User-Guided Interleaving of Planning and Execution

Peter Stone **Manuela Veloso**
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891
{pstone,veloso}@cs.cmu.edu

Abstract

We explore two advantages of interleaving execution with planning. First, the overall planning and execution time can be reduced. Second, information from the environment can be incorporated into the planner's knowledge of the world. We extend the PRODIGY planner to handle execution as prompted by the user and to incorporate information that results from this execution. Such information can either arise automatically or can be input by the user. Finally, we briefly discuss ways to help the user determine potentially useful or needed points for execution during planning.

Introduction

Planners do not generally have the ability to actually manipulate and sense the real world. Instead, they receive domain and problem descriptions from a human user and return a sequence of actions which the user must execute. Ideally, planners have enough time and information to reach a complete solution before the user must begin executing the plan. However, this is not the case in either time-critical or incompletely-defined situations. On the one hand, the user may *want* to begin execution while the planner is still planning to improve the combined planning and execution time. On the other hand, the user may *need* to start execution to gather information necessary to continue planning.

It is a well-recognized complex problem to decide when and how to interleave execution and planning (Ambros-Ingerson & Steel 1988; McCalla & Reid 1982; Morgenstern 1987). In this initial work towards addressing this general issue, we assume that the planner does not autonomously determine when to execute a plan step: the user decides. We present a planning and execution algorithm which we implemented as an extension to the PRODIGY planning algorithm (Carbonell *et al.* 1992). The algorithm allows the user to execute planning steps either for efficiency reasons or for information gathering purposes. The planner is extended to accommodate the user-selected execution in three ways. First, it is prepared to recommend actions for execution during the planning process. Second, it keeps track of which actions have been executed (as indicated by the user), so that it can produce a final plan accordingly. Third, it is able to incorporate new information from execution into its planning process.

In the paper, we identify the tradeoffs involved in interleaving planning and execution in terms of needed information gathering, and savings in overall running time. To conclude, we briefly discuss ways to help the user determine potentially useful or needed points for execution during planning.

Implications of Execution in Planning

Planning, independently of which planning algorithm is used, proceeds incrementally. New plan steps are introduced into the plan one at a time and choices and commitments are made as to which steps to select along the way. If a planning algorithm is to be complete, then all the choices must have a chance to be visited. Hence, no commitment made during the planning process should eliminate a portion of the search space that could possibly yield a solution: every significant choice is reversible through backtracking. Steps may be reordered when threats are found, different operators may be selected to achieve a particular goal, and different plan refinements may be explored.

Real execution of an action during the planning process removes some control from the planning algorithm: it can no longer backtrack over all of its commitments. In this sense, execution consists of real-world commitment. Real execution can also provide additional knowledge for the planning process. In this sense, execution consists of real-world sensing.

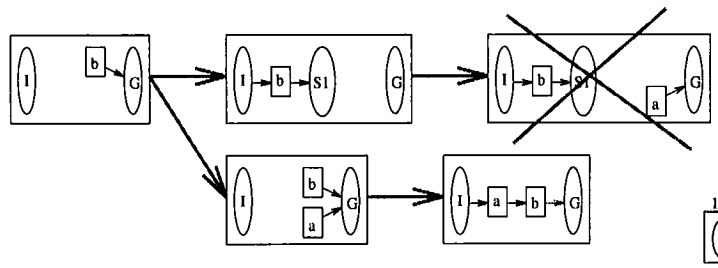
In general, real execution of plan steps while planning, i.e., before planning is completed, has multiple implications. In this paper, we focus primarily on two particular issues: the impact in terms of overall running time and quality of solutions of the combined planning and execution process; and the information gathering aspect, by which execution provides additional information to be used by the planner.

Quality of plans and time of plan execution

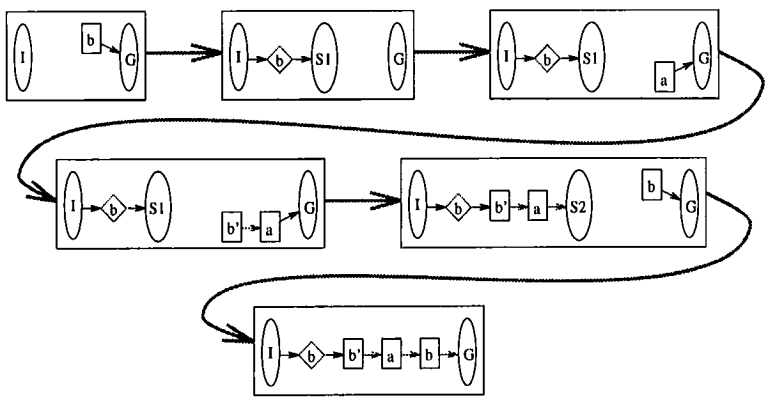
As mentioned above, one aspect of execution is real-world commitment as opposed to the exploration of alternatives at planning time. As an example, suppose that we need a flashlight and we have $10. We start planning which flashlight to buy. There are several flashlights of different prices that we could buy, so using a frugality heuristic, the planner plans to buy the cheapest one for $7. The planner

then finds out that, in order for the flashlight to work, it needs batteries. The planner plans to buy batteries, but the cheapest ones are $5. This leads to a failure, as the planner has only $3 left. At this point, the planner simply backtracks and plans to buy a $10 flashlight that has batteries included. But suppose planning were interleaved with execution: the user went ahead and bought the flashlight when the planner first decided that the cheapest one was the one to get. Instead of backtracking when the planner cannot buy batteries, it must now replan to acquire the lost resource, i.e. to get more money.

Figure 1 illustrates this particular trade-off in a general planning scenario using the representation of the search space in PRODIGY (Fink & Veloso 1994; Veloso *et al.* 1995 in press). The figure clearly illustrates the difference between the simulation of execution at planning time, which allows the planner to backtrack upon its choices, and the real execution of steps which triggers the need for replanning instead of simple backtracking.



(a) The planner can simulate execution. Operator b is applied to the state I and a new internal state, S1, is achieved. Planning fails as plan step a cannot be executed in state S1. The planner simply backtracks and succeeds.



(b) Operator b is executed; b' reverses its effects; Suboptimal solution results.

Figure 1: During planning, the planner can backtrack over its choices, as shown in (a). If planning is combined with real execution, replanning may be needed and new steps must be added to the plan, as shown in (b). Real executed steps are shown in diamonds. I is the initial state and G is the goal statement. Operator b' reverses the effects of b.

As shown in Figure 1(b) early execution may lead to solutions that are longer than an optimal solution (shown in Figure 1(a)). Interleaving planning and execution affects the global time of the combined planning and execution process. An optimal plan may be executed successfully concurrently with planning; but concurrent execution may also cause generation of unoptimal solutions in which conditions need to be reachieved.

Execution as a source of information

Time pressure is only one force that can cause the user to execute an action. Execution can also allow real observation of the effects of plan steps. In incompletely or incorrectly defined planning domains, execution is the best (and maybe the only) source of gathering accurate planning information. Execution adds knowledge from the world that can be used for future planning. Explicit requests for execution of plan steps can be triggered during the planning process, as designed by information-gathering planning operators (Etzioni *et al.* 1992; Pryor & Collins 1991). Opportunistic or informed execution may also be requested by a user during the planning process; the planning knowledge is then freshly updated for more informed future planning. In general, information gathered by execution during planning may open or prune alternatives for the planner. Figure 2 illustrates a planning scenario in which execution of an early step is needed to complete the plan.

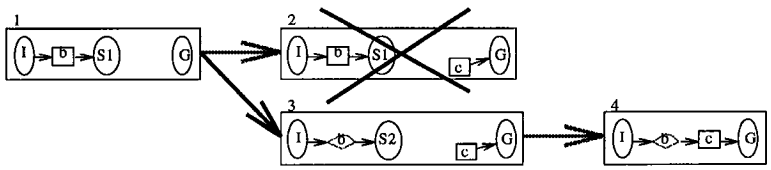


Figure 2: The planning process fails because, according to its internal knowledge of the world, the planner finds that a necessary plan step, namely the operator `c`, cannot be achieved in the current state `S1`. The user triggers execution of the plan step `b` which updates the planning knowledge to a new state `S2` where `c` can now be selected by the planner.

In general, information gathered via real execution makes the planner proceed in a more informed manner. The chances of needing to replan during the execution phase should decrease.

User-Guided Execution

What is so difficult about interleaving planning and execution? Since the planner loses control of the steps that it executes, it should try very hard to guarantee that these steps will not interact with future steps before committing irrevocably to them. The ability to give this guarantee is very difficult for a planner. However, the human user may have intuition or knowledge of when it is safe to execute. In addition, the information gathered at execution time may enable new planning choices or invalidate others. Although it is hard for a planner to predict the effects of executing an operator in the real world, a user may be better aware of the right moment to execute.

We implemented a framework where a user can interact with a planning algorithm to select the execution of plan steps. We extended the PRODIGY planning algorithm to incorporate real execution. The PRODIGY algorithm is well suited for interleaving planning and execution because it can reason about a simulated execution sequence (Fink & Veloso 1994; Stone, Veloso, & Blythe 1994). Thus real execution can proceed from a sequence of plan steps which are available for execution. Using its means-ends analysis strategy, early in the planning process, PRODIGY selects operators that reduce the differences between the current state and the goal statement. These plan step choices may be revised as planning proceeds, as long as they have not yet been executed. PRODIGY readily provides a set of plan steps to be executed rather early in its planning process. Other planners that do not use state information in their planning process would need to modify their algorithms to produce plan steps that are candidates to be executed at any given planning moment.

The planner queries the user every time it has a new operator to suggest for execution. It suggests operators that have been *applied*, but not *executed*. Applied operators are those whose execution has been simulated by the planner according to its internal state. Applying an operator by the planner produces a new internal world state. Executed operators are those selected for execution by the user from the set of applied ones. When an operator is executed, the internal state of the planner is updated with new information gathered. Table 1 sketches the extended planning algorithm combined with user-guided execution.

1. Terminate if the goal statement is satisfied in the current planning state. Return a list of plan steps indicating which have been executed.
2. Check if there is an *executable* plan step, i.e., a step which has been applied but not yet executed. If there is none, go to step 4.
3. Ask the user if this chosen plan step should be executed. If yes,
- Close all backtrack points corresponding to the executed operator.
- Incorporate newfound information from the execution into the planning state.
- Go to step 2.
4. Plan:
- Identify a goal that needs to be achieved.
- Add a new operator or link an existing plan step to achieve this goal.
- Go to step 1.

Table 1: The PRODIGY planning algorithm combined with user-guided real execution of plan steps.

The extension to the PRODIGY planner consisted mainly of adding the following functionality for when a plan step is executed: new state information is gathered and the internal state of the planner is updated; all of the choices made leading up to the application of that operator are finalized, i.e. they can no longer be backtracked over; and operators that would normally be discarded because they reverse the effects of previous operators are now considered, but as a last resort. In this way, when an operator is executed, all efforts are made to find a solution that uses that operator productively.

The interesting illustration of our technique would be a demo of the implemented algorithm, where the user can select which steps to be executed. As it is not possible to give a real demo in a written paper, we include a few running traces to exemplify the user-guided method, as developed so far.

Consider a domain where luggage is loaded into containers to be carried by an airplane. At planning time, there is no knowledge of the weight or size of each piece of luggage. Thus there is no basis on which to plan to select new containers after some amount of luggage has been loaded into a particular container. The planning operator "Load-Container" only checks if a container is available. Real execution of each loading step decreases the amount of available space in the container being loaded until it is full. Then the planner receives the information that the particular container is no longer available. If execution is not interleaved with planning, the planner plans to load all the luggage into the same container. In this case, the real execution, after planning is completed, necessarily leads to failure and the need to replan. The trace shown in Figure shows one example where execution is successfully interleaved with planning and another one where the lack of execution leads to uninformed planning.

We can run a variety of other execution examples that show other information gathering opportunities and the impact in the quality of solution and overall running time. We can also have the planner prompt the user for missing information that should have been discovered by the user during execution. For instance, in the example above, PRODIGY could ask the user for the weight of `obj1` when it is actually loaded and then determine whether or not there is any more room in `cont1` afterwards.

Discussion and Conclusion

In our current work, we are working towards connecting the algorithm to real execution agents, both software and robotic (Haigh, Shewchuk, & Veloso 1994). We would also like to propose useful execution breaking points to a completely automated system or to less-informed users. A domain-independent heuristic to select execution points should allow execution when there is reason to believe that either there will not be a need to backtrack over the resulting execution or that execution will provide additional information needed for future planning. In this case, allowing real execution will: save overall execution and planning time, as the plan starts being executed concurrently with planning; relieve the need for a completely-defined planning domain, as execution can provide information to refine other planning steps; and increase overall planning efficiency, as the planner is free from the need to keep track of a large number of open choices. After execution, the situation is equivalent to starting a new and potentially more informed planning problem.

```
In this situation there are three objects, obj1, obj2, obj3,
to be loaded and two containers available, cont1 and cont2;
cont1 has enough capacity to carry only object obj1.
```

```
;;User selects execution to gather additional information:
<c1> (run)
 ** <LOAD-CONTAINER OBJ1 CONT1> can be executed.
    Should I execute <LOAD-CONTAINER OBJ1 CONT1>? y
    Executed. Information added to the planning state.
 ** <LOAD-CONTAINER OBJ2 CONT2> can be executed.
    Should I execute <LOAD-CONTAINER OBJ2 CONT2>? n
 ** <LOAD-CONTAINER OBJ2 CONT2>, <LOAD-CONTAINER OBJ3 CONT2>
    can be executed -- independent steps.
    Should I execute <LOAD-CONTAINER OBJ2 CONT2>? n
    Should I execute <LOAD-CONTAINER OBJ3 CONT2>? n
Outcome of Planning:
     <LOAD-CONTAINER OBJ1 CONT1>  - executed.
     <LOAD-CONTAINER OBJ2 CONT2>
     <LOAD-CONTAINER OBJ3 CONT2>
Execution:
     <LOAD-CONTAINER OBJ2 CONT2>  - executed.
     <LOAD-CONTAINER OBJ3 CONT2>  - executed.
Success. Goals achieved after planning and real execution.
```

```
;;Execution is not interleaved with planning.
;;Replanning is needed.
<c1> (run)
 ** <LOAD-CONTAINER OBJ1 CONT1> can be executed.
    Should I execute <LOAD-CONTAINER OBJ1 CONT1>? no-more
Outcome of Planning:
     <LOAD-CONTAINER OBJ1 CONT1>
     <LOAD-CONTAINER OBJ2 CONT1>
     <LOAD-CONTAINER OBJ3 CONT1>
Execution:
     <LOAD-CONTAINER OBJ1 CONT1>  - executed.
     <LOAD-CONTAINER OBJ2 CONT1>  - failed.
Failure. Replan needed.
```

Figure 3: Interleaving execution and planning to gather information for informed planning.

The characteristics of the heuristic described remind us of the properties of Knoblock's abstraction hierarchies (Knoblock 1994), which can lead to no backtracking across refinement spaces. We can execute the refinement of each abstraction step incrementally, also with the hope that execution of the plan steps corresponding to the refinement of one abstraction level will gather information necessary for the refinement of the other abstraction steps as illustrated in Figure 4.

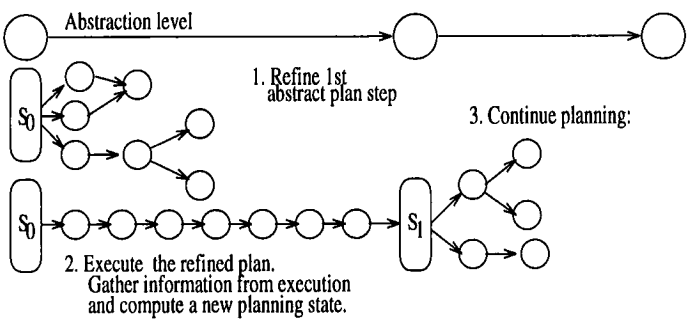


Figure 4: Execution break points guided by abstraction level information.

We discussed why interleaving planning and execution is hard, and presented the framework we created in which the user interacts with the planner. The user enables the planner to take advantage of execution to gather new planning information and to improve overall performance. We implemented the approach as an extension to the PRODIGY planner. Adding the ability to interleave planning and execution can increase the usefulness of general purpose planners.

Acknowledgements

This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Wright Laboratory or the U.S. Government.

References

Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution, and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 83–88.

Carbonell, J. G.; Blythe, J.; Etzioni, O.; Gil, 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.

Etzioni, O.; Hanks, S.; Weld, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*.

Fink, E., and Veloso, M. 1994. PRODIGY planning algorithm. Technical Report CMU-CS-94-123, School of Computer Science, Carnegie Mellon University.

Haigh, K. Z.; Shewchuk, J. R.; and Veloso, M. M. 1994. Route planning and learning from execution. In *Preprints of the AAAI 1994 Fall Symposium on Planning and Learning: On to Real Applications*.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68.

McCalla, G., and Reid, L. 1982. Plan creation, plan execution and knowledge acquisition in a dynamic microworld. *International Journal of Man Machine Studies* 16:189–208.

Morgenstern, L. 1987. Knowledge preconditions for actions and plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*.

Pryor, L., and Collins, G. 1991. Information gathering as a planning task. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 862–866. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Stone, P.; Veloso, M.; and Blythe, J. 1994. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on AI Planning Systems*, 164–169.

Veloso, M.; Carbonell, J.; Pérez, M. A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995, in press. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*.