE International Symposium on Intelligent Control* (1991), pp. 239–244.
24. K.C. Riesbeck and R.C. Schank, *Inside Case-based Reasoning* (Lawrence Erlbaum Association Publishers, Hillsdale, NJ, 1989).
25. I. Katsushi and T. Suehiro, "Towards an assembly plan from observation" *Proceedings of the IEEE International Conference on Robotics and Automation* (1992), pp. 2171–2177.
26. K.S. Barber and G.J. Agin, "Analysis of human communication during assembly tasks" *CMU-RI-TR-86-13* (The Robotics Institute, Carnegie Mellon University, 1986).
27. J.L. Kolodner, *Retrieval and Organizational Strategies in Conceptual Memory* (Lawrence Erlbaum Associates, Hillsdale, N.J., 1984).
28. M. Lebowitz, "Generalization and memory in an integrated understanding system" *Research Report 186*, (Yale University, 1980).
29. W.M. Bain, "Case-based reasoning: a computer model of subjective assessment" *Ph.D. Thesis* (Yale University, 1986).
30. K.J. Hammond, *Case-Based Planning: Viewing Planning as a*

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press

Memory Task (Academic Press Inc., Boston, MA, 1989).
31. P. Pu and L. Purvis, "Assembly planning using case adaptation methods" *Proceedings of IEEE International Conference on Robotics and Automation* (1995), pp. 982–987.
32. S. Chakraborty and J. Wolter, "A hierarchical approach to assembly planning" *Proceedings of IEEE International Conference on Robotics and Automation* (1994), pp. 258–263.
33. P. Pu "An assembly sequence generation algorithm using case-based techniques" *Proceedings of the IEEE International Conference on Robotics and Automation* (1992), pp. 2425–2430.
34. N. Deo, *Graph Theory With Applications to Engineering and Computer Science* (Prentice Hall Inc., N.J., 1974).

https://doi.org/10.1017/S0263574798000356 Published online by Cambridge University Press