# AND/OR GRAPH REPRESENTATION OF ASSEMBLY PLANS*

Luiz S. Homem de Mello and Arthur C. Sanderson

Department of Electrical and Computer Engineering
and Robotics Institute
Carnegie-Mellon University
Pittsburgh Pennsylvania 15213

## ABSTRACT

This paper presents a compact representation of all possible assembly plans of a given product using AND/OR graphs. Such a representation forms the basis for efficient planning algorithms which enable an increase in assembly system flexibility by allowing an intelligent robot to pick a course of action according to instantaneous conditions. Two applications are discussed: the selection of the best assembly plan (off-line planning), and opportunistic scheduling (on-line planning). 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 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. The scheduling efficiency using this representation is compared to fixed sequence and precedence graph representations. The AND/OR graph consistently reduces the average number of operations.

## I INTRODUCTION

Robotic assembly often requires reprogramming or reconfiguration in order to handle a variety of designs in the same system. The design and implementation of such flexible systems is difficult, and automated planning techniques may provide major advantages. Such task planning for robotic assembly is critically dependent on the task representation; a new approach to task representation using AND/OR graphs is described in this paper.

Flexibility in robotic workcells provides a number of advantages. Flexible robotic workcells may be reconfigured to handle a wide range of styles and products. Further flexibility can be achieved if those workcells are able to assemble the same product in different ways. In order to accomodate the assembling of several different products in the same shop, it is necessary to schedule the available machines to each job. Since different machines may have different capabilities, the assembly procedure may vary depending on what machine is scheduled to do the job. Another advantage is an improvement in the ability to recover from errors and other unexpected effects that cause the execution of a task to deviate from the preplanned course of actions. When deviations occur, it is preferred that the task execution continue, as efficiently as possible, from the unpredicted state towards the goal. Deviations of the desired course of actions are not necessarily error conditions, but may be due to random factors that affect the manufacturing process, and flexible shops should be able to cope with those factors autonomously.

*This research is supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (Brazil) and by the Robotics Institute of Carnegie-Mellon University.

Even with flexibility of the mechanical hardware, current robotic assembly systems are not able to follow many different courses of actions within a given task. A principal reason for this limitation is the inadequate data structure for the representation of task plans. Ordered lists of actions, that have been used in early robot systems, which were developed outside the manufacturing context, do not permit flexibility in task execution. Triangle tables [Fikes 72] have been used for the representation of plans, and they improve the capability to recover from errors, but only within one fixed sequence. A more significant improvement was the use of precedence diagrams [Fox,B. 85] for the representation of plans, but that technique has limitations also, and in most cases allows only a small amount of flexibility.

This paper presents a compact representation for the set of all possible assembly plans of a given product. Such a representation enables an increase in assembly flexibility by allowing an intelligent robot to pick the more convenient course of actions, according to the instantaneous conditions at the shop. In sections II and III, the necessary background is established. Section IV shows the representation, and section V presents its use for the assembly of a simple product. Two applications are discussed: section VI shows how the selection of the best assembly plan can be implemented as a graph search, and section VII shows the use of the representation in opportunistic scheduling. Section VIII summarizes the contribution of the paper and points to further research.

## II SCHEDULING AND PLANNING

Assembly of one product requires selection of a sequence of operations and assignment of times and resources for each operation. The problem is usually divided into two parts: planning, or process routing, which is the selection of a sequence of operations, and scheduling, which is the assignment of times and resources.

Scheduling problems, including job-shop scheduling, project scheduling, and assembly-line balancing, have been intensively investigated in Management Sciences and Operations Research [Bellman 82]. Mathematical programming techniques have most often been used to solve those problems. More recently, the scheduling problem has been studied using constraint-directed reasoning [Fox,M. 83].

Planning has been an important research issue in artificial intelligence. BUILD [Fahlman 74] and STRIPS [Fikes 71] are two early examples. Both systems aim to generate plans that enable robots to perform certain tasks. Typically, the tasks consist of achieving a state that satisfies some goal condition from a current state of the *world* (i.e., the robot environment), and the plans consist of ordered sequences of actions that will transform the initial state into a goal state.

The representation of plans are commonly based on ordered lists of preprogrammed primitive actions. There are some extensions to that representation scheme that enable the robot to take advantage of the work already done in planning, in case unexpected events happen during the execution of a plan. STRIPS, for example, uses a tabular form, called a *triangle table*, to store a plan. BUILD associates to each primitive action a REASON list (subgoals) as well as a description of the states of the world before and after the action is executed. More recent systems, such as NOAH [Sacerdoti 77], represent plans as partially ordered sequences of actions with respect to time.

A major emphasis of research on planning has been on the search aspect of the problem, especially control schemes for the search. Priority has been given to develop efficient, powerful and 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 type of situation. In applications where plans are executed one time only, inefficiencies in the plan do not cause any major harm. Also, if plans are generated on line, high speed in plan generation is often preferable to optimal plans.

Search for the most efficient plan requires a criterion to decide whether one plan is better than another. This decision, however, usually requires information available at execution time only, and producing the plan in real time may degrade the robot operation, or even be unfeasible, 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.

## III PLANNING FOR ROBOTIC ASSEMBLY

To achieve the desired high levels of productivity, the assembly plans must be efficient and keep wasted time and resources to a minimum. Should inefficiencies in the assembly plan of one product be multiplied by the size of the lot, which in common robotic assembly applications ranges from 1,000 to 100,000 units, the resulting total waste may reduce drastically the productivity and may jeopardize the whole process. Conditions at the shop, however, change with time (for example, parts may come in random order), and usually, there is no single plan that is efficient in every possible situation.

Fox and Kempf [Fox,B. 85] address the need to act opportunistically, as opposed to always follow a preprogrammed fixed order of operations. They suggest that plans generated off-line to be given to the robot 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 at the shop, the intelligent robot would pick the most appropriate sequence. Using Fox and Kempf notation, the selection of one sequence, and the assignment of operations to specific machines is what is commonly referred to as the scheduling process. Since that selection process involves much less computing time than the planning process, no degradation in the efficiency of the robot operation should occur.

Planning, in this sense, should yield all possible sequences of operations that can be used to assemble a product. That information is the input to the scheduling process, which in real time selects one of those sequences and assigns the machines that will do each operation.

The problem with the precedence diagram formalism, as Fox and Kempf themselves point out, 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 figure 1, for example, may be completed by following one of the ten different sequences of operations that are represented graphically in figure 2. It is possible to combine some sequences into one partial order using precedence diagrams. Figure 3 shows three possible ways to combine two of the first four sequences in figure 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 it is possible to combine any of those sequences with the other six sequences in figure 2.

A closer look at the partial ordering representation of plans, in the 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 sequence 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 receptacle 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, however, does not capture this subtle difference. The next section will describe another approach to the representation of plans that captures this difference, and that can combine all possible assembly sequences.

## IV AND/OR GRAPH REPRESENTATION OF ASSEMBLY PLANS

Planning the assembly of one product made up of several component parts can be seen as path search in the state space of all possible configurations of that set of parts. The initial state is that configuration in which all parts are disconnected from each other, and the goal state is that in which the parts are properly joined to form the desired product. The moves that change one state into another correspond to the assembly operations since they change the relative position of at least one part. There may be many different paths from the initial state to the goal state. Krogh and Sanderson [Krogh 85] present an overview of task decomposition and operations.

In this context, any set of parts that are joined to form a stable unit is called an *assembly*. A component part is also an assembly, with a special property. The word *subassembly* refers to an assembly that is part of another, more complex assembly, and it always carries the subset/set connotation.

Because there are many configurations that can be made from the same parts, the branching factor from the initial state to the goal state is greater than the branching factor from the goal state to the initial state. A backward search, therefore, will be more efficient than a forward search for the assembly planning problem. The problem of finding how to assemble a given product can be converted to an equivalent problem of finding how the same product can be *disassembled*. Since assembly operations are not necessarily reversible, the equivalence of the two problems will hold only if each operation used in *disassembly* is the reverse of a feasible assembly operation, regardless of whether these reverse operations themselves are feasible or not. The expression *disassembly operation*, therefore, refers to the reverse of a feasible assembly operation.

The backward search suggests a decomposable production system in which the problem of *disassembling* one product is decomposed into distinct subproblems, each one being to *disassemble* one subassembly. Each decomposition must correspond to a *disassembly operation*. If solutions for both subproblems that result from the decomposition are found, then a solution for the original problem can be obtained by combining the solutions to the subproblems and the operation used in the decomposition. For subassemblies that contain one part only, a trivial solution containing no operation always exists. Usually there will not be a unique way to decompose the problem, or to *cut* the assembly, because there may be several different ways to assemble the same product.

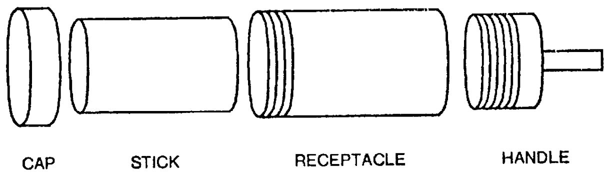

Figure 1: A simple product

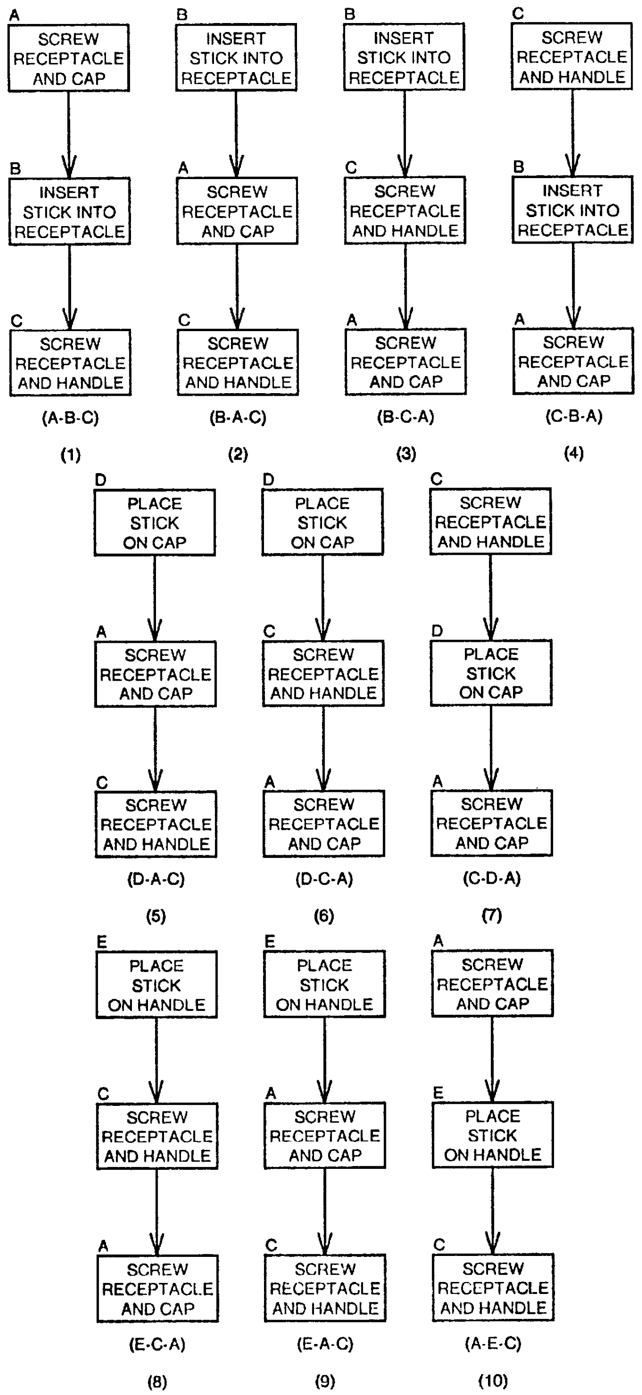
A SCREW RECEPTACLE AND CAP
B INSERT STICK INTO RECEPTACLE
C SCREW RECEPTACLE AND HANDLE
(A-B-C) (1)
B INSERT STICK INTO RECEPTACLE
A SCREW RECEPTACLE AND CAP
C SCREW RECEPTACLE AND HANDLE
(B-A-C) (2)
B INSERT STICK INTO RECEPTACLE
C SCREW RECEPTACLE AND HANDLE
A SCREW RECEPTACLE AND CAP
(B-C-A) (3)
C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

Figure 2: Possible sequences of operations to assemble the product shown in figure 1

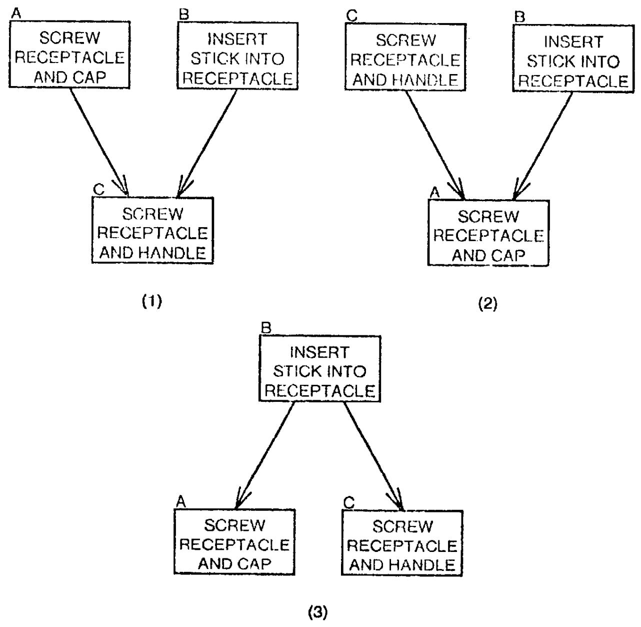
A SCREW RECEPTACLE AND CAP
B INSERT STICK INTO RECEPTACLE
C SCREW RECEPTACLE AND HANDLE
(1)
C SCREW RECEPTACLE AND HANDLE
B INSERT STICK INTO RECEPTACLE
A SCREW RECEPTACLE AND CAP
(2)
B INSERT STICK INTO RECEPTACLE
A SCREW RECEPTACLE AND CAP
C SCREW RECEPTACLE AND HANDLE
(3)

Figure 3: Precedence diagrams: *(1)* combines A-B-C and B-A-C; *(2)* combines C-B-A and B-A-C; *(3)* combines B-A-C and B-C-A

Structures called AND/OR graphs [Nilsson 80], or *hypergraphs*, are useful in representing decomposable problems and they have been used to represent the *disassembly* problem. The nodes in such a hypergraph correspond to assemblies; nodes corresponding to assemblies that contain only one part are the terminal nodes. The hyperarcs (or *k*-connectors, *k* being any integer greater than zero) correspond to the *disassembly operations*. Each hyperarc that leaves one node corresponds to a *disassembly operation* applicable to the assembly of that node, and the successor nodes to which the hyperarc points correspond to the resulting subassemblies produced by the *disassembly* operation. Because for most products the assembly operations usually mate two subassemblies, the hyperarcs in the corresponding AND/OR graph are usually *2*-connectors. There are cases, however, of operations that mate more than two subassemblies (e.g., assembling a hinge with two wings and one pin), as well as operations that involve only one subassembly (e.g., drilling a hole in a part). Hyperarcs in AND/OR graphs can represent all those possibilities.

A *solution tree* from a node N in an AND/OR graph is a subgraph that may be defined recursively as either N itself if N is a terminal node, or N plus one of its outgoing hyperarcs plus the set of solution trees from each of N's successors through that hyperarc. This definition assumes that the graph contains no cycle as is true in the *disassembly* problem. There may be none, one, or several solution trees from a node in an AND/OR graph.

The useful feature of the AND/OR graph representation for the assembly problem is that it encompasses all possible partial orderings of assembly operations. Moreover, each partial order corresponds to a solution tree from the node corresponding to the final (assembled) product. This feature is demonstrated through the example in the next section.

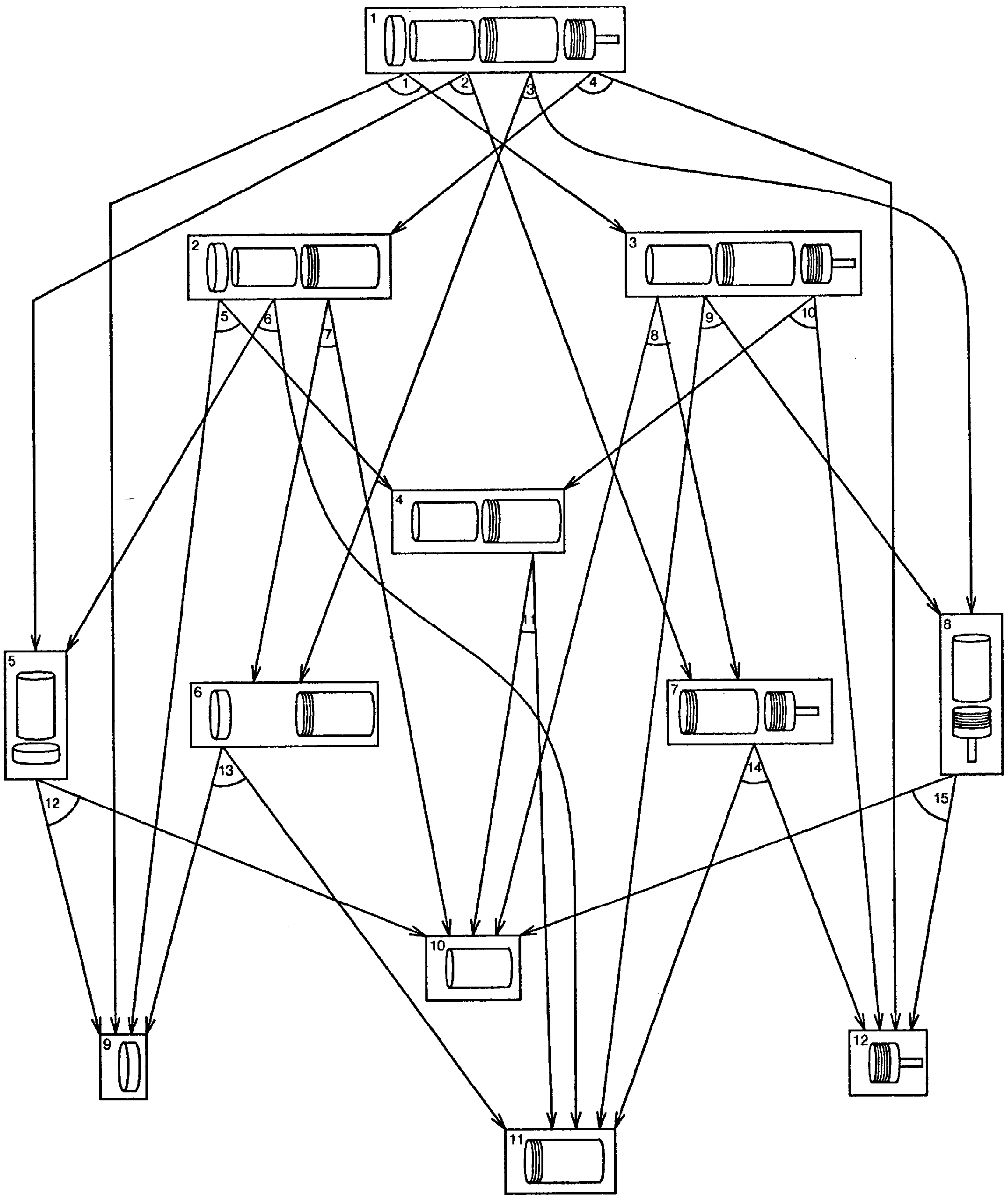

**Figure 4:** AND/OR graph for the product shown in figure 1

## V A SIMPLE EXAMPLE

Figure 4 shows the AND/OR graph for the product in figure 1. Each node in that graph is labeled by a database that correponds to an assembly. In figure 4, the databases are represented by exploded view drawings, whereas in a computational implementation, the databases are relational data structures. To facilitate the exposition, both the nodes and the hyperarcs in figure 4 have identification numbers.

The root node in figure 4 (node 1) is labeled by a database that describes the assembled product. There are four hyperarcs leaving that node. Each of those four hyperarcs corresponds to one way the whole assembly can be *disassembled* and each one points to two nodes that are labeled by databases 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.*

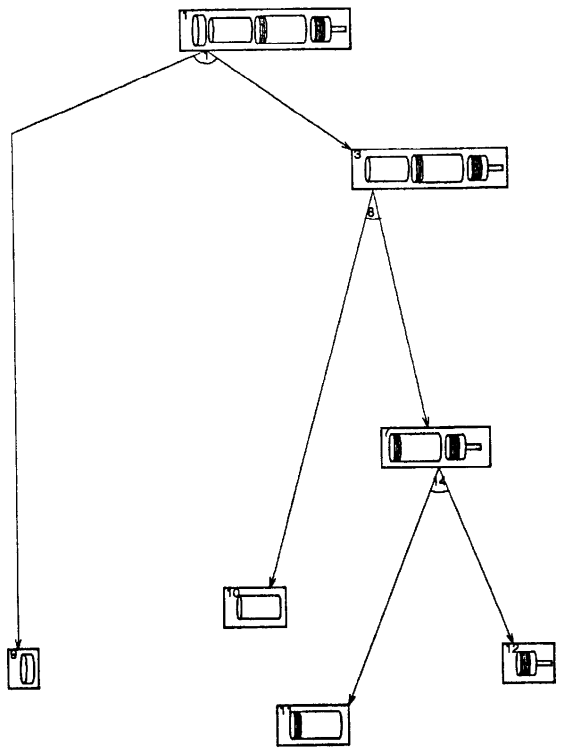

Figure 5: Solution tree corresponding to sequence 4 in fig 2

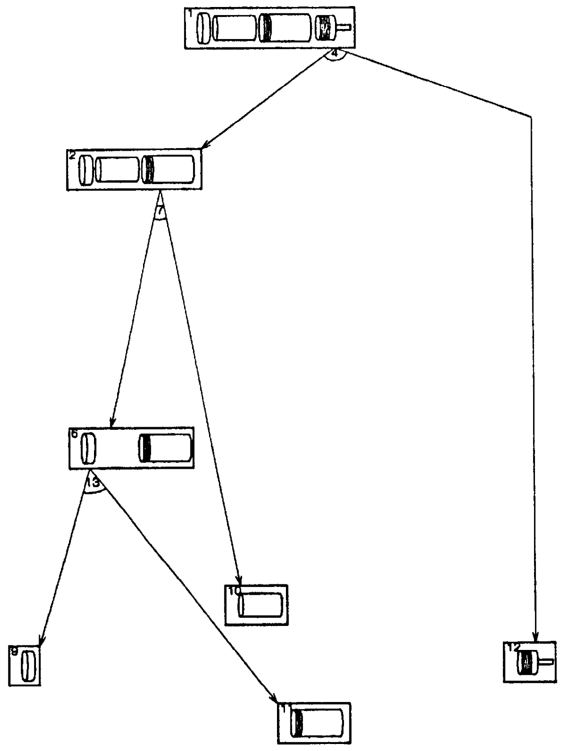

Figure 7: Solution tree corresponding to sequence 1 in fig 2

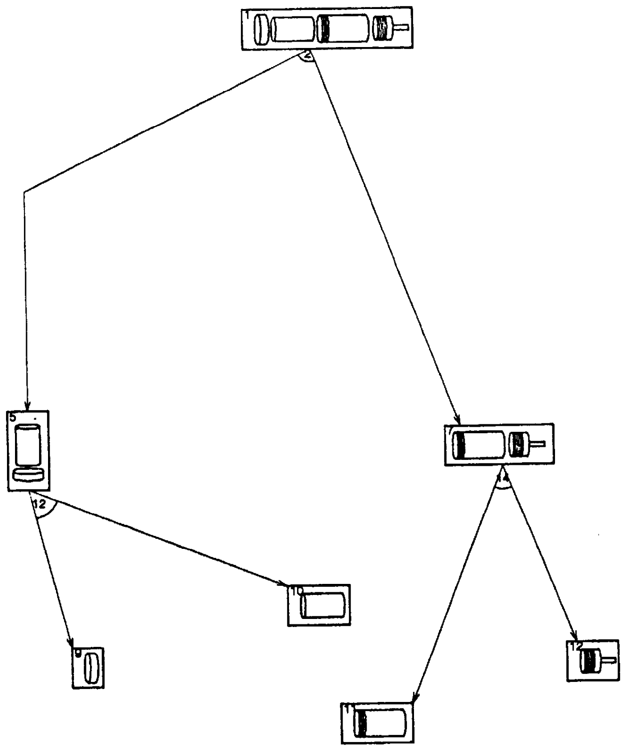

Figure 6: Solution tree corresponding to sequences 6 and 7 in fig 2

Any subassembly that can be made up of the component parts may appear only once in the graph, even when it may be the result of different *disassembly* operations. The subassembly of node 4, in figure 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. Nodes corresponding to component parts (nodes 9, 10, 11 and 12) are the terminal or goal nodes since they correspond to *disassembling* problems for which a (trivial) solution is known.

There are eight solution trees from the root node (node 1) and three of them are shown in figures 5 to 7. One important feature of the solution tree representation 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 solution trees include the hyperarc corresponding to that operation. The sequence diagrams in figure 2 and the precedence diagrams in figure 3 fail to make this distinction. The solution tree shown in figure 6 corresponds to two sequences, but unlike the precedence diagrams of figure 3, the operations are exactly the same, regardless of the order in which they are executed.

## VI FINDING THE BEST PLAN AS AN AND/OR GRAPH SEARCH

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 which enumerates all the elements.

The AND/OR graph representation encompasses all possible ways to assemble one product, and therefore allows one to explore the space of all possible plans. Since plans correspond to solution 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 proportional to the difficulty of their corresponding operations, and then compute the cost of a solution 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 costs of the solution trees from the successor nodes.

The best plan corresponds to the solution tree that has the minimum cost. The search for the best plan can be conducted using generic algorithms such as the AO* [Nilsson 80].

A variety of factors might be considered in assigning weights to hyperarcs, including time duration of their corresponding operations, requirements for reorientation of fixturing, cost of resources needed, reliability, as well as production priorities and constraints.

For the product in figure 1, the AND/OR graph (figure 4) has 15 hyperarcs, which correspond to 15 different assembly operations. Table 1 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 solution trees can be computed. The solution trees in figures 5 and 7 have the minimum cost of 11.

For more complex assemblies, instead of a complete enumeration as suggested above, search algorithms can be used to reduce computation. For the product in figure 1, a search using AO* will yield one of the solution trees shown in figures 5 or 7, depending on how the partial solutions and tip nodes are ordered for expansion.

Table 1: Assignment of weights to hyperarcs

| hyperarc | operation type | subassemblies degrees of freedom | total weight |
|---|---|---|---|
| 1 | 4 | 1 | 5 |
| 2 | 4 | 4 | 8 |
| 3 | 4 | 4 | 8 |
| 4 | 4 | 1 | 5 |
| 5 | 4 | 2 | 6 |
| 6 | 4 | 4 | 8 |
| 7 | 2 | 0 | 2 |
| 8 | 2 | 0 | 2 |
| 9 | 4 | 4 | 8 |
| 10 | 4 | 2 | 6 |
| 11 | 2 | 0 | 2 |
| 12 | 1 | 0 | 1 |
| 13 | 4 | 0 | 4 |
| 14 | 4 | 0 | 4 |
| 15 | 1 | 0 | 1 |

## VII OPPORTUNISTIC SCHEDULLING 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 figure 1, and the robot workstation of figure 8 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 central unit and they both are able to execute the following actions:

- acquire: fetching, by one of the manipulators, of one part from the part feeder
- buffer: temporarily storing one part into a fixed location within the workstation
- mate: joining two subassemblies which 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 figure 2 (A-B-C) has been used as an example of fixed sequence representation and the first precedence diagram of figure 3 (combines A-B-C and B-A-C) 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 of the 24 possible orderings in which the four parts of the simple product can be acquired is shown in Table 2. At least 7 operations are necessary: four acquisitions and 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 R H S C (receptacle, handle, stick, and cap) both the handle and the stick 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 R H S C, 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, however, avoids most of the buffering, and yields the average of 8 operations. For the same R H S C sequence, for example, no buffering is needed because the robot can follow the sequence of operations corresponding to the solution tree shown in figure 5.

## VIII CONCLUSION

A compact representation for the set of all possible assembly plans of a product has been presented, along with its applications in the selection of the best assembly plan and in opportunistic scheduling. One important feature of that representation is that it allows one to

Table 2: Number of operations needed to assemble the product of fig 1 for all the sequences in which the parts may be acquired, and for the three schemes of plan representation
C=cap S=stick R=receptacle H=handle

| sequence | first sequence in fig 2 | first precedence diagram in fig 3 | AND/OR graph fig 4 |
|---|---|---|---|
| C S R H | 9 | 9 | 7 |
| C S H R | 11 | 11 | 9 |
| C R S H | 7 | 7 | 7 |
| C R H S | 9 | 9 | 9 |
| C H S R | 11 | 11 | 9 |
| C H R S | 9 | 9 | 9 |
| S C R H | 9 | 9 | 7 |
| S C H R | 11 | 11 | 9 |
| S R C H | 9 | 7 | 7 |
| S R H C | 11 | 9 | 7 |
| S H C R | 11 | 11 | 9 |
| S H R C | 11 | 9 | 7 |
| R C S H | 7 | 7 | 7 |
| R C H S | 9 | 9 | 9 |
| R S C H | 9 | 7 | 7 |
| R S H C | 11 | 9 | 7 |
| R H C S | 9 | 9 | 9 |
| R H S C | 11 | 9 | 7 |
| H C S R | 11 | 11 | 9 |
| H C R S | 9 | 9 | 9 |
| H S C R | 11 | 11 | 9 |
| H S R C | 11 | 9 | 7 |
| H R C S | 9 | 9 | 9 |
| H R S C | 11 | 9 | 7 |
| average | 9.8 | 9.2 | 8 |

traverse the space of all possible assembly plans, and therefore provides an opportunity to select an optimal schedule and dynamically adapt scheduling to changing conditions. Both the fixed sequence representation and the precedence diagram representation are very limited in this aspect.

A number of issues related to this representation are under investigation. One important issue is the development of algorithms for opportunistic scheduling suitable for real time operation. As pointed out in section VII, some buffering could not be avoided, even with the use of AND/OR graph representation of plans. For complex products, the choice of which part or subassembly to buffer may affect the overall assembly efficiency and criteria for that decision will be necessary. These criteria will certainly depend on evaluation functions, also under investigation, used to select a plan, especially functions that do not possess the recursive property like the one used in section VI.

An additional important ongoing research issue is the development of a representation of assemblies suitable for the automatic generation of plans. Such automation can be helpful in design of both new products and assembly systems. In designing new products, the designer can quickly assess the difficulty of assembling and eventually modify the design to facilitate the assembly. In designing new assembly systems, the designer can evaluate the performance of a proposed design for a given set of products.

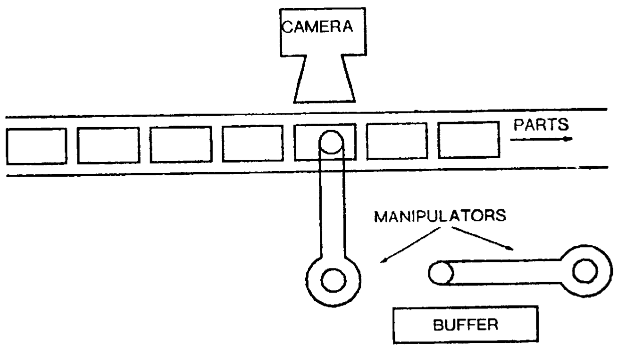

Figure 8: Robotic workstation

## REFERENCES

[Bellman 82] Bellman, R. et al. *Mathematical Aspects of Scheduling and Applications.* Pergamon Press, 1982.

[Fahlman 74] Fahlman, Scott Elliott. A Planning System for Robot Construction Tasks. *Artificial Intelligence* 5(1):1-49, 1974.

[Fikes 71] Fikes, Richard E. and Nilsson, Nils J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2:189-208, 1971.

[Fikes 72] Fikes, Richard E. et al. Learning and Executing Generalized Robot Plans. *Artificial Intelligence* 3:251-288, 1972.

[Fox,B. 85] Fox, B. R. and Kempf, K. G. Opportunistic Scheduling for Robotics Assembly. In *1985 IEEE International Conference on Robotics and Automation*, pages 880-889. IEEE Computer Society, 1985.

[Fox,M. 83] Fox, Mark S. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* PhD thesis, Carnegie-Mellon University, December, 1983.

[Krogh 85] Krogh, Bruce H. and Sanderson, Arthur C. *Modeling and Control of Assembly Tasks and Systems.* Technical Report CMU-RI-TR-86-1, Carnegie-Mellon University - The Robotics Institute, July, 1985.

[Nilsson 80] Nilsson, Nils J. *Principles of Artificial Intelligence.* Springer-Verlag, 1980.

[Sacerdoti 77] Sacerdoti, Earl D. *A Structure for Plans and Behavior.* Elsevier North-Holland, 1977.