PLANNING REPAIR SEQUENCES USING THE AND/OR GRAPH REPRESENTATION OF ASSEMBLY PLANS

L. S. Homem de Mello
Dept. of Electrical and Computer Eng.
Carnegie Mellon University
Pittsburgh PA 15213-3890

A. C. Sanderson
Electrical, Computer, and Systems Eng. Dept.
Rensselaer Polytechnic Institute
Troy NY 12180-3590

Abstract - Maintenance and repair are increasingly important functions in many applications of assembly robots in hazardous environments such as space, undersea, and nuclear power plants. Since it is impossible to predict which repair will be needed, robots for these applications must have the intelligence to figure out the sequence of tasks required to execute any maintenance job that may become necessary. Previous work [4] introduced the AND/OR graph representation of assembly plans and showed how it forms the basis for efficient planning algorithms to be used within the highest level of control of an assembly workstation [1]. This paper shows how a simple modification in the set of goal nodes of the AND/OR graph allows its use in planning repairs such as the replacement of a part, or a subassembly. It also shows an algorithm for the generation of all feasible sequences for disassembly and reassembly of parts that will achieve a repair. This approach has been demonstraded for an example of repair of space-based satellite equipment.

Maintenance and repair are increasingly important functions in many applications of assembly robots in hazardous environments such as space, undersea, and nuclear power plants.

Since it is impossible to predict which repair will be needed, robots for these applications must have the intelligence to figure out the sequence of tasks required to execute any maintenance job that may become necessary. This ability is also important to enable the robot to recover efficiently from unexpected events that may cause the execution of the repair to deviate from a planned course of action.

The needs for fast response times, for flexibility to adapt to changing conditions, and for the ability to recover from unexpected events render domain-independent planning methods inadequate to use in real-time applications such as maintenance and repair. Part of the limitation of existing domain-independent planners stems from the add-delete list representation of actions which drastically limits their use. Chapman [2] reviews the literature on domain-independent planning and discusses the limitations that stem from the representation used for actions. The limitation of existing domain-independent planners also stems from the formalisms used to represent plans. Early systems, such as HACKER [6], used ordered lists of actions to represent plans which left no flexibility for adaptation to the conditions at execution time. STRIPS [3] used a tabular form, called triangle table, which improves the capability to recover from errors, but only within one fixed order of the actions. NOAH [5] and other more recent systems (e.g. NONLIN [7] and SIPE [8]) represented plans as partially ordered sequence of actions.

In previous work [4], it has been shown that the partial order formalism cannot encompass all the orders of actions that will achieve the goals of a plan; therefore it allows only a limited amount of flexibility to adapt to changing conditions. In that work, the problem of planning the assembly of one product was viewed as a backward search in the space of all possible configurations of the set of parts that comprises the product. The backward search suggested a decomposable production system and led to an AND/OR graph representation of assembly plans which forms the basis for efficient planning algorithms to be used within the highest level of control of an assembly workstation [1]. Figure 1 shows one space-based satellite equipment, and figure 2 shows its corresponding AND/OR graph. Each node in that graph is labeled by a set of letters corresponding to the initials of the names of the parts that make up the node's subassembly; for example, the root node is labeled by F P M A B since it corresponds to the assembled equipment.

The useful feature of the AND/OR graph representation for the assembly problem is that it encompasses all possible assembly plans. Moreover, each assembly plan corresponds to a solution tree from the node corresponding to the final (assembled) product to the set of nodes corresponding to subassemblies that contain one part only; the nodes in this set are referred to as goal nodes. In real time, the intelligent robot can search the AND/OR graph for the solution tree that is most suitable to prevailing conditions. In particular, if hyperarcs are associated to resource requirements such as torque or gripper opening, the search may ignore those huperarcs whose requirements are beyond the capabilities of available resources.

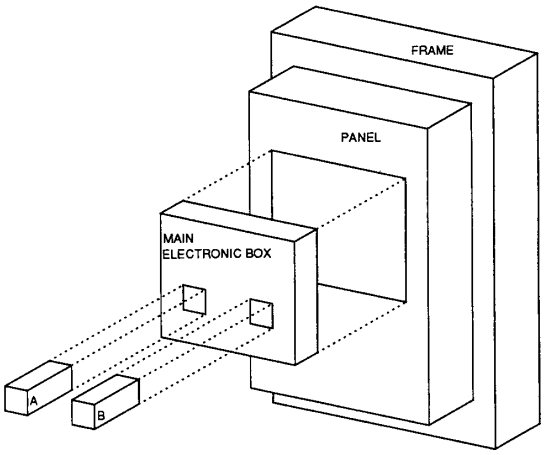

Figure 1: Space-based satellite equipment

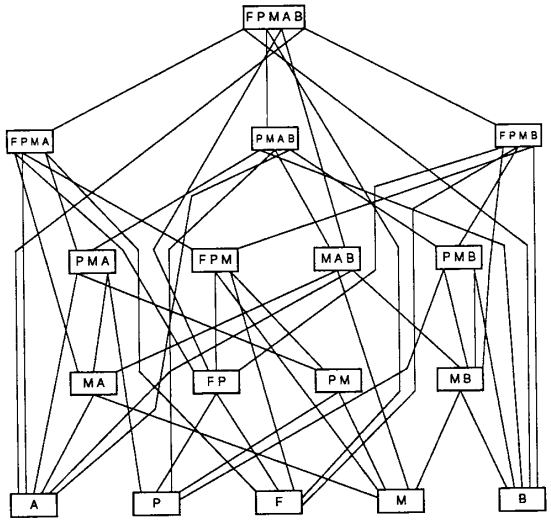

Figure 2: AND/OR graph for the space-based satellite equipment

The same formalism can be used in maintenance and repair applications where robots are expected to be able to carry out commands such as

`replace` *<faulty-part>* `in` *<product>*

autonomously.

The replace command is carried out in three phases:

1. a disassembly phase which consists of a a sequence of disassembly tasks that releases the faulty part;
2. a replacement phase in which the faulty part is replaced; and
3. a reassembly phase which consists of a sequence of assembly tasks that reassembles the product.

Both disassembly and reassembly sequences can be generated by searching solution trees in the AND/OR graph. Of course, one can use a sequence of tasks that disassembles the product completely in the disassembly phase, and a sequence of tasks that assembles the product from its individual parts in the reassembly phase. But very often there is no need to completely disassemble the product, and it is possible to withdraw the faulty part with just a few disassembly tasks. Such a shorter sequence of tasks can be generated by searching solution trees in the AND/OR graph provided that the set of goal nodes is changed. For the disassembly phase, the set of goal nodes includes the node whose subassembly is the faulty part, and all the nodes whose subassemblies do not contain the faulty part, since these subassemblies do not require further disassembly. Alternatively, the set of goal nodes for the disassembly phase may include nodes corresponding to subassemblies that do not contain the faulty part, as before, plus nodes whose subassemblies contain the faulty part but replacements for all the parts that make up these node's subassembly are available. The set of goal nodes for the reassembly phase includes the tip nodes of the disassembly solution tree, except those that contain the faulty part, plus the nodes that contain one part only and a replacement is available.

The algorithm *REPLACE-PART* for generating all complete sequences of tasks to replace a part with a given set of resources is shown in figure 3. It first generates all sequences of tasks for withdrawing the faulty part. For each of these sequences, it generates all sequences of tasks for reassembling the product. The algorithm operation consists in set the goal nodes for each type of search and actually searching the AND/OR graph using algorithm *SOLVE-ASSEMBLY* shown in figure 4. Algorithm *SOLVE-ASSEMBLY* takes as input a node corresponding to a subassembly, checks whether it is a goal node, and, if not, spawns descendent feasible tasks to be solved by algorithm *SOLVE-TASK* shown in figure 5. Algorithm *SOLVE-TASK*, in turn, takes as input one task, and hands back to *SOLVE-ASSEMBLY* all the solution trees in which the hyperarc leaving the root node corresponds to that task. These algorithms have been implemented in COMMON LISP and have been tested in planning repair of space-based equipment.

In practice, one needs to select one sequence among all those that are feasible. This selection involves a trade-off between speed of plan generation, and the *quality* of the plan generated, where the quality is measured by some criterion used to compare plans. Further exploratory research on criteria to compare plans and on search algorithms is currently under way. The important fact is that the AND/OR graph representation of assembly plans constitutes a compact representation of all possible sequences of disassembly and reassembly of parts that will achieve a repair; therefore it allows one to explore the space of all possible plans.

```
procedure REPLACE-PART(part product resources)
begin
sequences ← NIL
set data structures used in testing whether a node is goal for searching
    disassembly sequences
dseq ← SOLVE-ASSEMBLY(product DISASSEMBLY resources)
while dseq is not empty do
    begin
    set data structures used in testing whether a node is goal for searching
        reassembly sequences that follow the first sequence in dseq
    aseq ← SOLVE-ASSEMBLY(product ASSEMBLY resources)
    while aseq is not empty do
        begin
        add to sequences the concatenatation of the first sequence in dseq
            and the first sequence in aseq
        withdraw the first sequence from aseq
        end
    withdraw first sequence from dseq
    end
return sequences
end
```

Figure 3: Algorithm *REPLACE-PART*

```
procedure SOLVE-ASSEMBLY(a m resources)
if GOALP(a) is TRUE return trivial solution tree containing only node a
task-list ← list of tasks that are descendants of a in mode m
n-solutions ← NIL
while task-list is not empty do
 begin
 task ← FIRST(task-list)
 if FEASIBLEP(task m resources) do
  begin
  t-solutions ← SOLVE-TASK(task m resources)
  while t-solutions is not empty do
   begin
   n-solutions ← UNION( n-solutions ( ( a task FIRST(t-solutions) ) ) )
   t-solutions ← TAIL(t-solutions)
   end
  end
 end
return n-solutions
end
```

Figure 4: Algorithm *SOLVE-ASSEMBLY*

```
procedure SOLVE-TASK(t m resources)
node-list ← list of nodes that are descendants of t
list-of-node-solutions ← NIL
while node-list is not empty do
    begin
    node ← FIRST(node-list)
    node-list ← TAIL(node-list)
    node-solutions ← SOLVE-ASSEMBLY(node m resources)
    if node-solutions is NIL return NIL
    list-of-node-solutions ← UNION(list-of-node-solutions node-solutions)
    end
return MULTIPLY(list-of-node-solutions)
end
```

Figure 5: Algorithm *SOLVE-TASK*

Acknowledgements

This research was supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (Brazil); in part by Jet Propulsion Laboratory, California Institute of Technology; and in part by The Robotics Institute, Carnegie Mellon University. Part of this research was conducted in the summer of 1987 while L. S. Homem de Mello was at the Machine Intelligence Group of the Electronics and Control Division, at Jet Propulsion Laboratory. He thanks the members of the Group for helpful discussion; the satellite example was supplied by Wayne Zimmerman.

References

[1] A. J. Barbera. *An Architecture for a Robot Hierarchical Control System.* U.S. Government Printing Office, Washington D.C., 1977. National Bureau of Standards. Special Publication 500-23.

[2] D. Chapman. Planning for Conjunctive Goals. *Artificial Intelligence* 32(3):333-377, July, 1987.

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

[4] L. S. Homem de Mello and A.C. Sanderson. AND/OR Graph Representation of Assembly Plans. In *AAAI-86 Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1113-1119. American Association for Artificial Intelligence, Morgan Kaufmann Publishers, 1986.

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

[6] G. J. Sussman. *A Computer Model of Skill Acquisition.* Elsevier, 1975.

[7] A. Tate. Generating Project Networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888-893. August, 1977.

[8] D. E. Wilkins. Domain-independent Planning: Representation and Plan Generation. *Artificial Intelligence* 22(3):269-301, April, 1984.