User-Guided Interleaving of Planning and Execution

Peter Stone
pstone@cs.cmu.edu
tel: (412) 268-7123

Manuela Veloso
veloso@cs.cmu.edu
tel: (412) 268-8464

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891
fax: (412) 268-5576

April 3, 1995

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.

Keywords: Planning and reasoning on action and change, Planning and execution, Interactive Planning.

1 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 to be executed. Ideally, planners have enough time and information to reach a complete solution before execution must begin. 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 [1, 10, 11]. 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 current PRODIGY planning algorithm [2]. 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. This final extension also allows us to incorporate changes in the planning state due to extraneous events.

Several researchers have investigated the problem of interleaving planning and execution [5, 6, 8, 15]. The main focus of this research was on the definition and investigation of *reactive planning*, and on issues of efficient *replanning*. When deliberative planning was used to generate plans, most of the combined planning and execution approaches assumed that execution would be delayed as long as possible. During execution, elaborate methods for replanning are developed and invoked when execution alters the planning state in an unpredictable way. More recently, other researchers focused on producing *contingency plans* that try to enumerate different possible outcomes of actions at execution time. In addition, specific information gathering operators are added into the deliberative planning process to execute actions that probe the environment for planning information [3, 12, 13]. In our work, we learn from and build upon these different aspects of previous work. The main contributions of our work at this stage of development, as presented in this paper, are as follows. First, we recognize the possible benefits of early execution. Being aware of the difficulty of deciding correctly and generally when to start execution, we include the user in the planning loop, allowing the user to decide when execution should take place. Second, we extend the PRODIGY planning algorithm to handle user-guided execution,

suggesting possible break points for the user to request execution. Third, we provide a mechanism by which execution can be used to update the planning knowledge base.

2 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 deliberative 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.

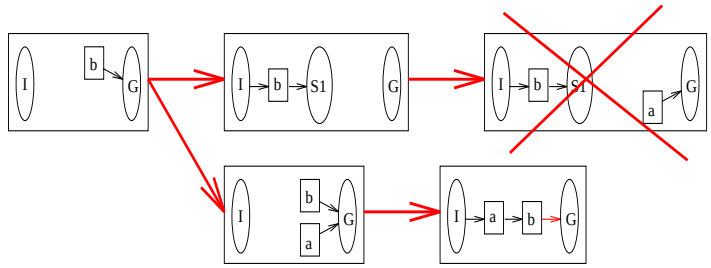
In general, real execution of plan steps while planning, i.e., before planning is completed, has multiple implications. In this paper, we focus primarily on two particular issues: the impact in terms of overall running time and quality of solutions of the combined planning and execution process; and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

2.1 Quality of plans and time of plan execution

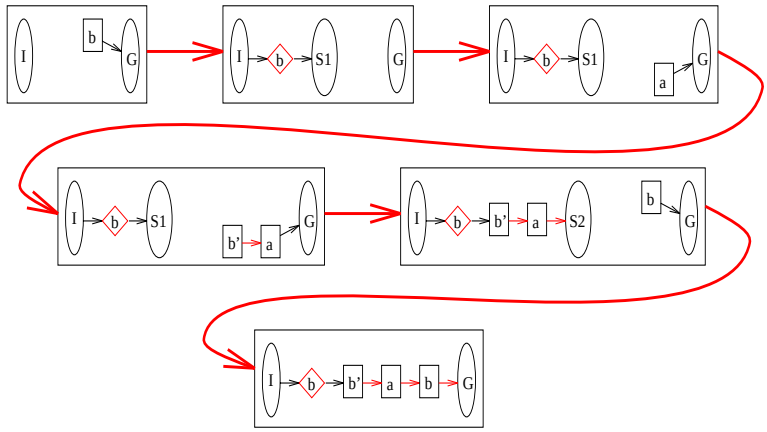
As mentioned above, one aspect of execution is real-world commitment as opposed to the exploration of alternatives at planning time. For example, consider the following example of planning with limited resources. Suppose a driver needs to plan a route from one location to another, perhaps including some necessary intermediate stops. Suppose further that it is late at night, so that gas stations are not open. When the vehicle moves, its fuel gets spent, and fuel in this situation is a limited resource. The intelligent agent starts planning for particular routes and destinations that would be reached. There are several alternatives. Suppose that the intelligent agent, in its deliberative mode, explores the alternative of moving along path A. It continues planning and later on, by analyzing further knowledge, it comes to the conclusion that path A leads to a dead end. This leads the planner to a failure, but the planner, in its deliberative reasoning, simply backtracks and plans to send the vehicles through a different path B.

Suppose now that planning is interleaved with execution: when path A is proposed as an option, the driver goes ahead and orders the vehicles to start moving along path A. When path A is found to lead to a dead end, the planner cannot control that choice any longer as the step is already under execution. The vehicle may not be able to return back from path A as its fuel may not be enough for the return path. The eager execution leads to a real failure.

There is therefore a clear tradeoff between the simulation of execution at deliberation 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. Figure 1 illustrates this particular trade-off in a general planning scenario using



(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.

the representation of the search space in PRODIGY [4, 16]. 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.

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 UN-optimal solutions in which conditions need to be reachieved. Figure 2 sketches the possible effects of interleaving planning with execution in terms of solution quality and overall running time.

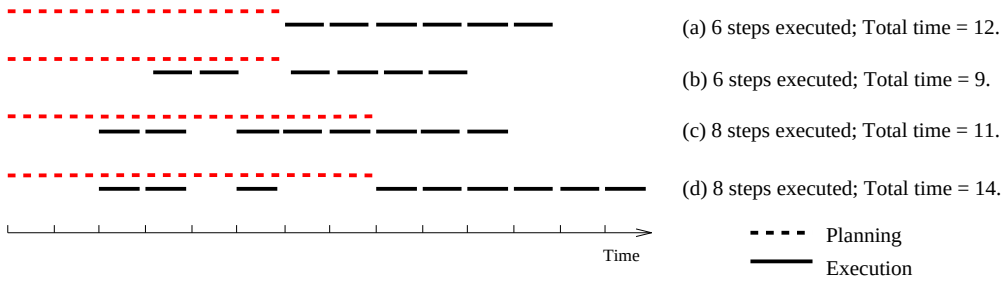


Figure 2: Interleaving planning and execution affects the global time of the combined planning and execution process. It may also affect the quality of the plans generated: in (b) the optimal plan (6-step long) is executed successfully in only 9 time steps; in (c) early execution leads into a non-optimal plan (8-step long) but combined execution and planning time is better than in (a); finally (d) shows the undesirable situation corresponding to a longer plan and delayed execution.

2.2 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 [3, 12]. 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 3 illustrates a planning scenario in which execution of an early step is needed to complete the plan.

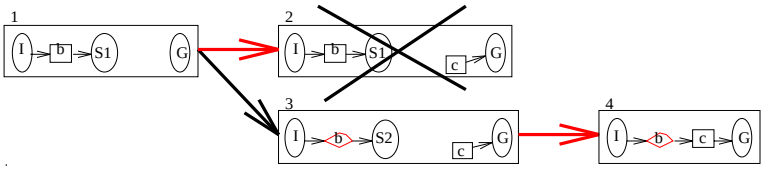


Figure 3: The planning process fails because, according to its internal knowledge of the world, the planner finds that a necessary plan step, namely the operator `c`, cannot be achieved in the current state `S1`. The user triggers execution of the plan step `b` which updates the planning knowledge to 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.

3 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 [4, 14]. 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 in a very simple task to exemplify the user-guided method as developed so far.

Consider a domain where luggage is loaded into containers of limited capacity 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", as shown below using PRODIGY representation language [2], only checks if a container is available.

```
(OPERATOR LOAD-CONTAINER
  (params <object> <container> <airport>)
  (preconds
   ((<object> OBJECT)
    (<container> CONTAINER)
    (<airport> AIRPORT))
   (and (at <object> <airport>)
(at <container> <airport>)
(available-container <container>)))
  (effects
   ()
   ((del (at <object> <airport>))
    (add (inside <object> <container>))
    (add (loaded <object>)))))
```

Real execution of each loading step decreases the amount of available space in the container being loaded until it is full. Then execution monitoring updates the planner's knowledge and 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 4 (slightly edited for presentation purposes) shows one example where execution is successfully interleaved with planning and another one where the lack of execution leads to uninformed planning.

In the first example of Figure 4, the user guides the planner to execute the first step as soon as possible. In doing so, the planner learns that `cont1` can not hold any more objects beyond `obj1`. This information is not available to the planner prior to execution: the state of the world is updated as a result of the execution. With this one step executed, the planner then knows to load the other objects in `cont2` rather than `cont1`. Indeed, the resulting plan can be successfully completed.

The motivation behind including the user in the loop, is that the user may have the sensitivity to know when it could be important to execute an operator. For example, the user may know that `cont1` can easily hold several objects and thus delay real loading for a while. After some time, the user may opt to execute in order to see if there is still room available. Note that in this case, the planner could not decide based on the operator being considered that execution is a good idea: the user may decide that the "Load-Container" operator should not be executed at first, but later that it should indeed be executed. The planner may be able to suggest which operators are occasionally useful to execute early, but ultimately the user decides.

In our modified version of the PRODIGY algorithm, the availability of `cont1` is deleted not by a planning operator, but by separate function that is invoked to represent real execution. Were our system hooked up to a robot that could manipulate

```
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:
<cl> (run)
 [Initial state: ... (available-container cont1)
             (available-container cont2)
                      (available-container cont3)]
 ** <LOAD-CONTAINER OBJ1 CONT1> can be executed.
    Should I execute <LOAD-CONTAINER OBJ1 CONT1>? y
    Executed. Information update to the planning state:
 [delete:            (available-container cont1)]
 ** <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.
<cl> (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 4: Interleaving execution and planning to gather information for informed planning.

and sense the world, then this change of state would not have to be modeled. Rather, it could be sensed by the robot. The planner's internal state that results from applying operators to it's initial state must incorporates such changes of state that are not the result of planning operators by always including them in its internal state representation and then reasoning from this representation. The successful result of such a process is illustrated in the first example of Figure 4.

On the other hand, when the user does not guide the planner to execute the first step, as in the second example of Figure 4, the planner continues on without the necessary information. Since nothing deletes the availability of `cont1`, the planner plans to load all three objects into this container. Then when execution is attempted, a failure results: planning must begin anew.

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.

4 Discussion and Conclusion

In our current work, we are working towards connecting the algorithm to real execution agents, both software and robotic [7]. 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.

The characteristics of the heuristic described remind us of the properties of Knoblock's abstraction hierarchies [9], 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 5.

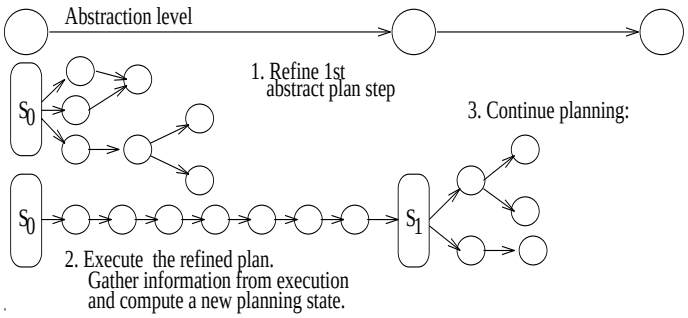


Figure 5: Execution break points guided by abstraction level information.

Another heuristic that the planner could use to suggest execution points is based on the past decisions of the user. By caching the operators that the user has decided to execute in the past, and by noticing at what point in the planning process the decision was made, the planner may be able to *learn* when it it may be useful to begin execution. We intend to explore this possibility further in the future.

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 incorporate this information into the planning state, thus improving overall performance. We implemented the approach as an extension to the PRODIGY planner. The trace shown in this paper does not make use of our current graphical user interface. We are currently also developing more sophisticated graphical representations of the planning alternative decisions and dependencies to better support the integration with the human user. Adding the ability to interleave planning and execution controlled by the user 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

[1] José Ambros-Ingerson and Sam Steel. Integrating planning, execution, and mon-

itoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 83–88, St. Paul, MN, 1988.

[2] Jaime G. Carbonell, Jim Blythe, Oren Etzioni, Yolanda Gil, Robert Joseph, Dan Kahn, Craig Knoblock, Steven Minton, Alicia Pérez, Scott Reilly, Manuela Veloso, and Xuemei Wang. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Department of Computer Science, Carnegie Mellon University, June 1992.

[3] Oren Etzioni, Steven Hanks, Daniel Weld, Neal Lesh, and Michael Williamson. An approach to planning with incomplete information. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, Boston, MA, 1992.

[4] Eugene Fink and Manuela Veloso. PRODIGY planning algorithm. Technical Report CMU-CS-94-123, School of Computer Science, Carnegie Mellon University, 1994.

[5] R. James Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, Seattle, WA, 1987.

[6] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 677–682, Seattle, WA, 1987.

[7] Karen Z. Haigh, Jonathan R. Shewchuk, and Manuela M. Veloso. Route planning and learning from execution. In *Preprints of the AAAI 1994 Fall Symposium on Planning and Learning: On to Real Applications*, New Orleans, LA, November 1994.

[8] James Hendler and J. Sanborn. Planning and reaction in dynamic domains. In *Proceedings of the DARPA Workshop on Planning*, Austin, TX, 1987.

[9] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68, 1994.

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

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

[12] Louise Pryor and Gregg Collins. Information gathering as a planning task. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 862–866, Hillsdale, NJ, 1991. Lawrence Erlbaum Associates, Inc.

[13] Louise Pryor and Gregg Collins. Cassandra: Planning for contingencies. Technical Report 41, The Institute for the Learning Sciences, Northwestern University, 1992.

[14] Peter Stone, Manuela Veloso, and Jim Blythe. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 164–169, June 1994.

[15] Austin Tate, James Hendler, and Mark Drummond. A review of AI planning techniques. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 26–49. Morgan Kaufmann, 1990.

[16] Manuela Veloso, Jaime Carbonell, M. Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), 1995.