Issues in Interleaved Planning and Execution

Scott D. Anderson
Spelman College, Atlanta, GA
anderson@spelman.edu

Abstract

This paper chronicles some issues that arose in attempting to implement two simple planners that interleave planning and execution. (For brevity, such planners will be called IPE planners.) The IPE planners are based on UCPOP (Barrett *et al.* 1993), a classic operator-based planner using goal-regression. Unfortunately, while the planners succeeded on very simple problems, they failed on problems as easy as the Sussman problem. This paper describes the two IPE algorithms and discusses why they failed.

Planning Technique

The point of interleaving planning and execution (IPE) is to gain time by doing an incomplete job of planning before executing the first action. An IPE planner that searches forward in situation space would stop before finding a complete sequence of actions from the initial situation to the goal. A goal-regression IPE planner would stop before finding a plan in which all goals are satisfied and all threats resolved. When the planner stops, the agent executes an action, and the planner resumes, somehow, in order to find other actions, until the goal is achieved.

The decision between forward search and goal regression planning is not clear-cut. A partial search by a forward-search, situation-space planner yields sequences of actions that are executable in the current situation. This is an obvious benefit, since the agent needs executable actions. A partial search in plan space by a goal-regression planner will not necessarily yield any executable actions. On the other hand, a goal-regression planner only considers actions that achieve subgoals, so any actions it finds are likely to be relevant to achieving the goal. Forward-search planners may consider actions that are executable but irrelevant. For example, a Blocksworld problem may also include a briefcase and operators to open and close the briefcase, which are executable in the current situation. The forward-search planner will consider actions of opening and closing the briefcase, while the goal-regression planner will not, since no preconditions of any sub-goals are achieved by opening or closing the briefcase.

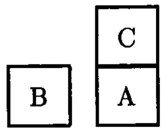

Figure 1: The Sussman Problem

The best resolution seems to be to use a goal-regression planner and search for long enough that an executable action is found. The action at least appears to be relevant, since it achieves some part of a subgoal. Nevertheless, an obvious problem arises, namely, that there is no easy way to determine how long it will take to find an executable action. Indeed, if the problem is sufficiently difficult, no executable action may ever be found. The unbounded nature of the search for an executable action makes an IPE planner based on goal regression unsuitable for hard-real-time problems (Stankovic 1988). An IPE planner based on forward search will at least find a promising action in a predictable amount of time, even if the action turns out to be a red herring. Of course, if the problem is unsolvable, the goal-regression planner will never stop searching, while the forward-search planner will just ceaselessly execute actions that never achieve the goal. It's not clear if either behavior is preferable. (Neither search needs to be infinite, of course, because a search limit can easily be imposed. Preferably, the planner would recognize unsolvable problems.)

Completeness

As planning algorithms, both forward search and goal regression can be complete—if a solution exists they will find it. Cutting off the search early, before a solution is found, and executing an action makes either planning technique incomplete. This can happen when the executed action leads to a world situation in which the problem is no longer solvable. For example, consider the Sussman problem, whose initial state is depicted in Figure 1. If there is a `vaporize` operator that destroys a block, that operator might well be used to achieve the subgoal `(CLEAR A)`, even though the problem is no longer solvable if block C no longer exists.

```
SEP-PLAN( situation, goal )
   while goal is not true in situation
       plan = create-initial-plan( situation, goal )
       plan = UCPOP( plan, plan-has-executable-action? )
       action = extract-action( plan )
       situation = execute( situation, action )
```

Figure 2: Pseudo-code for SEP-PLAN

The essential feature of **vaporize** is that it is not reversible—there is no way to undo its effects. As long as there are no irreversible operators, an IPE agent should be able to achieve any achievable goal. The resulting sequence of actions may be sub-optimal, in that there are extra actions or other inefficiencies, but the goal should still be achievable. In particular, the Sussman problem should be solvable using the standard Blocksworld operators, even by an IPE planner.

Interleaved Planning and Execution Using UCPOP

The basic design of the following IPE planners is to use UCPOP as a subroutine in a higher level algorithm, minimizing changes to the essential UCPOP planning algorithm. The standard UCPOP algorithm searches in plan space until it finds a plan in which there are no unachieved subgoals and no threats. In both IPE algorithms, the idea is to change the stopping UCPOP criteria so that UCPOP returns the first plan that has an executable action—one in which the action has no preconditions that are not true in the current situation. (Note that in this paper, the words "action" and "operator" will be used interchangeably.)

Separate Planning Problems

The first IPE algorithm is extremely simple: search until a plan with an executable action is found. (Clearly, if the problem is solvable, such a partial plan exists.) The action is then executed, resulting in a new world situation. The UCPOP planner is then called again, using the new world situation as the initial situation and using the same goal. Essentially, solving the initial problem is treated as solving a series of separate planning problems, hopefully converging on the goal. The pseudo-code in Figure 2 should clarify the algorithm, called SEP-PLAN. Note that UCPOP is viewed as taking two arguments—an initial plan and a predicate for stopping. In this case, the predicate is that the plan has an executable action. The initial plan is a trivial one with two dummy steps: the initial situation as the effects of the first dummy step and the goal as the preconditions for the second dummy step.

Because the successive planning problems are treated as separate problems, SEP-PLAN is vaguely reminiscent of Korf's Real-Time A* algorithm (Korf 1988), in that SEP-PLAN iteratively chooses an action based on an incomplete search and executes the action. Unfortunately, unlike RTA*, there is no way to guarantee being able to choose an action in a bounded amount of time, so this algorithm is not real-time in the sense that Korf's is.

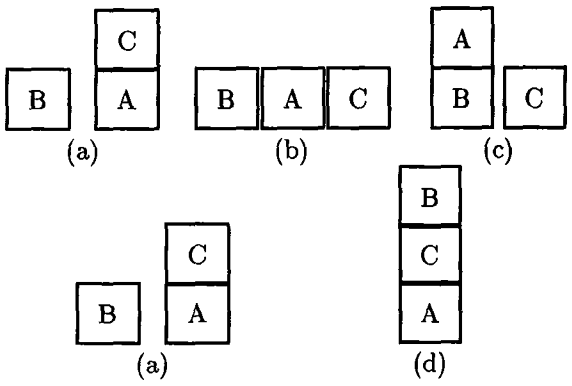

Figure 3: The Sussman Anomaly is that, starting from (a), the problem can't be solved by first achieving one goal and then the other. Trying to first achieve `(ON A B)` results in the upper sequence. Trying to first achieve `(ON B C)` results in the lower sequence.

This algorithm can solve trivial problems, such as building a short tower from blocks on the table, but fails on the Sussman problem. Let's look at this in detail. There are two possible subgoals to choose:

- `(ON A B)` If the planner chooses to achieve that subgoal first, it will find the action to move block C to the table.[1] This results in the world situation shown in Figure 3(b), which is the initial situation for the new planning problem.

  In the next planning problem, the goal is again `(ON A B)`, and is accomplished by putting A on B, resulting in Figure 3(c).

  In the third planning problem, the planner will address to subgoal `(ON B C)`, and it will find that it should remove A from B, resulting in Figure 3(b). Thus, the SEP-PLAN planner loops infinitely, repeatedly stacking and unstacking A from B.
- `(ON B C)` This is accomplished in one step, resulting in Figure 3(d).

[1]Or to move C to B, which is what UCPOP actually happens to find. This choice means the next action has to remove C from B, and it happens to move C back to A.

```
CONT-PLAN( situation, goal )
   plan = create-initial-plan( situation, goal )
   while goal is not true in situation
      plan = UCPOP( plan, plan-has-unexecuted-executable-action? )
      action = extract-action( plan )
      execute( action )
      mark_action_executed( plan, action )
      add_protection_interval( plan, action )
```

Figure 4: Pseudo-code for CONT-PLAN

In the next planning problem, the goal is (ON A B), and the planner will find that it should remove B from C. Thus, the planner loops infinitely, repeatedly stacking and unstacking B from C.

The Sussman problem is unsolvable by linear planners, and SEP-PLAN is a kind of linear planner, since it will try to achieve the goals one at a time (barring control rules that switch its attention, as is discussed in section). In retrospect, it's not surprising that SEP-PLAN should fail. More generally, SEP-PLAN fails because it has no long-term outlook; it myopically tries to achieve the first goal it can.

The solution, of course, is to do non-linear planning, so that the planner can interleave the actions necessary to solve both goals. Any algorithm to interleave planning and execution will have to more closely mimic standard non-linear planning.

Continued Planning Problem

The second attempt at an IPE algorithm again keeps the core planning algorithm the same as conventional, non-linear planning (specifically UCPOP) except that the agent executes actions during planning. Therefore, the planner must take into account the commitment to those actions. In other words, the planner continues its planning from where it left off, but it must be prevented from considering plans in which the previously executed actions either don't occur or occur in a different order from their execution order.

This is implemented with two changes to the planning system. First, the higher level algorithm starts UCPOP with a modification of whatever plan was returned at the previous iteration. Second, the plan is modified so that executed actions are marked as such, and a protection interval is added so that new steps are constrained to follow the executed steps. The pseudo-code in Figure 4 describes the algorithm, which is called CONT-PLAN because the planner continues from where it left off.

Note the differences between this algorithm and the SEP-PLAN algorithm. SEP-PLAN algorithm builds a trivial initial plan based on a changing initial situation. The CONT-PLAN algorithm starts with the plan it computed on the previous iteration. This change requires a change in the stopping predicate: since the initial plan has executable actions, UCPOP must now

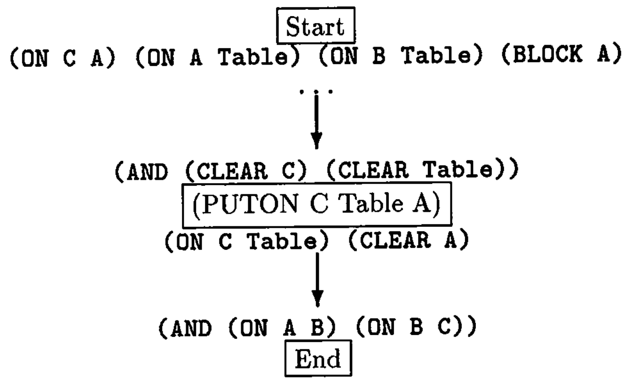
Start
(ON C A) (ON A Table) (ON B Table) (BLOCK A)
(AND (CLEAR C) (CLEAR Table))
(PUTON C Table A)
(ON C Table) (CLEAR A)
(AND (ON A B) (ON B C))
End

Figure 5: The initial plan for the second iteration of the Sussman planning problem. The planner found the PUTON action in the first iteration.

look for a plan with an *unexecuted* executable action. Finally, note that the action is executed only for side-effect; it is not necessary to change the variable describing the initial situation. Any changes to the world situation, made by executed actions, are taken into account by the planner because new actions must follow the executed actions, so that the post-conditions of those executed actions modify the initial situation.

Suppose that the chosen action is a poor one—either benign but useless or actually harmful—one that a normal non-linear planner would eventually reject. The IPE planning system would be forced to undo the action (assuming that's possible), and that "undo" action must follow the executed action.

In practice, using the Sussman problem, what happens is as follows:

- The planner chooses to try to achieve (ON B C). It quickly finds that, since both are clear in Figure 3(a), stacking B on C is executable.
- This action is returned and executed. The core planner, UCPOP, is run again, this time with the initial plan in Figure 5, which includes the executed action.
- Since the (ON B C) goal is accomplished, the planner tries to achieve (ON A B). That involves moving B onto A, which has the precondition of (CLEAR B).
- Clearing B requires moving blocks off of it, so a PUTON step is inserted to move C off of B. That step requires

C to be clear. There are two ways for C to be clear:

- Block C is clear in the initial state. Unfortunately, the first step (the one that was executed) is a threat to C being clear, and there is no way to solve that, so the planner reaches a dead end.
- Making C clear by moving blocks. That threatens the causal link between the step that was executed to put B on C and the goal of `(ON B C)`. There is no way to resolve that threat.

This last difficulty is exactly the subgoal-clobbers-brother-goal that was first noticed with the Sussman anomaly. The CONT-PLAN algorithm is unable to recover from these failures because it has already committed to a poor initial step. Since that step was useful, it's now protected from being undone. In practice, the planner searches infintely for an executable action. To solve this, the planner's handling of protection intervals would have to be changed—a substantial modification of the basic UCPOP algorithm.

What happens if the planner tries to accomplish `(ON A B)` first? In this case, the planning begins normally: it adds a step to put A on B, which requires A to be clear, so it adds a step to remove C from A. There are two ways to solve that: moving C to B and moving C to the table. These steps are both immediately executable and are ranked equally highly (using the default settings of UCPOP), so by pure ill luck, the planner moves C to B. The higher level system then executes that step.

Next, the system starts the planning over again with an initial plan that first moves C to B (clearing A), then puts A on B. A precondition of the first step—that B be clear—is not satisfied. The planner finds that to achieve the subgoal of having B clear, it needs to move C again. Any action that moves C will violate the protected link between the `(PUTON C B A)` step (the one the agent executed) and the `(PUTON A B ?)` step. Therefore, the action cannot be chosen, and there will be no way to achieve `(PUTON A B ?)`. Again the CONT-PLAN algorithm is stuck.

Summary

The SEP-PLAN algorithm fails because it gets into a loop, taking B off C and putting it back on again. The underlying reason is the essential problem with interleaved planning and execution: no long-term vision. The system goes for the short-term gain that prevents achieving the overall goal.

The CONT-PLAN algorithm fails because there is a protection interval between the initial state and the executed steps, and a protection interval between the step and the goal (to protect its achievement), so it becomes impossible to undo an executed step or to insert actions before it. Thus, the planner fails to find a plan. Fixing the behavior would require a radical rethinking of protection intervals; they are, after all, still useful.

Interleaved Planning and Execution in Prodigy

Prodigy can solve the Sussman problem and execute actions along the way. How? First of all, it does not consider each goal in turn, like a linear planner, but works with the set of goals. This paves the way for control rules that can change the particular goal that Prodigy is attempting to achieve. By using a control rule that switches to the other goal at just the right time, Prodigy solves the Sussman problem while interleaving planning and execution.

Prodigy proceeds just as CONT-PLAN does, but when it finds a plan that includes the executable action to put A on B, it has a choice: to execute the action or to continue planning. (CONT-PLAN has no choice; it stops and returns the plan with the executable action.) When that choice arises, the following control rule (Blythe *et al.* 1992, p. 27) is triggered:

```
(control-rule avoid-apply-for-wrong-goal
  (IF (and (on-goal-stack (on <x> <y>))
           (candidate-goal (on <y> <z>))
           (applicable-operator (pickup <x>))))
  (THEN sub-goal))
```

In English, this control rule means that Prodigy knows that if the goals include (ON A B) and (ON B C), the action to put A on B should be deferred until (ON B C) is achieved. This is a clever and effective rule, but is domain-specific. Change the statement of the Sussman problem to use ABOVE instead of ON, and the problem is the same, yet the rule doesn't work. Prodigy may decide to execute the action instead of switching to the other subgoal.

There is nothing wrong with assuming that control knowledge is critical to efficiently solving problems. This particular control rule is perfectly sensible; yet, somehow it seems contrary to the spirit of the Sussman problem, which is that subgoals interact. The control rule fixes the interaction, but in a domain-dependent way.

No problem in the Blocksworld is difficult if the system knows that it should build a tower up from the bottom. After all, the "Penguins Can Make Cake" paper (Chapman 1989) showed that a purely reactive planner can build a very tall tower, as long as it knows to start from the bottom. No one ever claimed the Sussman problem was hard; it's interesting only because of the goal interaction.

What if Prodigy were deprived of this control rule, perhaps because it is solving a problem in some new domain; could it still solve the problem? I believe so, because Prodigy can be parameterized to allow *state loops*, which would permit the action of stacking B on C to be undone (thereby causing the initial situation to recur). Ordinarily, state loops are disallowed, to avoid inefficiencies in plans. However, I have not yet run this experiment.

Conclusion

The paper has described several approaches to interleaved planning and execution. Modified forward search was rejected because it seemed difficult to ensure that only relevant actions were considered. A domain with many operators and many objects could yield a forward search with a huge branching factor, with most of the search spent uselessly. Still, a forward search could be informed by a goal-regression search, so that the search focuses on promising operators and objects. This search algorithm might be like McDermott's (McDermott 1996).

Two attempts to use simple goal-regression were described, along with their failures. The first treated the whole problem as a series of independent plan-space searches using a changing initial situation. This failed because the planner is essentially linear, achieving one goal at a time. The second attempt executed actions as they were found, picking up the search where it left off, but pruning parts of the space that considered plans that do not start with the executed actions. This approach failed because the protection intervals prevented steps from being undone.

Finally, a successful approach by Prodigy was described, using either control rules to switch among goals or permitting state loops in the search space, thereby allowing actions to be reversed. The control rules may be too domain-dependent to be reliable for general problem solving. Allowing state loops is a general approach to goal interaction and reversing actions, but may introduce unacceptable inefficiencies in the planning. More research should go into the role that state loops play in the completeness and efficiency of interleaved planning and execution.

References

Barrett, A.; Golden, K.; Penberthy, J. S.; and Weld, D. S. 1993. UCPOP user's manual (version 2.0). Technical Report 93-09-6, Department of Computer Science and Engineering, University of Washington.

Blythe, J.; Etzioni, O.; Gill, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; Minton, S.; Pérez, A.; Reilly, S.; Veloso, M.; and Wang, X. 1992. Prodigy4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Carnegie Mellon University, School of Computer Science.

Chapman, D. 1989. Penguins can make cake. *AI Magazine* 10(4):45–50.

Korf, R. E. 1988. Real-time heuristic search: New results. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, volume 1, 139–144. American Association for Artificial Intelligence.

McDermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, 142–149. American Association for Artificial Intelligence.

Stankovic, J. A. 1988. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer* 21(10):10–19.