AND/OR Graph Representation of Assembly Plans

LUIZ S. HOMEM DE MELLO, MEMBER, IEEE, AND ARTHUR C. SANDERSON, SENIOR MEMBER, IEEE

Abstract—**A compact representation of all possible assembly plans of a given product using AND/OR graphs is presented. Such a representation forms the basis for efficient planning algorithms that enable an increase in assembly system flexibility by allowing an intelligent robot to pick a course of action according to instantaneous conditions. The AND/OR graph is equivalent to a state transition graph but requires fewer nodes and simplifies the search for feasible plans. Three applications are discussed: the preselection of the best assembly plan, the recovery from execution errors, and the opportunistic scheduling of tasks. The same approach used to preselect the best assembly plan can be used to preselect the best disassembly plan. A change in the definition of goal nodes allows the preselection of the best disassembly plan that will release a given part for maintenance or repair tasks. An example of an assembly with four parts illustrates the use of the AND/OR graph representation to find the best assembly plan based on the weighing of operations according to complexity of manipulation and stability of subassemblies. In practice, a generic search algorithm, such as the AO*, may be used to find this plan. A hypothetical error situation in the assembly of the four-part assembly is discussed to show how a bottom-up search of the AND/OR graph leads to an efficient recovery. The scheduling efficiency using this representation is compared with fixed sequence and precedence graph representations. The AND/OR graph consistently reduces the average number of operations.**

I. INTRODUCTION

ROBOTIC assembly systems have the flexibility to handle a wide range of styles and products, to assemble the same product in different ways, and to recover from errors or other unexpected events that cause the execution of the assembly to deviate from the preplanned course of action. Even with flexibility of the mechanical hardware, current robotic assembly systems are limited due to the inadequate data structure for the representation of task plans. Ordered lists of actions that have been used in early robot systems [5], [6] outside the manufacturing context do not permit flexibility in task execution. Triangle tables [7] have been used for the representation of plans, and they improve the ability to recover from errors but only within one fixed sequence. A more significant improvement was the use of precedence diagrams [9], [16] for the representation of plans, but that technique has limitations also, and in most cases, it allows only a small amount of flexibility.

Manuscript received January 11, 1988; revised November 15, 1989. This work was supported in part by Conselho Nacional de Desenvolvimento Cientifico e Tecnologico (Brazil); the Jet Propulsion Laboratory, California Institute of Technology; the Robotics Institute, Carnegie-Mellon University; and the New York State Center for Advanced Technology in Automation and Robotics, Rensselaer Polytechnic Institute.

L. S. Homem de Mello was with the Department of Electrical and Computer Engineering, and The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA 15213-3890. He is now with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109-8099.

A. C. Sanderson is with the Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY 12180-3590.

IEEE Log Number 9034533.

This paper presents a compact representation for the set of all possible assembly plans for a given product using AND/OR graphs. Such a representation enables an increase in assembly flexibility by allowing an intelligent robot to pick the more convenient course of action according to instantaneous conditions. Section II discusses background to assembly planning. Section III presents the AND/OR graph representation, and Section IV describes its use for the assembly of a simple product. Three applications are discussed. Section V shows how the selection of the best assembly plans can be implemented as a top-down search of the AND/OR graph; it also shows how the same approach can be used to select the best disassembly plan to release a given part for repair. Section VI shows the use of a bottom-up search of the AND/OR graph for recovery from execution errors. Section VII illustrates the use of the representation in opportunistic scheduling. Section VIII summarizes the contribution of the paper and points to further research.

II. PLANNING FOR ROBOTIC ASSEMBLY

Planning has been an important research topic in artificial intelligence (AI), and the AI approach has dominated much of the research in robot task planning using domain-independent methods. The central idea of domain-independent planning is to have one general-purpose inference engine, which can be used for any domain by describing the initial state, the goal, and the operators in a logic formalism.

One of the earliest domain-independent planners was based on Green's question answering system [10]. The operators were represented as implications, and a situation variable was included in the well-formed formulas that described a given state of a robot environment. A sequence of operations that transforms an initial state into a goal state was constructed by Green's question answering system in order to answer the question of the situation in which a given goal fact was true.

That approach has the advantage of generality, but it has serious disadvantages. One disadvantage, known as the *combinatorial explosion*, is the rapid growth of the search trees, which hinders the solution of nontrivial problems. Another disadvantage, known as the *halting problem*, is that the search is not guaranteed to terminate unless there actually exists a sequence of operators that convert the initial state into the goal state. If no plan is generated after a long search, one does not know whether the task is indeed unfeasible or the shortest plan is longer than those explored. These disadvantages are further aggravated by the *frame problem*, which is the need to express, in *frame axioms*, all the relations that are not affected by the application of an operator.

Most of the AI research that followed Green's work [6],

[7], [15]–[21] focused on solving the *combinatorial explosion*, *halting*, and *frame* problems. The emphasis was on developing powerful control schemes to guide the search. The representations of plans were typically based on ordered lists of preprogrammed primitive actions. More recent systems, such as NOAH [16] and SIPE [19], represented plans as partially ordered sequences of actions.

The priority in the AI research has been to develop efficient general-purpose procedures that can find at least one plan in a wide variety of situations, rather than procedures that eventually find the most efficient plan in a more restricted situation. In applications where plans are executed one time only, inefficiencies in the plan may not be of major concern. If plans must be generated in real time, high speed in plan generation is often preferable to optimal plans. In an assembly manufacturing environment, plans may be executed many times and therefore must be efficient to keep wasted time and resources to a minimum. Flexible manufacturing situations may require tradeoffs in these priorities for planning efficiency.

One additional difficulty in planning for robotic assembly is the continuous change in production conditions. For example, parts may come in random order. In most robotic assembly applications, there is no single plan that is efficient in every possible situation. To decide which plan is the best may require information that is only available at execution time, but producing plans in real time may degrade the robot operation due to the long computing time it usually takes to generate a plan. The choice between planning ahead of time (off line) and planning in real time (on line) is difficult; the former may lead to inefficient plans, whereas the latter may cause a degradation in the robot operation.

Fox and Kempf [9] addressed the need to act opportunistically as opposed to always following a preprogrammed fixed order of operations. They suggested that plans generated off line be a set of operations with minimal ordering constraints. Such a plan was represented by a precedence diagram and would actually encompass several possible sequences of operations that would perform the task of assembling a given product. In real time, depending on the conditions, the intelligent robot would pick the most appropriate sequence. Planning, in this sense, should yield all possible sequences of operations that can be used to assemble a product. That information is used by the workstation control [1], which in real time selects one of those sequences and assigns the resources to perform each operation.

The problem with the precedence diagram formalism is that for most products, no single partial order can encompass every possible assembly sequence. The assembly of the simple product shown in exploded view in Fig. 1, for example, may be completed by following one of the ten different sequences of operations that are represented graphically in Fig. 2. It is possible to combine some sequences into one partial order using precedence diagrams. Fig. 3 shows three possible ways to combine two of the first four sequences in Fig. 2; the only restriction is that the insertion of the stick cannot be the last operation. It is possible to combine three of those four sequences into one partial order by using a dummy operation, but it is not possible to combine the four sequences into one partial order, nor is it possible to combine any of those sequences with the other six sequences in Fig. 2.

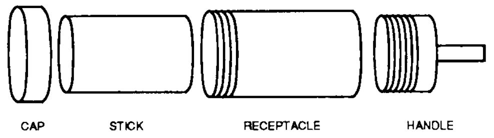


Fig. 1. Simple product.

A closer look at the partial ordering representation of plans, in light of the above assembly example, shows another deficiency of that solution. Two distinct feasible sequences, A–B–C and B–A–C, for example, do not differ simply by the sequences of the operations. Inserting the stick first is not the same operation as inserting it after the receptacle and the cap have been screwed together. The latter operation is probably easier to execute. Similarly, screwing the receptable and the handle with the stick inside is probably easier to do if the receptacle and the cap are screwed than otherwise. The partial ordering approach does not capture this subtle difference.

More recently, Fox [8] has introduced a representation for serial robotic tasks based on logic combinations of precedence relationships. Each precedence relationship relates assembly operations or logic combinations of assembly operations. Although this representation encompasses all assembly plans, it has the restrictions that the set of operations must be fixed (only their order can change) and that operations are executed serially, that is, one at a time. These restrictions also occur in petri-net representations such as those used by Bourjault *et al.* [2].

The next section introduces the AND/OR graph representation of assembly plans that does not have the restrictions that the set of operations be fixed and that operations be executed one at a time. Furthermore, the AND/OR graph representation of assembly plans combines all possible assembly sequences. It also shows explicitly when assembly operations can be executed in parallel and do not have any time dependence.

III. AND/OR Graph Representation of Assembly Plans

Assembly plans are ordered sequences of operations that transform one configuration of parts into another, starting with all parts disconnected from each other and ending with all parts properly joined to form the desired assembly. The state of the assembly process is given by the configuration of parts, and an assembly plan can also be seen as a sequence of states. A configuration of parts can be described by the fixed (i.e., determined) relative positions between pairs of parts. The initial configuration of an assembly plan is the empty set since there is no fixed relative position between any pair of parts at the beginning of the assembly process. Alternatively, a configuration of parts can be described by the subsets of parts that have fixed relative position among themselves. Each subset of parts in a configuration within an assembly plan corresponds to a subassembly that has already been assembled. The initial configuration is described by as many subsets as the number of parts in the assembly. Each assembly operation corresponds to one transition between two configurations, and an assembly plan is a sequence of these operations. In addition, each as-

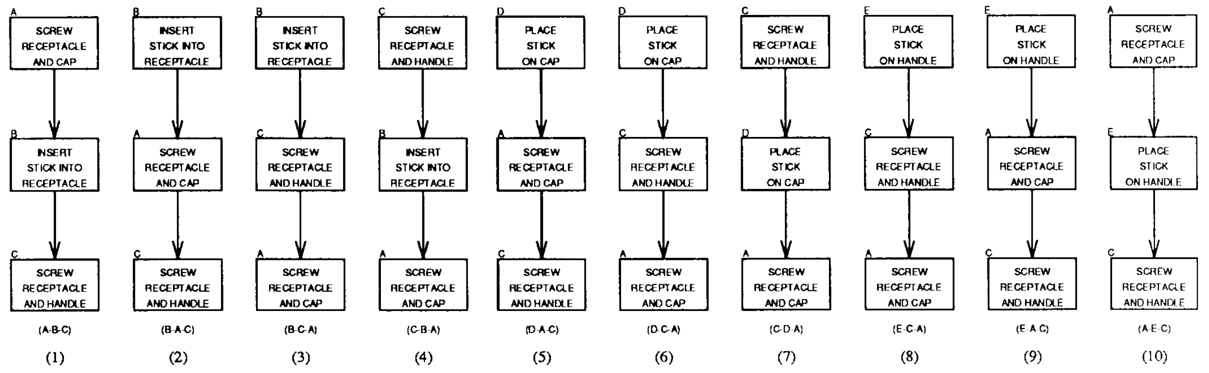

A SCREW RECEPTACLE AND CAP
B INSERT STICK INTO RECEPTACLE
C SCREW RECEPTACLE AND HANDLE
(A-B-C)
(1)
B INSERT STICK INTO RECEPTACLE
A SCREW RECEPTACLE AND CAP
C SCREW RECEPTACLE AND HANDLE
(B-A-C)
(2)
B INSERT STICK INTO RECEPTACLE
C SCREW RECEPTACLE AND HANDLE
A SCREW RECEPTACLE AND CAP
(B-C-A)
(3)
C SCREW RECEPTACLE AND HANDLE
B INSERT STICK INTO RECEPTACLE
A SCREW RECEPTACLE AND CAP
(C-B-A)
(4)
D PLACE STICK ON CAP
A SCREW RECEPTACLE AND CAP
C SCREW RECEPTACLE AND HANDLE
(D-A-C)
(5)
D PLACE STICK ON CAP
C SCREW RECEPTACLE AND HANDLE
A SCREW RECEPTACLE AND CAP
(D-C-A)
(6)
C SCREW RECEPTACLE AND HANDLE
D PLACE STICK ON CAP
A SCREW RECEPTACLE AND CAP
(C-D-A)
(7)
E PLACE STICK ON HANDLE
C SCREW RECEPTACLE AND HANDLE
A SCREW RECEPTACLE AND CAP
(E-C-A)
(8)
E PLACE STICK ON HANDLE
A SCREW RECEPTACLE AND CAP
C SCREW RECEPTACLE AND HANDLE
(E-A-C)
(9)
A SCREW RECEPTACLE AND CAP
E PLACE STICK ON HANDLE
C SCREW RECEPTACLE AND HANDLE
(A-E-C)
(10)

Fig. 2. Possible sequences of operations to assemble product of Fig. 1.

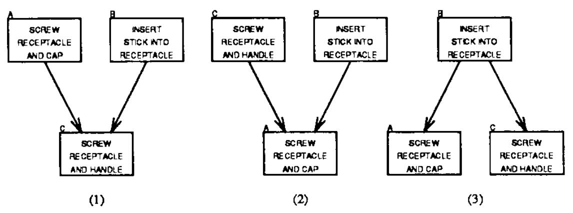

A SCREW RECEPTACLE AND CAP
B INSERT STICK INTO RECEPTACLE
C SCREW RECEPTACLE AND HANDLE
(1)
C SCREW RECEPTACLE AND HANDLE
B INSERT STICK INTO RECEPTACLE
A SCREW RECEPTACLE AND CAP
(2)
B INSERT STICK INTO RECEPTACLE
A SCREW RECEPTACLE AND CAP
C SCREW RECEPTACLE AND HANDLE
(3)

Fig. 3. Tree of the AND/OR graph.

sembly operation fixes the relative position between a number of pairs of parts.

This section describes a directed graph of assembly states that includes all feasible assembly plans for a given assembly. In addition, a hypergraph, or AND/OR graph, which provides an equivalent representation of all plans, is presented. The AND/OR graph reduces the number of nodes in the representation of all feasible plans and provides the basis for planning by tree search.

The following notation will be used in the discussion that follows: $\Pi(Y)$ is the set of all subsets of set Y, $\Delta(Y)$ is the set of all partitions of set Y, $|Y|$ is the cardinality of set Y, and $U(\Psi)$ is the union of all elements of set Ψ, which is a set of sets.

Any nonempty set of parts that are joined to form a stable unit is called an *assembly*. A component part is also an assembly. An assembly made up of N parts can be characterized by the set of parts $P = \{p_1, p_2, \cdots, p_N\}$ plus a coordinate transformation function $T: P \times P \to A$, where A is the set of four-dimensional homogeneous transformation matrices. The coordinate transformation function $T(p_i, p_j)$ characterizes the geometry of the assembly. If there are two different assemblies made up of the same set of parts, they will be characterized by two distinct coordinate transformation functions $T_1(p_i, p_j)$ and $T_2(p_i, p_j)$. Any subassembly of an assembly can be similarly characterized by a subset of parts plus the same coordinate transformation function $T(p_i, p_j)$ that characterizes the original assembly. For the discussion in this paper, a unique assembly geometry will be assumed, and the function $T(p_i, p_j)$ is included implicitly in the representations of assemblies and of assembly configurations.

Given an assembly P made up of N parts, $P = \{p_1, p_2, \cdots, p_N\}$, a configuration Θ of P is a set of subassemblies of P such that

- Every member of Θ characterizes a connected stable subassembly
- every part of P is in one of the elements of Θ
- there is no part of P that is in two elements of Θ.

The space of all possible configurations of parts of assembly P is a subset of $\Delta(P)$, which is the set of all partitions of P. An assembly plan for P can be seen as a sequence of configurations of P. The initial configuration $\Theta_I = \{\{p_1\}, \{p_2\}, \cdots, \{p_N\}\}$ is that configuration in which all parts are disconnected from each other. The final configuration $\Theta_F = \{\{p_1, p_2, \cdots, p_N\}\}$ is that configuration in which all parts are properly joined to form the desired assembly. Each transition between two configurations corresponds to joining a number of subassemblies to form one larger subassembly. An assembly plan must include physically feasible transitions only. Transitions are physically feasible if there exists an unobstructed path of relative motion that separates the two subassemblies, and no forces exist that restrict that motion. The development of algorithms to test for operations feasibility from a design description of the assembly is another phase of the current development and will not be described here.

Planning the assembly of P can therefore be seen as a path search in the directed graph $\langle C_P, E_P \rangle$. The set of nodes in this graph is

$$C_P = \{\Theta | [\Theta \in \Delta(P)] \wedge [\forall \theta (\theta \in \Theta) \Rightarrow \mathrm{csp}(\theta)]\}$$

where the connected-stable subassembly predicate is

$$csp : \Pi(P) \to \{\text{true}, \text{false}\}$$

with

$$csp(\theta) = \begin{cases} \text{true}, & \text{if } \theta \text{ characterizes a connected stable subassembly;} \\ \text{false}, & \text{otherwise.} \end{cases}$$

The set of edges is

$$E_P = \{(\Phi, \Gamma) | [(\Phi, \Gamma) \in C_P \times C_P] \wedge [|\Gamma - (\Phi \cap \Gamma)| = 1] \wedge [U(\Phi - (\Phi \cap \Gamma)) \in \Gamma - (\Phi \cap \Gamma)] \wedge [ftp(\Phi, \Gamma)]\}$$

where the feasible-transition predicate is

$$ftp : C_P \times C_P \rightarrow \{\text{true, false}\}$$

with

$$ftp(\Phi, \Gamma) = \begin{cases} \text{true,} & \text{if the transition from } \Phi \text{ to } \Gamma \text{ is a physically feasible assembly operation;} \\ \text{false,} & \text{otherwise.} \end{cases}$$

The nodes in this graph (i.e., the elements of C_P) are the possible states of the assembly process, that is, the possible configurations of P. C_P contains all the partitions of P whose elements correspond to connected stable subassemblies (i.e., they satisfy the connected-stable predicate). The edges in this graph (i.e., the elements of E_P) are the possible transitions between states of the assembly process. E_P contains pairs of elements of C_P such that

- There is exactly one element in the second configuration that is not in the first configuration.
- The union of the elements of the first configuration that are not in the second is exactly the element in the second configuration that is not in the first.
- The transition corresponds to a physically feasible assembly operation (i.e., they satisfy the feasible-transition predicate).

An assembly plan for P corresponds to a path in $\langle C_P, E_P \rangle$ starting from node Θ_I and ending at node Θ_F. A path in $\langle C_P, E_P \rangle$ from node n_I to node n_F is defined as a finite sequence of nodes,

$$\text{path }(n_I, n_F) = (n_1, n_2, \cdots, n_P)$$

satisfying

$$n_1 = n_I$$
$$n_p = n_F$$
$$\forall i \forall j \{[(1 \le i \le p) \wedge (1 \le j \le p) \wedge (i \ne j)] \Rightarrow (n_i \ne n_j)\}$$
$$\forall i \{(1 \le i \le p-1) \Rightarrow [(n_i, n_{i+1}) \in E_P]\}$$

Therefore, a path from n_I to n_F is a finite sequence of nodes that starts with n_I and ends with n_F such that there is no node that appears more than once, and every pair of successive nodes is an edge in E_P.

The path search in planning the assembly of P is usually more efficient going backward from the final configuration Θ_F to the initial configuration Θ_I. The advantage of the backward search stems from the fact that no backtracking is necessary. In the forward search, backtracking may be necessary because there are assembly operations that are physically feasible but do not lead to a solution. For example, for the simple product shown in Fig. 1, it is possible to start the assembly by attaching the cap to the receptacle. After this first operation is completed, it is physically feasible to attach the handle to the receptacle, but that renders the insertion of the stick unfeasible.

The problem of finding how to assemble a given assembly can be converted to an equivalent problem of finding how the same assembly can be *disassembled*. Since assembly operations are not necessarily reversible, the equivalence of the two problems requires that each disassembly operation be defined as the logical reverse of a feasible assembly operation, regardless of whether the reverse operation itself is feasible or not. For the purpose of this discussion, the expression *disassembly operation* is used to refer to the reverse of a feasible assembly operation.

The resulting backward search may be viewed as a decomposable production system in which the problem of disassembling one assembly is decomposed into distinct subproblems, each one being to disassemble one subassembly. Each decomposition must correspond to a disassembly operation. If solutions for the subproblems that result from the decomposition are found, then a solution for the original problem can be obtained by combining the solutions to the sub-problems and the operation used in the decomposition. This view of disassembly as a decomposable production system leads to an alternative representation of the problem using structures called AND/OR graphs [13] or *hypergraphs*.

In this approach the path search in $\langle C_P, E_P \rangle$ is redefined in terms of a tree search over the hypergraph $\langle S_P, H_P \rangle$. The set of nodes in this hypergraph is

$$S_P = \{\sigma | \sigma \in \Pi(P) \wedge csp(\sigma)\}$$

where csp is the connected-stable subassembly predicate defined previously. The set of hyperarcs is

$$H_P = \{(\sigma, \Theta) | [\sigma \in S_P] \wedge [\Theta \in C_\sigma] \wedge [fop(\sigma, \Theta)]\}$$

where the set of configurations of subassembly σ is

$$C_\sigma = \{\Theta | [\Theta \in \Delta(\sigma)] \wedge [\forall \theta (\theta \in \Theta) \Rightarrow csp(\theta)]\}$$

the feasible-operations predicate is

$$fop : S_P \times \sum \rightarrow \{\text{true, false}\}$$

with

$$fop(\sigma, \Theta) = \begin{cases} \text{true,} & \text{if it is physically feasible, in one assembly operation, to obtain } \sigma \text{ by joining all the elements of } \Theta \\ \text{false,} & \text{otherwise.} \end{cases}$$

and the set of configurations of subassemblies of P is

$$\Sigma = \{\Theta | \exists \sigma [\sigma \in S_P] \wedge [\Theta \in C_\sigma]\}.$$

The nodes in this hypergraph (i.e., the elements of S_P) are the subsets of P that correspond to connected-stable subassemblies (i.e., they satisfy the connected-stable predicate). The node corresponding to the set of all parts is referred to as the root node. The hyperarcs (i.e., the elements of H_P) are the physically feasible decompositions of subassemblies into smaller subassemblies. Each hyperarc is an ordered pair in which

- The first element is a node (i.e., an element of S_P) that corresponds to a connected stable subassembly.

- The second element is a set of nodes such that the set of their corresponding subassemblies is a configuration of the first element, where a configuration of a subassembly is defined the same way a configuration of the whole assembly was defined.
- There is one physically feasible assembly operation that joins all the subassemblies corresponding to the members of the second element in the ordered pair and yields the subassembly corresponding to the first element in the ordered pair (i.e., the ordered pair satisfies the feasible-operation predicate).

The hyperarcs in an AND/OR graph are sometimes referred to as k connectors, where k is an integer greater than zero, corresponding to the cardinality of the second element in the ordered pair. Typically $k = 2$ since most assembly operations join two subassemblies.

An assembly plan for P corresponds to a tree in the hypergraph $\langle S_P, H_P\rangle$ from node $\{p_1, p_2, \cdots, p_N\}$ to the set of nodes $\{\{p_1\}, \{p_2\}, \cdots, \{p_N\}\}$. A tree in $\langle S_P, H_P\rangle$ from node n_I to the set of nodes $\Omega = \{n_1, n_2, \cdots, n_L\}$ is defined recursively as

$$\text{tree } (n_I, \Omega) = \begin{cases} (n_I) & \text{if } n_I \in \Omega \\ (n_I, \text{ tree } (n_I, \Omega) \cdots \text{tree}(n_k, \Omega)) & \\ \quad \text{with } (n_I, \{n_1, \cdots, n_k\}) \in H_P & \\ \quad \text{otherwise.} & \end{cases}$$

Therefore, a tree from n_I to Ω is either n_I itself, if n_I is in Ω, or n_I plus the trees from each of n_I's successors through one hyperarc. The set Ω is referred to as the set of goal nodes, or tip nodes, of the tree. This definition assumes that the hypergraph contains no cycle, as is true in $\langle S_P, H_P\rangle$.

The construction of the AND/OR graph requires the ability to find all connected stable subassemblies and all physically feasible disassembly operations of a given assembly. This involves the computation of both the connected-stable predicate and the feasible-operation predicate. Although it is not addressed in this paper, the automatic construction of the AND/OR graph is an important topic of ongoing research [11]. That capability is particularly important in assessing the assembly effort corresponding to several different design solutions because minor differences in the design of a product may yield drastic changes in the assembly process.

The useful feature of the AND/OR graph representation for the assembly planning problem is that it encompasses all possible partial orderings of assembly operations with a reduced number of nodes. The reduction in the number of nodes varies from one assembly to another since the number of nodes in the graph $\langle C_P, E_P\rangle$ and in the hypergraph $\langle S_P, H_P\rangle$ depends on how the parts in the assembly are interconnected. Table I shows the maximum number of nodes in the graph and the hypergraph as a function of the number N of parts in the assembly. Two cases for how the parts may be interconnected are presented here as examples. One case is a strongly connected assembly in which every part is connected to every other part. The other case is a weakly connected assembly in which there are $N - 1$ interconnections between the N parts, with the ith interconnection connecting part p_i and part p_{i+1}.

TABLE I
NUMBERS OF NODES IN THE GRAPH OF ASSEMBLY STATES AND IN THE AND/OR GRAPH

	Strongly Connected Assemblies		Weakly Connected Assemblies	
N	Graph of Assembly States $\lvert C_P\rvert$	AND/OR Graph $\lvert S_P\rvert$	Graph of Assembly State $\lvert C_P\rvert$	AND/OR Graph $\lvert S_P\rvert$
1	1	1	1	1
2	2	3	2	3
3	5	7	4	6
4	15	15	8	10
5	52	31	16	15
6	203	63	32	21
7	877	127	64	28
8	4140	255	128	36
9	21 147	511	256	45
10	115 975	1023	512	55
11	678 570	2047	1024	66
12	4 213 597	4095	2048	78
13	27 644 437	8191	4096	91
14	190 899 322	16 383	8192	105
15	1 382 958 545	32 767	16 384	120

In both cases, it was assumed that every connected subassembly is stable and that every disassembly operation is feasible. Therefore, the number of nodes in Table I correspond to extreme cases and will be reduced for assemblies in which not all connected subassemblies are stable and not all disassembly operations are feasible.

For the strongly connected assemblies, every subset of parts is connected. Therefore, the number of nodes in the graph is

$$|C_P| = \text{partitions } (N) = \sum_{i=0}^{N-1} \binom{N-1}{i} \text{ partitions } (N - 1 - i)$$

with

$$\text{partitions}(0) = 1$$

which is the recursive formula for the number of partitions of a set of N elements [4]. The number of nodes in the hypergraph is

$$|S_P| = 2^N - 1$$

which is the number of nonempty subsets of a set of N elements.

For the weakly connected assemblies, the number of nodes in the graph is

$$|C_P| = 2^{N-1}$$

which is the number of subsets of interconnections, and the number of nodes in the hypergraph is

$$|S_P| = \sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

which is the number of connected subassemblies since there are N subassemblies with one part only, $N - 1$ subassemblies with two parts, $N-2$ subassemblies with three parts, and so on.

As shown in Table I, for assemblies of five or more parts, if every connected subassembly is stable and every disassembly operation is feasible, then the AND/OR graph has substantially fewer nodes than the directed graph of assembly states, and this advantage becomes greater as the number of parts increases. Table I also shows that the reduction in the number of nodes is greater for strongly connected assemblies.

The comparison between the number of nodes in the directed graph of assembly states $\langle C_P, E_P \rangle$ and in the AND/OR graph $\langle S_P, H_P \rangle$ also depends on the assembly plans. For example, if an eight-part assembly has only one assembly plan (i.e., only one assembly tree), then the number of nodes in the AND/OR graph is 15. If the assembly plan is such that every hyperarc points to a one-part subassembly, then there is only one assembly sequence, and the configuration graph has nine nodes, which is less than 15, but if the assembly plan is such that every hyperarc points to subassemblies with the same number of parts, then there are 80 assembly sequences, and the configuration graph has 26 nodes, which is more than 15.

In practice, for assemblies with large numbers of parts, each part is connected to only a few, typically two to four, other parts. Therefore, for assemblies with large numbers of parts, the numbers of nodes for weakly connected assemblies in Table I are more typical of real assemblies.

For assemblies with very large numbers of parts, the AND/OR graph may still be too large. In these cases, parts can be clustered hierarchically into subassemblies. By artificially reducing the numbers of parts, the size of the AND/OR graph will be reduced, but it will not include all the different ways in which the clusters of parts can be assembled. The maximum reduction in the size of the AND/OR graph occurs when the parts are clustered in a way that the whole assembly becomes weakly connected.

IV. A Simple Example

Fig. 4 shows the AND/OR graph for the product in Fig. 1. Each node in that graph is associated with a subset of parts that corresponds to a subassembly. To facilitate the exposition, both the nodes and the hyperarcs in Fig. 4 have identification numbers.

The root node in Fig. 4 (node 1) is associated with the set of parts that corresponds to the whole assembly. There are four hyperarcs leaving that node. Each of those four hyperarcs corresponds to one way the whole assembly can be disassembled, and each points to two nodes that are associated with sets of parts that describe the resulting subassemblies. Similarly, the other nodes in the graph have a leaving hyperarc for each possible way in which their corresponding subassembly can be disassembled.

Any connected stable subassembly that can be made up of the component parts appears only once in the AND/OR graph shown in Fig. 4, even when it may be the result of different disassembly operations. The subassembly of node 4, for example, may result from two different operations, which correspond to hyperarcs 5 and 10. Moreover, those two hyperarcs come from two distinct nodes.

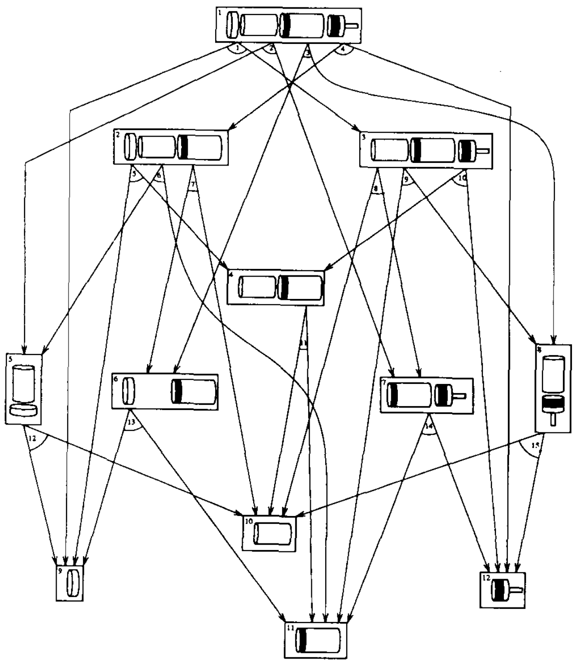


Fig. 4. AND/OR graph for simple product of Fig. 1.

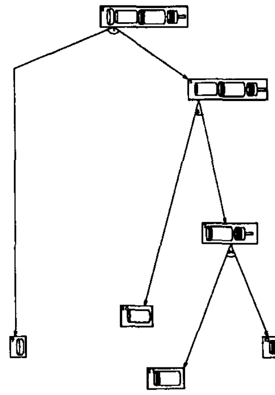


Fig. 5. Tree corresponding to sequence 4 in Fig. 2.

There are eight trees from the node corresponding to the whole assembly (node 1) to the set of nodes corresponding to subassemblies that contain one part only (nodes 9, 10, 11, and 12). They are shown in Figs. 5–12.

One important feature of the trees shown in Figs. 5–12 is that the distinction between operations becomes apparent because distinct operations correspond to distinct hyperarcs. In other words, two distinct assembly sequences include the same operation only if the two corresponding trees include the hyperarc corresponding to that operation. Hyperarc 1, for example, is present in the solution trees in Figs. 5–7; therefore, the same assembly operation is part of three distinct sequences. Conversely, the operations "screw the receptacle and the cap" in sequences A-B-C, B-A-C, and B-C-A of Fig. 2 correspond to hyperarcs 1, 5, and 13 in Fig. 4; therefore,

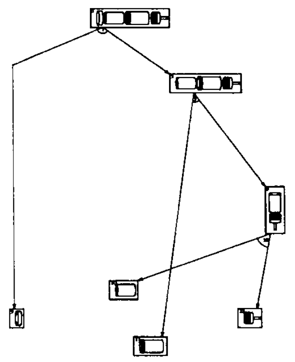


Fig. 6. Tree corresponding to sequence 8 in Fig. 2.

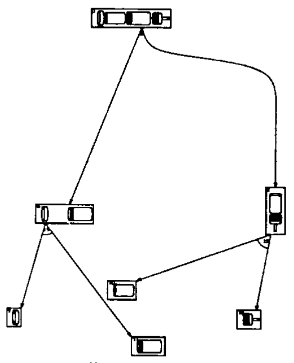


Fig. 9. Tree corresponding to sequences 9 and 10 in Fig. 2.

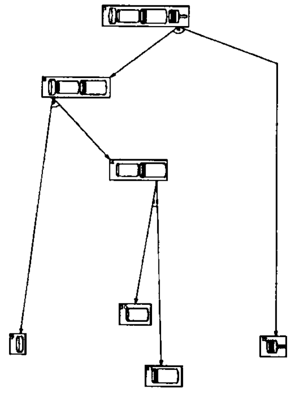


Fig. 10. Tree corresponding to sequence 2 in Fig. 2.

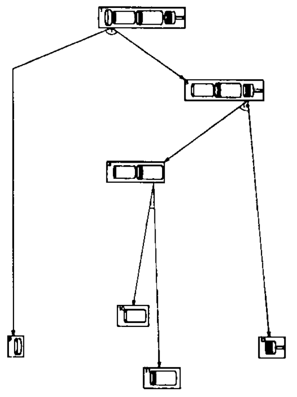


Fig. 7. Tree corresponding to sequence 3 in Fig. 2.

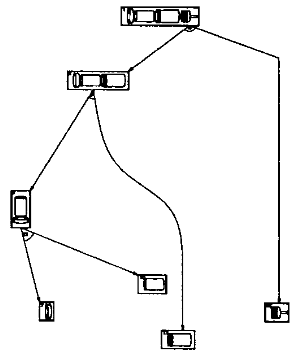


Fig. 11. Tree corresponding to sequence 5 in Fig. 2.

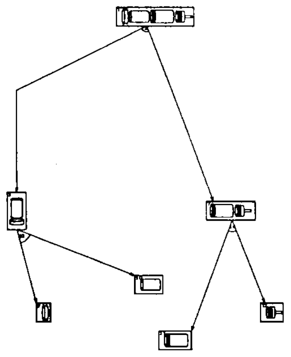


Fig. 8. Tree corresponding to sequences 6 and 7 in Fig. 2.

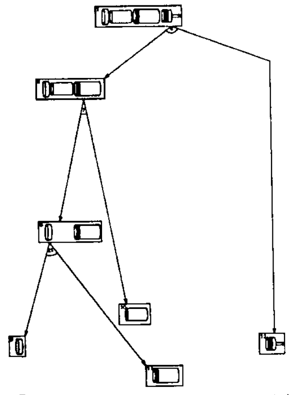


Fig. 12. Tree corresponding to sequence 1 in Fig. 2.

they are three different operations. The sequence diagrams in Fig. 2 and the precedence diagrams in Fig. 3 fail to make this distinction.

Each tree shown in Figs. 8 and 9 corresponds to two sequences, but unlike the precedence diagrams of Fig. 3, the operations are exactly the same, regardless of the order in which they are executed. Whenever a tree corresponds to more than one sequence, there are assembly operations that can be executed in parallel. For example, for the tree shown in Fig. 8, the assembly operations corresponding to hyperarcs 12 and

14 can be executed in parallel, but they do not have to be executed in parallel. The tree shown in Fig. 8 indicates that there is no time dependence between those two assembly operations.

V. A Top-Down Search for Finding the Best Plan

To solve problems that require optimization, such as the selection of the best assembly plan, one must be able to traverse the space of all candidate solutions, regardless of the method used to solve the problem. The choice of the representation is critical since it is often difficult to delimit the set of potential solutions in a form that enumerates all the elements.

The AND/OR graph representation encompasses all possible ways to assemble one product and therefore allows the exploration of the space of all possible plans. Since plans correspond to trees in the AND/OR graph, the selection of the best plan can be seen as a search problem. Any such search problem requires a criterion to compare plans. One possibility is to assign to the hyperarcs weights that are proportional to the difficulty of their corresponding operations and then compute the cost of a tree from a node, recursively, as

- Zero, if the node has no leaving hyperarc; or
- the sum of the weight of the hyperarc leaving the node and the cost of the trees from the successor nodes.

The best plan corresponds to the tree that has the minimum cost. For this type of evaluation function, the search for the best plan can be conducted using generic algorithms such as the AO* [13], [14].

A variety of factors might be considered in assigning weights to hyperarcs including the time duration of their corresponding operations, requirements for reorientation and fixturing, cost of resources needed, reliability, and production priorities and constraints. For the product in Fig. 1, the AND/OR graph (Fig. 4) has 14 hyperarcs, which correspond to 15 different assembly operations. Table II shows one possible assignment of weights to hyperarcs. Those weights have been computed by adding two factors. The first factor is the type of assembly operation, with screw operation weighing 4, insertion 2, and placement 1, in accord with typical time, fixturing, and manipulation requirements. The second factor taken into account is the difficulty of handling the participating subassemblies and is proportional to their number of degrees of freedom. Subassemblies with more degrees of freedom are more unstable and therefore more difficult to handle.

Using that assignment of weights to hyperarcs, the total cost for the trees of Figs. 5–12 were computed and are shown in Table III. The trees in Figs. 5 and 12 have the minimum cost of 11.

For more complex assemblies, instead of a complete enumeration as done above, search algorithms can be used to reduce computation. For the product in Fig. 1, a top-down search using AO* will yield one of the trees shown in Figs. 5 or 12, depending on how the partial solutions and tip nodes are ordered for expansion.

A. Disassembly for Maintenance or Repair

For maintenance and repair applications, it is often necessary to find the best way to disassemble a product. AND/OR graphs can be used in planning the disassembly of a product, provided that the hyperarcs correspond to feasible disassembly operations. The approach is analogous to that of the assembly planning problem. Since operations are not necessarily reversible, there may be two different AND/OR graphs for the same product: one corresponding to assembly and another corresponding to disassembly. Furthermore, even when an operation is reversible, the weight assigned to a hyperarc in the assembly AND/OR graphs may be different from the weight assigned to the corresponding hyperarc in the disassembly AND/OR graph because the difficulty of two reverse operations may be different.

One common objective of maintenance is to replace or to examine one part, and there is no need to completely disassemble the product. A top-down search of the disassembly AND/OR graph can be used to find the best way to release a given part. The approach is similar to that of finding the best assembly plan, except that the set of goal nodes also includes nodes that correspond to subassemblies that do not contain the target part since those nodes correspond to subassemblies that do not require further disassembly. Instead of being defined explicitly by the enumeration of its elements, the set of goal nodes is defined implicitly as

$$\Omega = \{\sigma | (\sigma \in S_P) \wedge [(\sigma = \{p^*\}) \vee (p^* \notin \sigma)]\}$$

where p^* is the part to be replaced.

One example of the use of the AND/OR graph in planning the repair of the simple product shown in Fig. 1 is to find the easiest way to release the stick. For simplicity of exposition, it is assumed that all the assembly operations for that product are reversible, and therefore, the disassembly and the assembly

TABLE II
Assignment of Weights to Hyperarcs

Hyperarc	Operation Complexity	Subassembly Degrees of Freedom	Total Cost
1	4	1	5
2	4	3	7
3	4	3	7
4	4	1	5
5	4	2	6
6	4	3	7
7	2	0	2
8	2	0	2
9	4	3	7
10	4	2	6
11	2	0	2
12	1	0	1
13	4	0	4
14	4	0	4
15	1	0	1

TABLE III
Total Costs for Trees of Figs. 5 to 12

Figure	5	6	7	8	9	10	11	12
Cost	11	13	13	12	12	13	13	11

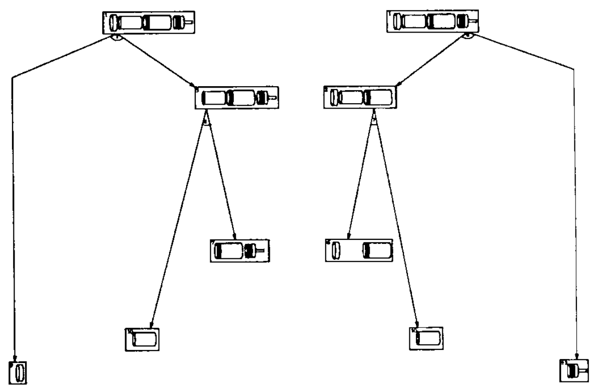


Fig. 13. Two best trees for disassembly of product of Fig. 1 aimed at releasing the stick.

AND/OR graph are the same. It is also assumed that the weights associated with the hyperarcs in the disassembly graph are the same weights associated with the corresponding hyperarcs in the assembly graph, which are shown in Table II. A top-down search of the disassembly AND/OR graph would indicate the best ways to disassemble the product in order to release the stick. For this search, the terminal nodes include the nodes corresponding to one-part subassemblies (i.e., nodes 9, 10, 11, and 12) and all the nodes corresponding to subassemblies that do not contain the stick (i.e., nodes 6 and 7). Fig. 13 shows the two best trees for that disassembly.

VI. A Bottom-Up Search for Error Recovery

Whenever unexpected events cause the execution of an assembly to deviate from the preplanned course of action, it is preferred that the assembly proceed in the most efficient way from the unpredicted state toward the goal.

STRIPS [6], [7] and SIPE [20], [21] are domain-independent planning systems that address error recovery. In both systems, the strategy is to preserve as much of the original plan as possible. There are many situations in which adhering to the original plan, although feasible, is not the most efficient approach to error recovery. For example, suppose that time considerations led the planner to opt for assembling in parallel two subassemblies to be joined later. If, in joining the two subassemblies, one of them falls apart, it may be better to complete the assembly following a different plan instead of rebuilding the subassembly that fell.

The AND/OR graph can be used to dynamically adapt the assembly plan to changing conditions since it encompasses all possible ways to assemble the product. Whenever the assembly process evolves to a situation that is not in the current plan, a bottom-up search [12] of the AND/OR graph will indicate the new best assembly plan. This bottom-up search should start from the current (unpredicted) situation and go up to the assembled product.

An example will illustrate the use of the AND/OR graph for recovery from execution errors. In the previous section, a criterion for assessing the quality of plans was presented. Using this criterion for the product shown in Fig. 1, and the assignment of weights shown in Table II, the best assembly plans are those corresponding to the trees shown in Figs. 5 and 12. Suppose that the assembly plan corresponding to the solution tree shown in Fig. 12 is chosen for execution. In the normal execution of this assembly plan, the cap and the receptacle are joined first. The subassembly produced is then joined with the stick, and finally, the new subassembly is joined with the handle.

One error condition that may happen after the execution of the second operation (i.e., the insertion of the stick) is the accidental disassembly of the subassembly shown in node 2 (cap, stick, and receptacle) into the subassemblies shown in node 5 (cap and stick) and node 11 (receptacle). To recover from this unexpected situation, a bottom-up search of the AND/OR graph would be performed, starting from nodes 5, 11, and 12, using the same assignment of weights shown in Table II. This bottom-up search would indicate the assembly plan shown in Fig. 14 as the most efficient way to complete the assembly. If plans were represented by fixed sequences or by precedence diagrams, this kind of recovery, which avoids the complete disassembly of the product, would not be feasible without time-consuming replanning.

VII. Opportunistic Scheduling Using the And/or Graph Representation

To evaluate how the use of AND/OR graph representation for assembly plans affects assembly efficiency, a comparative analysis among the three representation schemes discussed in this paper has been conducted.

The product in Fig. 1, and the robot workstation of Fig. 15, have been used as examples. The workstation is equipped with two manipulators, and the parts are presented in random order. It is assumed that a cap, a stick, a receptacle, and a handle always come together, varying only in their order. It is also assumed that both manipulators are controlled by the same workstation control unit [1], and they both are able to execute the following actions:

- *Acquire*: fetching, by one of the manipulators, one part from the part feeder.
- *Buffer*: temporarily storing one part into a fixed location within the workstation.
- *Mate*: joining two subassemblies that are currently held by the manipulators.
- *Retrieve*: fetching, by one of the manipulators, one part known to be in the parts buffer.

The efficiency of this assembly station depends on the capacity to handle parts in random order. This requires on-line scheduling of system resources depending on the order of parts arrival. The relative impact of plan representation schemes on assembly efficiency can be compared by the average number of operations needed; a smaller average number of operations corresponds to more efficiency.

The first sequence of Fig. 2 (A-B-C) has been used as an example of fixed-sequence representation, and the first precedence diagram of Fig. 3 (which combines A-B-C and B-A-C) has been used as an example of precedence graph representation. Similar results will be produced using the other fixed sequences or precedence graphs. The number of operations that would be performed for each one of the 24 pos-

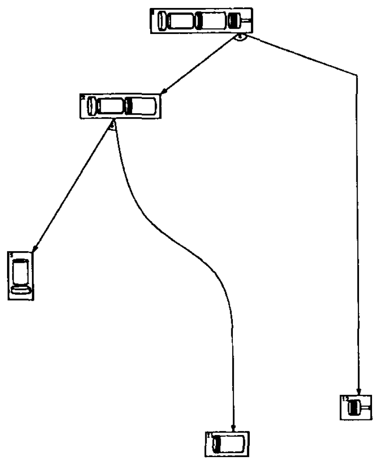


Fig. 14. Tree for recovery from unpredicted state.

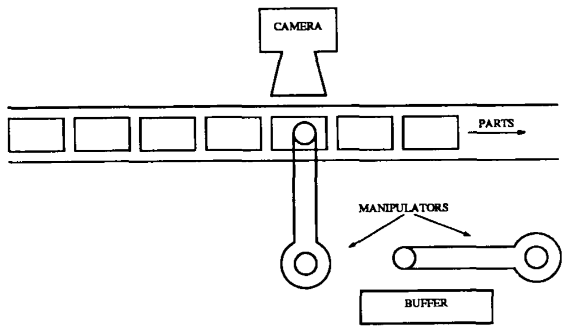


Fig. 15. Robotic workstation.

TABLE IV
NUMBER OF OPERATIONS NEEDED TO ASSEMBLE PRODUCT OF FIG. 1 FOR ALL SEQUENCES IN WHICH PARTS MAY BE ACQUIRED AND FOR THE THREE SCHEMES OF PLAN REPRESENTATION

Sequence*	First Sequence of Fig. 2	First Precedence Diagram of Fig. 3	AND/OR Graph of Figure 4
CSRH	9	9	7
CSHR	11	11	9
CRSH	7	7	7
CRHS	9	9	9
CHSR	11	11	9
CHRS	9	9	9
SCRH	9	9	7
SCHR	11	11	9
SRCH	9	7	7
SRHC	11	9	7
SHCR	11	11	9
SHRC	11	9	7
RCSH	7	7	7
RCHS	9	9	9
RSCH	9	7	7
RSHC	11	9	7
RHCS	9	9	9
RHSC	11	9	7
HCSR	11	11	9
HCRS	9	9	9
HSCR	11	11	9
HSRC	11	9	7
HRCS	9	9	9
HRSC	11	9	7
Average	9.8	9.2	8

C—cap; *S*—stick; *R*—receptacle; *H*—handle.

sible orderings in which the four parts of the simple product can be acquired is shown in Table IV. At least seven operations are necessary: four acquisitions, three matings (depending on the order in which the parts are presented), buffering, and therefore retrieving, may also be necessary.

When using the fixed-sequence representation of plans, extensive buffering is necessary. For example, if the order the parts come is *SHRC* (stick, handle, receptacle, and cap), both the stick and the handle must be buffered since they are not used in the first operation; adding two bufferings and two retrievings to the four acquisitions and three matings that are always necessary yields 11 operations. The average number of operations for all 24 possible orders is 9.8.

Using precedence diagrams for the representation of plans avoids some of the buffering and reduces the average number of operations to 9.2. For the sequence *SHRC*, for example, only the handle must be buffered since the insertion of the stick into the receptacle may be the first operation.

Using the AND/OR graph representation of plans avoids most of the buffering and yields the average of eight operations. For the same *SHRC* sequence, for example, no buffering is needed because the robot can follow the sequence of operations corresponding to the tree shown in Fig. 6.

This example has emphasized the efficiency of the AND/OR graph representation of plans, comparing a single fixed sequence to a single precedence diagram with a single AND/OR graph, illustrating the tradeoff between the efficiency of execution of the plan and the space required to store the plan. A complete set of fixed sequences would be required to obtain the same average number of operations obtained using the AND/OR graph in the opportunistic scheduling case discussed. Similarly, a complete set of precedence diagrams would be required in order to obtain the same average number of operations. In both cases, the representation size required for the complete set of sequences would be larger than that required for storage of the AND/OR graph. As discussed in Section III, the AND/OR graph has, in general, substantially fewer nodes than the directed graph, which in turn eliminates some of the redundancies of sets of fixed sequences and of sets of precedence diagrams. This example of opportunistic scheduling illustrates the basic property that for a given plan representation size, the average number of operations is reduced by using the AND/OR graph, or alternatively, to obtain the same average number of operations, the AND/OR graph requires less space. In addition, use of the AND/OR graph will in most cases require less search for the best sequence. Although the search using the AND/OR graph will prune sets of less desirable sequences, the search using the fixed sequences would prune one by one. A quantitative comparison of the complexity of the search itself in each case is beyond the scope of this paper.

The relative efficiency of the AND/OR graph representation is not uniform and depends on the assembly as well. If the assembly is such that there is only one fixed sequence in which it can be assembled, then using the AND/OR graph yields no savings in storage requirements, but for most products the AND/OR graph drastically reduces the storage requirements and results in lower search complexity as well. As an example of a practical industrial problem, we have studied the ten-part

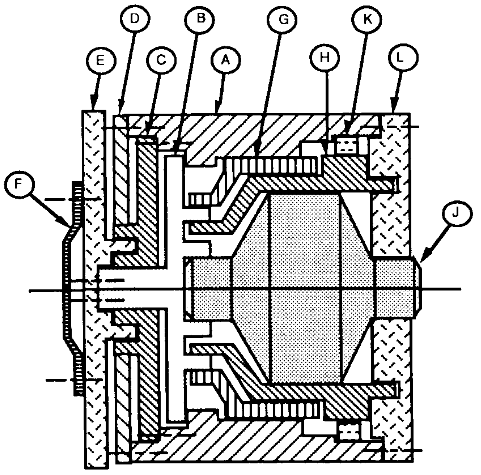


Fig. 16. Ten-part assembly from industry.

TABLE V
STORAGE REQUIREMENTS FOR 10-PART ASSEMBLY FROM INDUSTRY

	AND/OR Graph	Precedence Diagrams	Fixed Sequences
Nodes	75	33 190	507 480
Arcs	221	29 871	456 732

assembly from industry introduced by De Fazio and Whitney [3] and shown in Fig. 16. We have generated the AND/OR graph, the precedence diagrams, and the fixed sequences for this example. The number of nodes and the number of arcs needed to store these representations are shown in Table V. There are 50 748 different fixed sequences, each requiring ten nodes and nine arcs. The AND/OR graph for that product contains only 75 nodes and 221 hyperarcs. The search through the AND/OR graph will also be more efficient than the search through the fixed sequences, though the specific complexity depends on the search method and the heuristic used. For this example, a breadth-first search using the fixed sequences, will include 456 732 expansions, whereas a breadth-first search of the AND/OR graph, assuming uniform branching factor, will include 243 302 expansions.

VIII. Conclusion

The AND/OR graph representation of assembly plans described in this paper constitutes a compact representation of all feasible assembly sequences. It provides a useful tool for the selection of the best assembly plan for the selection of the best disassembly or repair plan, recovery from execution errors, and opportunistic scheduling. Unlike the directed graph of assembly states, the AND/OR graph can show explicitly the possibility of parallel execution of assembly operations and the time independence of operations that can be executed in parallel.

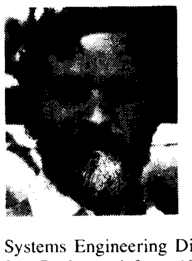
A number of issues related to this representation are under investigation [11]. Automatic construction of the AND/OR graph from design descriptions, development of evaluation functions for selection among alternative plans, and the incorporation of the mechanical manipulation capability requirements are key issues. Computational complexity and storage requirements are also important topics of continuing research.

Acknowledgment

Fig. 16 was kindly provided by T. L. DeFazio and D. E. Whitney.

References

[1] A. J. Barbera, "An architecture for a robot hierarchical control system," Nat. Bur. Standards, Spec. Pub. 500-23, U.S. Gov. Printing Office, Washington, DC, 1977.
[2] A. Bourjault *et al.*, "Elaboration automatique des gammes d'assemblage à l'aide de réseaux de petri," *R.A.I.R.O. APII*, vol. 21, pp. 323-342, 1987.
[3] T. L. De Fazio and D. E. Whitney, "Simplified generation of all mechanical assembly sequences," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 640-658, Dec. 1987; corrections, *ibid.*, vol. 4, pp. 705-708, Dec. 1988.
[4] T. Donnellan, *Lattice Theory*. New York: Pergamon, 1968.
[5] S. E. Fahlman, "A planning system for robot construction tasks," *Artificial Intell.*, vol. 5, pp. 1-49, 1974.
[6] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intell.*, vol. 2, pp. 189-208, 1971.
[7] R. E. Fikes *et al.*, "Learning and executing generalized robot plans," *Artificial Intell.*, vol. 3, pp. 251-288, 1972.
[8] B. R. Fox, "A representation for serial robotic rasks," Ph.D. dissertation, Univ. Missouri, Rolla, 1987.
[9] B. R. Fox and K. G. Kempf, "Opportunistic scheduling for robotics assembly," in *Proc. IEEE Int. Conf. Robotics Automat.*, 1985, pp. 880-889.
[10] C. Green, "Theorem-proving by resolution as a basis for question-answering systems," in *Machine Intell.*, vol. 4. New York: American Elsevier, 1969, pp. 183-205, ch. 11.
[11] L. S. Homem de Mello, "Task sequence planning for robotic assembly," Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, PA, May 1989.
[12] A. Martelli and U. Montanari, "Additive AND/OR graphs," in *IJCAI*-3, 1973, pp. 1-11.
[13] N. J. Nilsson, *Principles of Artificial Intelligence*. New York: Springer-Verlag, 1980.
[14] J. Pearl, *Heuristics*. Reading, MA: Addison-Wesley, 1984.
[15] E. D. Sacerdoti, "Planning in a hierarchy of abstraction spaces," in *Proc. 3rd Int. Joint Conf. Artificial Intell.* (Stanford, CA), 1973, pp. 412-422.
[16] E. D. Sacerdoti, "The nonlinear nature of plans," in *Adv. Papers 4th Int. Joint Conf. Artificial Intell.*, Sept. 1975, pp. 206-214.
[17] E. D. Sacerdoti, *A Structure for Plans and Behavior*. Amsterdam: Elsevier North-Holland, 1977.
[18] M. Stefik, "Planning with constraints (MOLGEN: Part 1)," *Artificial Intell.*, vol. 16, pp. 111-140, 1981.
[19] M. Stefik, "Planning and meta-planning (MOLGEN: Part 2)," *Artificial Intell.*, vol. 16, pp. 141-170, 1981.
[20] D. E. Wilkins, "Domain-independent planning: Representation and plan generation," *Artificial Intell.*, vol. 22, pp. 269-301, Apr. 1984.
[21] D. E. Wilkins, "Recovering from execution errors in SIPE," *Comput. Intell.*, vol. 1, pp. 33-45, Feb. 1985.



Luiz S. Homem de Mello (S'86-M'89) received the degree of electrical engineer from the University of Sao Paulo, Brazil, the M.S. degree in electrical engineering and computer science from the University of California, Berkeley, and the Ph.D. degree in electrical and computer engineering from Carnegie-Mellon University, Pittsburgh, PA.

He currently is at the Jet Propulsion Laboratory of the California Institute of Technology, Pasadena, working in the Robotic Intelligence Group. From 1980 to 1984, he was a Research Scientist in the Systems Engineering Division of the Instituto de Pesquisas Tecnológicas in Sao Paulo, and from 1982 to 1984, he was also an Assistant Professor of

Electrical Engineering at the University of Sao Paulo. Prior to that, he had appointments as a Systems Engineer, first with the Sao Paulo Subway Company and later with Consórcio Nacional de Engenheiros Consultores, both in Sao Paulo. His current research interests include robotics, intelligent control, and artificial intelligence.

Dr. Homem de Mello is a member of AAAI.

Arthur C. Sanderson (M'74-SM'86) received the B.S. degree from Brown University, Providence, RI, in 1968, and the M.S. and Ph.D. degrees from Carnegie-Mellon University, Pittsburgh, PA, in 1970 and 1972, respectively.

From 1968 to 1970, he was a Research Engineer at Westinghouse Research Laboratories and worked on the design and simulation of solid-state electronic devices. From 1972 to 1973, he was a Visiting Research Fellow at Delft University of Technology, Delft, The Netherlands, where he conducted research in the areas of signal processing and pattern recognition. From 1973 to 1987, he was a faculty member in the Department of Electrical and Computer Engineering at Carnegie-Mellon University, and from 1980 to 1987, he was Professor of Electrical and Computer Engineering. He participated in the founding and development of the Robotics Institute at CMU and was the Associate Director from 1980 to 1987, coordinating many of its research initiatives. From 1985 to 1987, he was on leave from CMU and held the position of Director of Information Sciences Research at Philips Laboratories, Briarcliff Manor, NY. In 1987, he joined Rensselaer Polytechnic Institute, Troy, NY, as Professor and Department Head of the Electrical, Computer, and Systems Engineering Department. He is Codirector of the New York State Center for Advanced Technology in Automation and Robotics and a Coprincipal Investigator in the RPI/NASA Center for Intelligent Robotics Systems in Space Exploration. He is the author of over 120 technical publications and proceedings. His current research interests include planning systems for robots and automation systems, sensor-based control, computer vision, and applications of knowledge-based systems.

Dr. Sanderson is President of the IEEE Robotics and Automation Society. He was Associate Editor of the IEEE JOURNAL OF ROBOTICS AND AUTOMATION from 1984 to 1989, Program Chairman for the 1987 IEEE International Conference on Robotics and Automation, and Chairman of the 1989 IEEE International Symposium on Intelligent Control. He is a member of AAAI, SME, and AAAS.