SPLICING PLANS TO ACHIEVE MISORDERED GOALSs

Steven A, Vere
Information Systems Division
Jet Propulsion Laboratory
Pasadena, California 91109

ABSTRACT

Most parallel planners are sensitive to the order in which goals and activity preconditions are specified. A "wrong" ordering can easily cause a solution to be missed. Permuting goals and preconditions on failure in hopes of finding a soluble order is in general computationally unacceptable. Plan splicing is a solution to this problem. Splicing is a violent conflict resolution procedure which involves the cutting of assertion dependencies, recursive demotion or excision of selected activities around the cut, and reinsertion of deachieved goals back into the middle of the planner's goal stack so that they can be replanned later to mend the plan around the splice. In a temporal planner, after an excision it is further necessary to relieve the "temporal stress" induced on surviving activities by the activities which were excised. This is an important capability for two reasons: first, because the order of achievement can of course not always be known in advance, and secondly because it is desirable to be able to present goals in priority order.

I INTRODUCTION

AI planners are beginning to reach the maturity required for real world applications. DEVISER I [Vere, 1983] has been successfully applied to generating command sequences for the Voyager spacecraft. DEVISER I is on the evolutionary path of NOAH [Sacerdoti, 1977] and NONLIN [Tate, 1977]. NOAH was the first planner to deal with parallel activities. NONLIN is an improved parallel planner which is able to recover from bad decisions and which implements a more sophisticated treatment of goal protection. A recent version with a consumable resource management capability [Tate and Whiter, 1984] has been demonstrated on a naval replenishment problem. DEVISER I extended the mechanisms of the 1977 NONLIN to permit planning in time, with arbitrary time constraints on activities, preconditions, and goals. It also handles events, scheduled events, and inferences in a uniform manner. Temporal planning has also attracted the attention of a number of other researchers within the last few years [Allen and Koomen, 1983; Cheeseman, 1983; Dean, 1984; McDermott, 1982]. However, time is largely orthogonal to the issue of goal order and splicing, which is the focus of this paper. SIPE [Wilkins, 1984] is another contemporary planner derived from NOAH which has special resource handling features. It has been applied to planning activities on an aircraft carrier.

A common difficulty experienced with most planners is the phenomenon I will call goal protection deadlock, in which the achievement and subsequent protection of an earlier goal can block the achievement of a later goal [Dreussi, 1982, pg. 59]. For example, suppose a robot is in front of a closed door leading into a room. It's goals are to be inside the room with the door closed. Goal protection deadlock is experienced if it first tries to achieve (DOOR CLOSED) (with a null action), and then attempts to achieve (IN ROBOT ROOM). To enter, it must plan to open the door, but this would violate the first goal, (DOOR CLOSED), which is achieved by the start state and is now protected. The problem is that the goals have been attempted in the "wrong" order. Previous parallel planners have avoided this problem by requiring goals to be presented to the planner in an order in which they can be achieved. This problem applies both to the original conjunctive goal set as well as to the ordering of the preconditions of an activity, which become subgoals during plan synthesis. In complex domains it may be impossible to know the correct order in which to attempt goals, and it is oomputationally unacceptable to try every possible permutation of goals and preconditions.

There are reasons other than ignorance for wanting a planner to be Insensitive to goal ordering. A greedy person may try to give a planner many more goals than are logically achievable, due to time or resource limitations. If goal deadlook can be avoided, it then becomes attractive to order the goals by decreasing priority. The most Important goals can be planned first and allowed first claim to the finite resources. If possible, activities to achieve goals lower on the list are fitted into the plan later. Otherwise the lower priority goal is discarded. In this way the planner is able to generate a partial solution for insoluble goal sets

*This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, and sponsored by the National Aeronautics and Space Administration under contract No. NAS-918.

instead of just just giving up.

It is for these two reasons, to avoid goal deadlock and to allow goal prioritization and discard, that I have investigated and implemented splicing in a new version of my planner designated DEVISER III.

For tutorial purposes, the plan splicing process will be illustrated in this paper on blocksworld and abstract examples. However, splicing has been applied in practice on large plans constructed with a very detailed knowledge base for the Voyager spacecraft consisting of 1800 lines and describing about 140 different actions, inferences, and events.

II PLAN SPLICING

Plan splicing may be regarded as a new variety of conflict resolution in a parallel planner. Figure 1 illustrates a prototypical conflict situation. During parallel plan synthesis, a conflict is said to occur when two parallel nodes, such as Node A and Node B, assert contradictory literals, represented by P and "P. (You are cautioned not to confuse this usage of "conflict" with the completely unrelated notion of "conflict sets" in forward chaining production rule systems such as OPS5). The dashed lines show that Node C depends on the assertion P from Node A, and Node D depends on the assertion "P of Node B. These will be called assertion dependencies. They indicate that the truth of the assertion must be protected in the region of the plan between the two nodes. Tate calls these dependency relationships the goal structure. Of course there may be many nodes like Node C depending on P in Node A and many nodes like Node D.

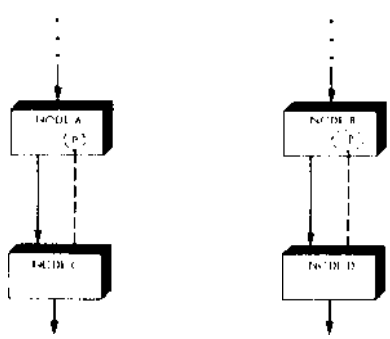

Figure 1. A Conflict Situation in a Plan

There are two possible conflict resolutions, as performed in NONLIN (and DEVISER I): either D must be ordered before A, or C must be ordered before B, so that the assertion dependencies are respected and preserved. This is illustrated in Figures 2a and 2b. Because assertion dependencies are never violated, neither of these two conflict resolutions can overcome the problem of goal protection deadlock.

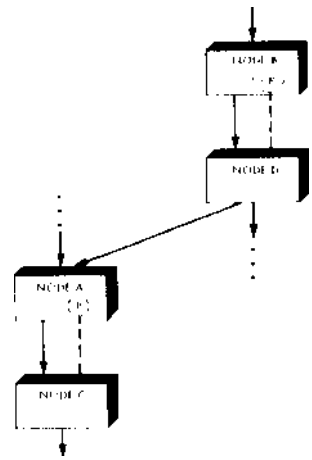

Figure 2a. One Non-Violent Conflict Resolution

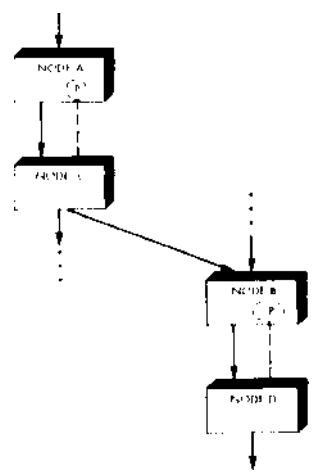

Figure 2b. The Other Non-Violent Conflict Resolution

Plan splicing is a violent form of conflict resolution which is only tried as a last resort, after nonviolent conflict resolution has failed. There are two possible splicings for every

conflict, just as there are two possible ordering resolutions. Figures 3a and 3b illustrate these two alternatives. Because of symmetry! only Figure 3a will be discussed. There the assertion dependency between A and C has been cut, and B has been ordered between A and C. Simultaneously, C has been "demoted." Node demotion is a generic term for an involved process whose details depend on whether C is a phantom or an activity. (A phantom node is a null action which signifies that a precondition has "already" been achieved above in the plan; an activity node is everything else—an action, an inference, or a (forward chaining) event). Thus splioing literally cuts the Gordian knot in a goal deadlook situation. The (reoursive) demotion prooess is responsible for ensuring that the plan will mend properly around the splice. At one extreme, demotion may involve simply changing a phantom node back to a goal. At the other extreme, demotion may trigger the erasure of large sections of the plan around and below the splice, with many goals being Inserted at a variety of positions within the goal staok for later replanning. Most of this paper is in fact concerned with the details of the demotion process.

Loosely speaking, one or more goals below the splice were achieved too early. Demotion sends these back into the goal stack in exactly the position which allows interfering later goals a chance. A solution is then obtained as though the goals had originally been attempted In the right order. In effect, interfering goals are dynamically reordered during plan synthesis, and this is accomplished without erasing any more of the existing plan than is logically necessary.

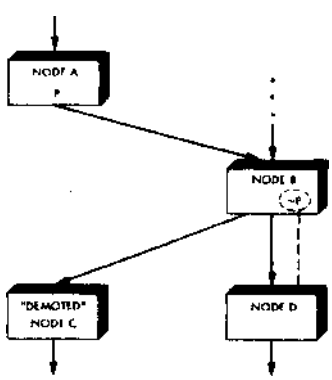

Figure 3a. One Conflict Resolution by Splicing

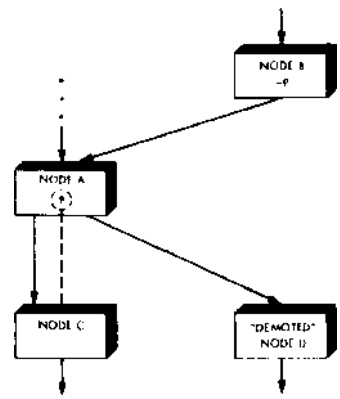

Figure 3b. The Other Possible Splice

The advantage of splioing is that it renders the planner insensitive to goal and subgoal ordering. However, a certain penalty is incurred. The size of the planner's search tree is enlarged, since there are now four possible resolutions to every conflict situation rather than two. In DEVISER this is mitigated by discarding the two splicing alternatives if one of the two nonviolent resolutions is successful. Otherwise, in backtracking the two splicing alternatives can lead into irrelevant sections of the search tree and waste time.

As already mentioned, the details of demotion depend on whether the demoted node is a phantom or an aotivity. These two oases will now be Investigated in turn.

III DEMOTING A PHANTOM NODE

Demotion of a phantom node is potentially the simplest case. If the assertion of the phantom is P, we simply oonvert the node baok to a goal, and enter P at the "appropriate place" in the planner's goal staok. The appropriate place is generally not the top of the goal stack, but somewhere in the middle. Details of the procedure for entering a goal baok in the goal staok will be presented later, in Section V.

This simple picture is complicated if a substitution was applied at the time the phantom node was created. For example, suppose the goal assertion was (ON .x C). In creating the phantom node, suppose that the substitution {B/.x} was applied to oause the goal assertion to match (ON B C) asserted by an earlier action. This means that to demote the phantom we must restore the goal assertion to its original unInstantlated form, (ON .x C).

A further, more serious complication is encountered if the substitution was in fact also applied to other literals in the plan. If we cannot somehow restore these literals to their uninstantiated state, existing parts of the plan may remain unnecessarily constrained, preventing us from finding a solution plan when in fact one exists. However, in DEVISER it is effectively impossible to deinstantiate an arbitrary literal. Suppose the instantiated literal is (P A A) and the substitution was {A/.x}. Restoring the literal to (P ,x .x) can be incorrect if the original literal was actually (P A ,x). The alternative of "remembering" precisely which terms were replaced for each substitution application was judged to be unacceptable. The approach I have adopted is to demote all plan nodes having a literal Instantiated by the substitution. This only requires "remembering", for each phantom, the list of nodes affected by the substitution (if any) applied when that phantom is created. Thus the demotion of one phantom may in turn call for the demotion of several additional "affected nodes", i.e., those affected by that phantom's substitution. If an affected node is also a phantom it is in turn treated in the manner just described. If the affected node is an activity, it is demoted as described in the next section.

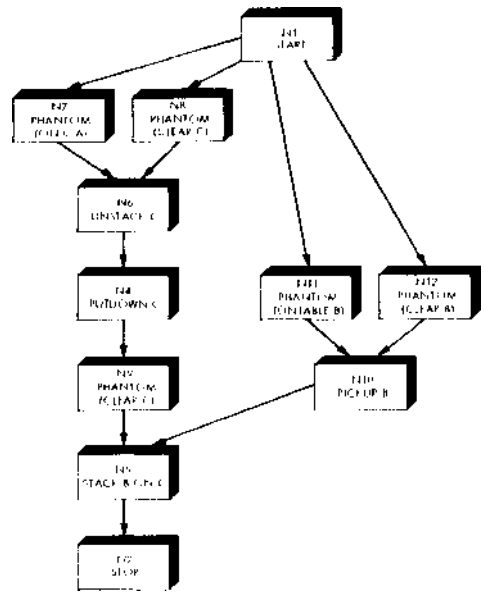

Figure 4. A Blocksworld Plan Illustrating Activity Pyramids

IV DEMOTING AN ACTIVITY NODE

As seen above, splicing may require the demotion of an activity node N back to a goal, either because N is Node C in Figure 3a, or because N was affected by the substitution of a demoted phantom. This in turn calls for the excision (erasure) of selected nodes above N in the plan. The nodes which must be excised are those which exist in the plan exclusively to satisfy preconditions of N. These will be called the activity pyramid above N.

Consider the blocksworld plan in Figure 4. Assertion dependencies are not shown to avoid cluttering the diagram. This plan was generated to achieve the goals (ONTABLE C) (ON B C), given the initial state (CLEAR C) (ON C A) (ONTABLE A) (CLEAR B) (ONTABLE B). N4 aohieves the first goal; N5 achieves the second. Suppose that in the course of planning to achieve other additional goals (not shown), we wish to do a splice and demote N5. Should we excise all the nodes above N5 in the plan? No. Nodes N6, N7, and N8 exist to enable the putdown action of N4 , and should be retained. We should excise only nodes N9, N10, N11, and N12. These were backchained into the plan to enable the stack action of N5. Nodes N9, N10, N11, and N12 constitute the activity pyramid above node N5, because they form an inverted pyramid of nodes backchained above N5, with N5 at the apex. Similarly, N6, N7, and N8 form the activity pyramid above N4. Consequently, to demote N5 we must first excise N9, N10, N11, and N12, and then convert N5 back to a goal and insert it into the goal stack.

One key aspect of activity demotion is then the excision of all nodes in the activity pyramid above the demoted node. Excision of a set of nodes involves erasure of the nodes from the plan by modification of the successor and predecessor lists of nodes which will remain, removal of the assertions of the excised nodes from the assertion database, and similar bookkeeping activities. Note that this erasure must be reversible, so that if the planner must backtrack, these excised nodes are "unerased" and restored to the plan. In addition, all phantom nodes outside the pyramid which depend on an assertion of an excised node must be demoted, because that assertion is going to disappear.

Having excised the activity pyramid above an activity node, that node is converted back to a goal. Side effect assertions of the activity are deleted, and any nodes below it which depend on one of these side effect assertions must be demoted too. The original goal assertion must be restored as the single assertion of the goal node, and the node must be inserted back into the goal stack as in the case of phantom demotion. Also, as in phantom demotion, other nodes affected by substitutions applied to the activity node or its pyramid must be demoted too.

One minor problem with splicing is that the planner may occasionally go into a search loop consisting of demotion, replanning, demotion, replanning, etc. In my implementation this was cured by keeping a record of demoted nodes and and the activity which caused the demotion. If the demotion subroutine is about to try to demote a node a second time for the same reason, this information causes the demotion to fail, breaking the loop and forcing the planner to backtrack.

Finally, in a temporal planner it is necessary to relieve the temporal stress induced on remaining nodes by those which have been excised. The situation is like a crowded elevator: when some people get off, those that remain can space themselves out more comfortably. In the same way, in a temporally crowded plan the start time windows of many nodes are compressed by the durations of adjacent activities. When some nodes are excised, the remaining nodes may be able to expand their start time windows. One possible approach would be to simply open the windows of all remaining nodes to the maximum interval, and then recompute all the start times based on the ordering and consecutivity constraints. However, this is unacceptable because excisions will be done frequently, and in practice only a small percentage of the nodes in a plan will be under stress from a set of excised nodes. A much more efficient technique is to follow stress chains from nodes on the boundary of the excised pyramid. Two sequential nodes, NA and NB, in a plan are temporally stressed if:

1. they are constrained to be consecutive (cf. [Vere, 1983]), or
2. the earliest finish time of NA equals the earliest start time of NB, or
3. the latest finish time of NA equals the latest start time of NA.

For cases 2 and 3 above it seems natural to say that their windows touch. Figure 5 illustrates a stress chain for an abstract plan. Suppose that node 1 is an activity node to be demoted. An activity pyramid with 1 at its apex is indicated by the dashed lines. Thus nodes 2, 3, 4, 5, and 6 are going to be excised. Nodes 1, 3, 4, 5, and 6 are on the boundary of the pyramid, i.e., they are adjacent to nodes not in the pyramid. The bold lines connect temporally stressed nodes leading away from the pyramid and beginning at boundary nodes. Note that a zig-zag pattern is possible, since nodes 15, 9, 8, 7, and 6 may be a chain of nodes with touching windows. The chain can continue in another direction since nodes 9 and 20, and 22 and 21 are assumed to be consecutive. The "C" label on the arcs indicates a consecutivity constraint. The subgraph connected by the bold lines is the stress chain in this diagram. It is only necessary to recompute start time windows for nodes in this stress chain. The windows of all other nodes outside the pyramid, such as 10, are not affected by the excision of the pyramid nodes. Of course it is possible for a stress chain to exist below the pyramid as well as above it, as in this example.

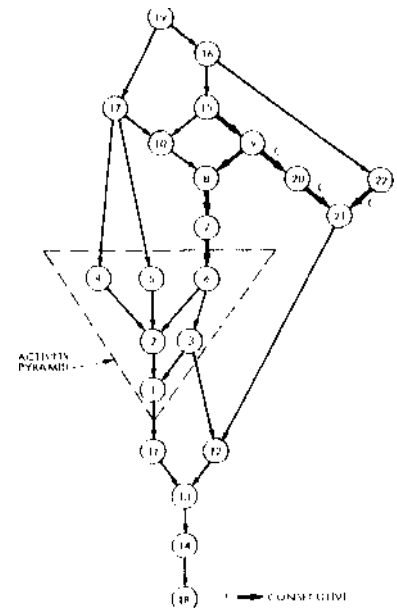

Figure 5. A Temporal Stress Chain

V INSERTING A DEACHIEVED GOAL INTO THE GOAL STACK

We have seen that demotion of both phantoms and activities leads to the creation (or, more accurately, the recreation) of one or more goals, which must then be inserted into the goal stack of the planner. How should such a goal be positioned relative to the existing goals in the stack? The answer is found in an analysis of why a node is demoted: it is demoted to "give another goal a chance." Stated differently, the planner decides that, in effect, the order of two goals must be reversed. Referring back to Figure 3a, Node C was demoted because of Node B. From the diagram we can infer that the goal of C was attempted first and then later the goal of B. If we are compelled to attempt a splice, it means that the goal of B should be "completely achieved" first before going back to work on the goal of C. By completely aohieved I mean that all backward chaining above B must be completed before going back to work on C. This can be ensured if the goal of C is inserted into the goal stack just below the lowest goal node in the activity pyramid above B.

Figure 6 illustrates with an example. It shows Figure 3a redrawn with an activity pyramid above B, as well as the associated goal stack. GC is the goal node created by demoting C. G1 and G2 are goal nodes in the pyramid above B. (Here the pyramid serves for analysis, and is not excised). G1 happens to be the lowest in the stack. If we insert GC just below G1 in the stack, then the planner will have completed the section of the plan above Node B before it starts to work on GC.

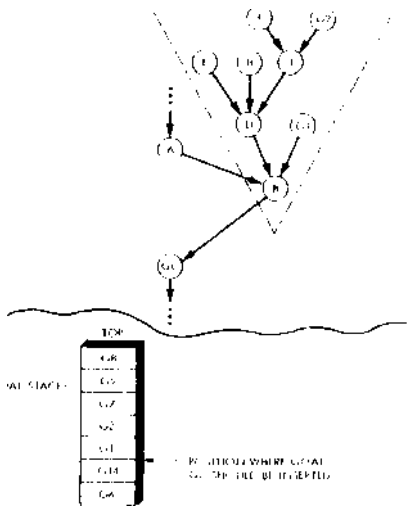

Figure 6. Goal Insertion During Demotion

VI SUMMARY

Planning is inherently sensitive to the order in which goals are accomplished, with failure the possible result of attempting goals in the wrong order. Splicing is a way, in effect, to reorder goals "on the fly" as planning proceeds. It is attempted only when normal conflict resolution fails. A portion of the plan which accomplishes an earlier goal is erased, and the earlier goal is put back into the planner's goal stack in a position which allows the later goal to be accomplished without interference. Through the stack mechanism, the planner's attention later returns to achievement of the demoted goal. The result is that actions to achieve the later goal are spliced into the middle of the plan, and some activities around the splice are replanned. This permits a solution to be found in goal protection deadlock situations where the planner would otherwise fail.

REFERENCES

[1] Allen, J. F. and J. A. Koomen, "Planning Using a Temporal World Model" In Proc. IJCAI-83. 741-747.

[2] Cheeseman, P. "A Representation of Time for Planning," Tech. Note 278, AI Center, SRI International, Menlo Park, California, Feb. 1983.

[3] Dean, T. "Planning and Temporal Reasoning under Uncertainty" In Proc. IEEE Workshop on Principles of Knowledge-Baaed system, Denver, Colorado, 1984.

[4] Dreussi, J. F. The Detection and Correction of Errors in Problem-solving Systems. Ph.D. dissertation, Univ. of Texas at Austin, 1982.

[5] McDermott, D. A. "A Temporal Logic for Reasoning About Processes and Plans." Cognitive Science, Vol. 6, 1982, 101-155.

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

[7] Tate, A. "Generating Project Networks" In Proc. IJCAI-77, 888-893.

[8] Tate, A. and A. M. Whiter, "Planning with Multiple Resource Constraints and an Application to a Naval Planning Problem" In Proc. 1st. Conf. £n. AI Applications, 198*4, 410-415.

[9] Vere, S. A. "Planning in Time: Windows and Durations for Activities and Goals." IEEE trans. Pattern Analysis and. Machine Intelligence, Vol. PAMI-5, No. 3, May 1983, 246-267.

[10] Wilkins, D. E. "Domain-independent Planning: Representation and Plan Generation." Artificial Intelligence, April 1984, 269-301.