In: Innovative Approaches to Planning, Scheduling and Control. pp 166-174, M. Kaufmann, San Mateo, Ca, 1990.

Deferred Planning and Sensor Use*

Duane Olawsky and **Maria Gini**
Computer Science Dept., University of Minnesota
4-192 EE/CSci Building
200 Union Street SE
Minneapolis, MN 55455
olawsky@umn-cs.cs.umn.edu
gini@umn-cs.cs.umn.edu

Abstract

Traditional approaches to task planning assume the planner has access to all of the world information needed to develop a complete, correct plan—a plan which can then be executed in its entirety by a robot. We consider problems where some crucial information is missing at plan time but can be obtained from sensors during execution. We discuss the solution of these problems through deferred planning (i.e., by deferring specific planning steps until more complete information is available and then restarting the planner). We also present early results of a comparative study of strategies for deciding which plan steps to defer.

1 Introduction

Traditional approaches to task planning assume that the planner has access to all of the world information needed to develop a complete, correct plan—a plan which can then be executed in its entirety by a robot. Unfortunately, this information about the world may not always be available at plan time. This is particularly true when we consider autonomous robots that must operate under general goals over extended periods in unpredicatable, and changing environments. When crucial information is missing at plan time, it may be impossible to find a complete plan without obtaining additional information. Fortunately, this information is often available at execution time through the use of sensors. The problem, then, is how to integrate execution time sensory data into the planning process which, in traditional approaches, is completed before execution begins.

The ability to integrate sensory data into the planning process is important. First, it provides greater robustness for autonomous robots. With this capability a robot could complete novel variations of tasks by realizing what information it knows and what it must find out through sensor use. It could then obtain the necessary information and perform the task. In this way the robot could work around its incomplete knowledge, filling in the gaps, to solve what would otherwise be an unsolvable problem. Second, this integration is helpful in robot recovery from execution errors and unexpected events. In these cases, it is likely that significant world information is missing (or at least in doubt). A robot controller could use this ability to collect the required information and then finish its task.

There are two reasons why it is difficult to integrate sensory data into the planning process. The first has already been mentioned—the planning process is traditionally completed before execution (and therefore sensor use) begins. The second difficulty is that the information obtained from sensors can have a dramatic effect on the shape of the plan. To make our discussion more concrete we will use the following tool box domain throughout this paper.

The robot is in a room with n tool boxes $t_1, t_2, \ldots t_n$, each containing wrenches and bolts of various sizes. The robot knows the initial locations of the wrenches and bolts. Bolts are identified by a unique name, and wrenches are identified by size (assume one wrench per size). The robot has been instructed to close and bolt one or more tool boxes with particular bolts. To perform each bolting operation, the robot must use a wrench of a size that matches the bolt. A sensor is available that can classify bolts by the size (e.g., a number from 1 to 10). For simplicity, the bolts sizes are indicated along the same scale as the wrench sizes. We also assume the robot has a tool belt into which it can put an unlimited number of bolts and wrenches.[1]

Figure 1 describes a sample problem instance. There

*This work has been funded by the NSF under grants NSF/DMC-8518735 and NSF/CCR-8715220.

[1]We are not concerned here with the arm-empty conditions used to define the blocks world. Our main goal in defining this domain is to study how sensor use can be interleaved with planning.

Initial State:
((at t)(bolt-not-inserted s)(bolt-not-inserted t)
(open s)(closed t)
(wrench-in-tbox 4 t)(wrench-in-tbox 5 s)
(bolt-in-tbox b_t t))

Goal State:
((bolted t b_t))

Figure 1: Sample Problem.

(Open-Tbox t)
(Get-Bolt b_t)
(Get-Wrench 4)
(Close-Tbox t)
(Insert-Bolt b_t t)
(Bolt t b_t)

Figure 2: Sample plan when b_t has size 4.

are two tool boxes, s and t. Box t is to be bolted with bolt b_t. Initially, the robot is at box t. There are two wrenches available, one of size 4 and another of size 5. The correct action sequence will vary depending upon which tool box contains the needed wrench, and this in turn depends on the size of b_t. The plan when the wrench is in s will differ from the plan when the wrench is in t. Sample action sequences are shown in Figures 2 and 3.

If the planner knows the size of b_t, it can find a complete plan before execution begins. Otherwise, the robot must use its sensors during execution to obtain the bolt size, and this information then determines the further actions that are necessary to achieve the end goals.

2 Why Use Conventional Planning

Planning is desirable in robotics because it attempts to map out future activities of the robot so that the robot avoids undesirable situations during plan execution. Although planning systems are known to suffer from computational complexity, with well crafted heuristics they have proven to be useful even for complex tasks [Wilkins, 1989].

A well recognized problem with planning is the inability of most planners to deal with the inexactness and noise of the real world. Several solutions have been proposed including the following:

- eliminating planning altogether in favor of reactive planning [Brooks, 1986] or situated systems [Agre and Chapman, 1987, Kaelbling, 1988],
- combining reactivity and planning [Georgeff and Lansky, 1987, Drummond, 1989, Hayes-Roth, 1987, Nilsson, 1989],
- preplanning for every contingency [Schoppers, 1987],
- verifying the executability of plans and adding sensing whenever needed to reduce the uncertainty [Brooks, 1982, Doyle, Atkinson and Doshi, 1986],
- interleaving planning with execution [Durfee and Lesser, 1989, Turney and Segre, 1989, Dean and Boddy, 1988, Hsu, 1990, McDermott, 1978].

Reactive systems, which are often proposed as a solution to the problems with conventional planning, suffer from being myopic. They tend to react to local changes, and have a short-term view of the problem they are trying to solve.

We are interested in exploring how to use conventional planning in domains in which the plan-time information is incomplete. This includes exploring strategies to maximize the chances of producing a plan that, despite incomplete knowledge, avoids premature actions.

(Open-Tbox t)
(Get-Bolt b_t)
(Close-Tbox t)
(Insert-Bolt b_t t)
(Goto s)
(Get-Wrench 5)
(Goto t)
(Bolt t b_t)

Figure 3: Sample plan when b_t has size 5.

3 Adapting Conventional Planning Techniques

The next question is how best to use conventional planning techniques to solve the problems we are considering. Is it necessary to extend these techniques in some way, or can we just define new operators at the correct level of abstraction that will allow a conventional planner to handle these problems? We contend that extensions are necessary. To demonstrate this, we attempt to define the required operators and point out the difficulties we encounter.

We must define the operators so that the planner need not be explicitly aware of the fact that sensors are being used. Thus, no sensor processes will be available to the planner. Assume that the size of some bolt B is unknown. We begin by collapsing two separate subgoals of the BOLT process with the properties (Boltsize B ?z) and (Have-Wrench ?z), into a single goal with property (Have-Wrench-for-Bolt B). In this way we hide the size of the bolt and the identity of the matching wrench. Let this new goal be achieved by the process (Get-Wrench-For-Bolt B). It is this process upon which we must focus. What effect does this process have on the world state. At the very least, after executing this process, the robot will have a wrench that it did not have before, and that wrench will no longer be in any tool box. (It is also likely that the robot will be in a different location.) The crucial observation is that the planner cannot know which wrench has been removed from a tool box. The identity of that wrench is determined entirely by execution-time sensory data that is not available to the planner. Thus, from the planner's perspective, (Get-Wrench-For-Bolt B) has nondeterministic effects, and this is problematic in conventional planners. If we allow such nondeterministic effects, the planner will have difficulty solving other goals that require obtaining a wrench since it no longer knows the location of all of the wrenches.

It thus appears necessary to extend conventional planning techniques to deal with the class of problems we are considering. There are three basic ways to do this:

1. Find a complete plan (or set of plans) that will work for all possible values of the relevant sensor reading. That is, plan for all contingencies (This is similar to universal planning [Schoppers, 1987] and to "tree plans" [Nilsson, 1989]).

2. Find a single complete plan based on an assumed value of the sensor reading. This plan will work (without modification) only if the assumption is correct.

3. Defer planning decisions that depend on sensor readings until those readings are available, then continue planning with the new information.

Which of these strategies is appropriate depends on external considerations such as the criticality of mistakes (i.e., Are they reversible? Is reversal costly?), the complexity of the domain, and the acceptability of suspending execution to do more planning.

3.1 The Three Approaches Compared

Planning all paths is often expensive and difficult and should be avoided if possible. If there are 20 different sizes of bolt, the planner might need to find a slightly different plan for each of the 20 possible sensor values. Matters are even worse in the likely event that more than one sensor reading is required. If the size of two different bolts must be determined by sensor readings from 20 possible values, there would be 400 combinations, each of which might correspond to a slightly different plan. The amount of planning grows exponentially in the number of readings that are needed. Although it might be possible to represent these 400 possible plans efficiently through the use of disjunctive nodes in the plan network, this does not really solve the problem. To do complete preplanning, the planner must still analyze the potential interactions (e.g., conflicts) that arise when any of the 400 possible combinations occurs. Despite the expense of this approach, there are still cases where it might be appropriate if it is computationally feasible:

- The same plan will be used many times with potentially different sensor values in each execution. Note that the same initial state must be satisfied in each use of the plan. In this case the cost of the plan is justified by its long-term usefulness.

- Time constraints during execution make it undesirable or impossible to do any execution-time planning (either deferred planning or replanning).

- The criticality of errors in the plan is so high that the cost of extra planning is outweighed by the cost of a mistake.

Unfortunately, even with all the planning effort associated with this approach, most execution-time errors and unexpected events are not anticipated. Unless these problems can be anticipated and handled in the plan, replanning may still be necessary. Due to the size and complexity of a plan in this approach, replanning to correct these problems could be difficult and costly.

Although approach (2) is less expensive, there is always a possibility that the assumptions made were incorrect, and the plan is therefore invalid. When this happens, replanning is necessary. Parts of the original plan will likely be discarded, and as a result, some planning effort is wasted. It is also possible that, due to the assumptions, some action is taken prematurely and must later be undone. If the premature action is irreversible, it might be impossible to solve the problem. Approach (2) is most appropriate when the following are all true:

- It is acceptable to have the robot stop during execution while replanning occurs.

- The criticality of plan errors is low. That is, actions are reversible, or the cost of failure is small (e.g., the robot can throw away an inexpensive part and start over with a new one).

- Some particular value for a sensor reading is more likely than any of the other possible values. In this case the planner has something upon which to base its guess. The odds are more in its favor.

One advantage of this approach over the deferred planning approach discussed below is that, when the planner guesses correctly, no execution-time planning is needed. However, if the planner guesses incorrectly, the time needed for replanning will probably be longer than the time needed to continue planning in a deferred planner since the replanner usually must remove parts of the original plan.

In the same vein, probabilistic reasoning has been proposed to reduce the complexity of planning. For instance, when expectations are available concerning how long propositions are likely to persist, probabilistic predictions can be made [Dean and Kanazawa, 1988]. Drummond [Drummond and Bresina, 1990] proposes an algorithm that maximizes the probability of satisfying a goal. The algorithm achieves a balance, in terms of robustness, between triangle tables [Fikes and Nilsson, 1971] and universal plans [Schoppers, 1987].

With the deferred planning approach, the planner avoids doing a lot of work that will later be discarded. Instead, it completes only those portions of the plan for which it has enough information at plan-time. Since the planner, in its initial phase, does not find a complete plan, there is the possibility that important dependencies and constraints in the plan will be missed. In this case some action might be taken prematurely which must later be undone. As with the replanning approach, if the premature action is irreversible, it might be impossible to solve the problem. Thus, care must be taken to detect these dependencies and constraints as early as possible before the robot has taken too many actions. Deferred planning is appropriate when the following are true:

- It is acceptable to have the robot stop during execution while planning continues.
- The criticality of plan errors is low. That is, actions are reversible, or the cost of failure is small.

It is the deferred planning approach that we are studying. The central problem for this approach is how to avoid premature actions that must be reversed (or even worse, that cannot be reversed). In Section 3.2 we will describe how we have implemented this approach, and integrated it with an execution simulator. In Section 3.3 we will give an example of how this system works. Section 4 outlines a number of strategies for deciding which plan goals to defer.

3.2 A Deferring Planner

The basis of our system is an agenda-controlled planner called BUMP (Basic University of Minnesota Planner). BUMP uses STRIPS-style operators [Fikes and Nilsson, 1971] to build a plan network consisting of goal nodes and process nodes. At present, BUMP is very basic in that it does not do hierarchical planning [Sacerdoti, 1974], nor does it use special methods to reason about resources [Wilkins, 1988]. It does maintain links from process nodes to goal nodes that record the purposes of each process node in the plan. The other major component is the EXECUTION CONTROLLER (EC). This controller is at the top-level in our system. It invokes BUMP to get solutions (plans) for particular problems, and it then controls the execution (in simulation) of the steps within those plans. It can also invoke the planner on a partially specified plan, asking BUMP to finish it. A system diagram is shown in Figure 4.

To solve the problems with which we are dealing, the BUMP plan must contain requests for sensor readings that obtain the information that the planner is missing. This is accomplished by adding a new type of process node to the planning system. A SENSOR PROCESS NODE constitutes an instruction to the execution controller (and hence the robot) to take a particular sensor reading at a particular point in the execution. We assume that the results of a sensor process can, at the planner's level of abstraction, be described by one or more logical predications.[2] This allows us to represent sensor processes in much the same way as non-sensor processes. That is, they are described by three lists of predications:

Add List — A list of predications describing the properties asserted as a result of the process. At least one of these will be the new information obtained by the sensor. This list can also specify side effects of the sensor process.

Delete List — A list of predications for properties denied as a result of the process. (This would likely include things that are changed in the world as side effects of sensing.)

Precondition List — Properties that must be true in order to use the sensor. This list will be used to generate the set-up actions for the sensor.

Since sensor processes are explicitly represented in the plan in much the same way as all other processes, their side effects as well as their set-up actions can be dealt with by BUMP.

A sensor process is used (like any other process node in a BUMP plan) to achieve one or more of the properties on its Add List. For example, to solve a goal node G for property (Boltsize B ?z), BUMP can insert a sensor process node (SENSE-BOLTSIZE B) into the plan. This sensor process node, when executed, will assert that the bolt B has some particular size as determined by the relevant sensor or sensors. The node could for example assert the property (Boltsize B #4). If some property matching (Boltsize B ?z) is already asserted, either in the initial state or by some process node that can occur before G, then the planner can solve G by performing the appropriate linking operation. No additional sensor process node is needed. Thus, the planner can easily recognize what information it already has available and what information must be obtained from sensors. Furthermore, it performs this reasoning through the same

[2]How sensor data is converted into such predications is a nontrivial problem that is beyond the scope of this paper. We do however assume that the conversion would be based upon some hierarchical representation of sensor data which allows that data to be represented at multiple levels of abstraction [Henderson and Shilcrat, 1944]. The planner would work at one of the highest levels.

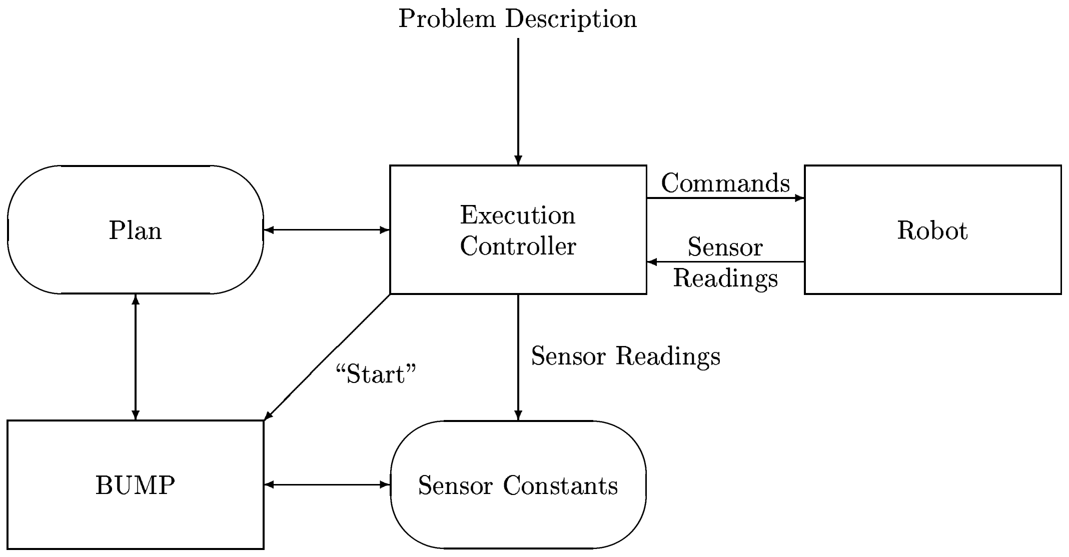

Figure 4: System Architecture.

mechanisms that determine whether to use a helpful interaction or an operator to solve a goal. While planning, BUMP uses special dummy constants in place of the values that will come from sensor readings. During the initial planning phase, the plan variable ?z from (Boltsize B ?z) will be bound to one of these constants. Any subsequently attempted plan goals that refer to one of these constants will be deferred until the executor has obtained the reading.

When all goals in the plan network have been either solved or deferred, BUMP returns the plan at its current state of completion. The execution controller then begins executing the partial plan, preferring sensor processes over other parallel processes since the former increase the robot's information about the world. This preference also extends to the set-up actions of sensor processes and to any other process nodes that are constrained to occur before a sensor process. This strategy is intended to obtain the sensory data at the earliest possible point in execution in order to avoid the problems caused by premature actions. Once a plan-requested sensor reading is obtained, BUMP is immediately restarted with the new information which it can use to make additional plan decisions. BUMP returns a new (perhaps still partial) plan to the executor. This cycle continues until all the necessary sensing has been done and BUMP has found a complete plan. The execution controller then executes the remainder of that plan.

3.3 An Example

To clarify this process we present an example. Consider the problem shown in Figure 1. A sample trace for this problem is shown in Figure 5. After an initial planning phase, BUMP halts with one Sense-Boltsize process in the plan and with the corresponding Have-Wrench goal deferred. EC begins the execution of the partial plan. The first two operations are required as preparatory steps for the third operation (Sense-Boltsize B). Since this solution is done in simulation, EC asks the user for a sensor reading. In this case, 4 is entered. BUMP is now restarted with this new information and this time produces a complete plan. The remainder of the operations are now executed and the task completed.

4 Deferral Strategies

The primary question in deferred planning is deciding what goals to defer. At the very least we want to defer the goals that are defined in terms of a sensor reading since we do not know the complete goal statement until the reading has been obtained. For example, we cannot formulate a plan or solve the goal (Have-Wrench ?s) until we know the value of ?s, the size of the wrench we must retrieve. We may not know this until we have used sensors to determine the size of some bolt.[3]

Is it advantageous to defer additional goals? That is, should we do as much preplanning as possible, or should we be more conservative? To study this question, we have defined two distinct deferral strategies:

[3]It should be noted that goals such as (Boltsize B ?s) are not treated in this way. When this goal is first encountered BUMP does not immediately know that a sensor reading is needed. Recall that when the bolt size is already known, a sensor process is not added to the plan. If the bolt size is unknown, a sensor process is added, ?s is bound to a sensor constant, and any further references to ?s will be recognized as a reference to a sensor reading.

```
<Initial Planning...>

Executing #<PROCESS19> (Open-Tbox T)
Executing #<PROCESS15> (Get-Bolt Bt)
Executing #<SENSOR-PROCESS11> (Sense-Boltsize Bt)
Enter the size of bolt Bt:  4

<More Planning...>

Executing #<PROCESS33> (Get-Wrench 4)
Executing #<PROCESS28> (Close-Tbox T)
Executing #<PROCESS23> (Insert-Bolt Bt T)
Executing #<PROCESS5> (Bolt T Bt)
```

Figure 5: Sample run for boltsize = 4.

Continue Elsewhere - In this strategy we defer only those goals that are defined in terms of data that must be obtained through a sensor reading. This strategy preplans as much as possible.

Stop and Execute - As soon as BUMP reaches a goal defined in terms of a sensor reading, it stops, deferring all remaining goals until the sensor reading has been obtained. This approach is "maximally conservative".

The Stop and Execute strategy does less preplanning than the Continue Elsewhere strategy. This has the disadvantage that crucial plan dependencies can be missed, and as a result, actions can be taken prematurely. On the other hand the planner will do significantly less planning with incomplete information. This tends to decrease the number of premature actions. Furthermore, Stop and Execute respects the order in which the planner wants to attack goals (which is, of course, independent of the order in which they are achieved during execution), but Continue Elsewhere does not. This is important for BUMP since it orders goals heuristically, and it is likely to be important for other planners as well.

We are currently conducting a study on the performance of these two strategies. We have conducted actual system tests for a set of 32 problems defined for a 2-box version of the tool box world. In this version there are two boxes s and t, and they are to be bolted shut with bolts b_s and b_t, respectively. Initially, the robot is at tool box s. Without loss of generality, we assume b_s has size 4 and b_t has size 5 (we can rename sizes to make this true), but the planner does not know this and must add sensor processes to the plan. The problem space is defined as in Figure 6. Note that since there are two boxes that must be closed, the planner must be careful not to bolt a box containing a wrench that will be needed later. We consider such a case to be a failure since significant actions are taken prematurely.[4]

[4]In this case the mistake is easily reversed. If the robot were welding the boxes shut, the recovery would be more difficult.

$$\begin{aligned}
\text{Initial State: } & (\text{Bolt-in-tbox } b_x\ x) \\
& \wedge \neg\ (\text{Bolt-in-tbox } b_y\ x) \\
& \wedge\ \ (\text{Wrench-in-tbox } 4\ x) \\
& \wedge\ \ (\text{Wrench-in-tbox } 5\ x) \\
& \wedge\ \ (\text{At } s)
\end{aligned}$$

$$\text{Goal State: } (\text{Bolted } x\ b_x)(\text{Bolted } y\ b_y)$$

Figure 7: 2-Box Study "Stop and Execute" Failure Cases.

$$\begin{aligned}
\text{Initial State: } & (\text{Bolt-in-tbox } b_s\ s) \\
& \wedge \neg\ (\text{Bolt-in-tbox } b_t\ s) \\
& \wedge\ \ (\text{Wrench-in-tbox } 5\ s) \\
& \wedge\ \ (\text{At } s) \\
\text{Goal State: } & (\text{Bolted } x\ b_x)(\text{Bolted } y\ b_y)
\end{aligned}$$

Figure 8: 2-Box Study "Continue Elsewhere" Failure Cases.

As a control we tested BUMP with complete information on the 32 problems. It produced a correct plan with no failures for every problem. The Stop and Execute strategy fails on two of the 32 cases, and Continue Elsewhere fails on four of them. The failure cases are described in Figures 7 and 8. The variables x and y range over the set $\{s, t\}$. The Continue Elsewhere failures occur because BUMP follows the rather natural heuristic of doing everything it can at its initial location before going somewhere else. In cases where b_t is not in s, the initial plan will instruct the robot to bolt s before sensing b_t. When the size of b_t is finally determined, its wrench may have already been bolted inside s. BUMP has ordered these actions prematurely and incorrectly

$$\begin{array}{rcl}
\text{goal-ordering} & \in & \{[(\text{bolted } s\ b_s)(\text{bolted } t\ b_t)], [(\text{bolted } t\ b_t)(\text{bolted } s\ b_s)]\} \\
b_s \text{ location} & \in & \{s, t\} \\
b_t \text{ location} & \in & \{s, t\} \\
\text{wrench 4 location} & \in & \{s, t\} \\
\text{wrench 5 location} & \in & \{s, t\}
\end{array}$$

Figure 6: 2-Box Study Problem Space.

$$\begin{array}{ll}
\text{Initial State:} & (\text{Bolt-in-tbox } b_s\ s) \\
 & \wedge \neg\ (\text{Bolt-in-tbox } b_t\ s) \\
 & \wedge \quad (\text{Wrench-in-tbox } 5\ s) \\
 & \wedge \quad (\text{At } s) \\
\text{Goal State:} & (\text{Bolted } t\ b_t)(\text{Bolted } s\ b_s)
\end{array}$$

Figure 9: 2-Box Study "Sense Before Closing" Failure Cases.

since insufficient information was available at the time.

We have also experimented with a modified Continue Elsewhere strategy called Sense Before Closing. In this strategy, the planner attempts to order all sensor processes before all Close-Tbox processes. (This ordering is not always possible because of other ordering constraints that may already be in the plan.) This strategy performed as well as Stop and Execute (see Figure 9 for the failure cases), however, it is not as general-purpose as the first two strategies. It is applicable only in domains where we want *all* sensor operations to precede all box closings. This would not be the case if the robot were requested to bolt a box, move to another room, and then do more sensing there.

5 Discussion

Interleaving of planning and execution has been used extensively. For instance, in the work of [Durfee and Lesser, 1989] the planner uses a blackboard based problem solver to abstract sensory data. This enables the planner to approximate the cost of developing potential partial solutions to achieve long-term goals. Detailed plans are created only for the immediate future using the sketch of the entire plan. By keeping the long-term goals the planner bases its short-term details on a long-term view.

Dean and Boddy [1988] propose a class of algorithms that they call "anytime" algorithms. These algorithms can be interrupted at any point, returning a partial plan. The quality of this plan depends upon the time used to compute it.

We have decided to investigate a more limited class of problems. We are interested in proposing and evaluating strategies to be used when some information is missing at planning time and needs to be obtained with sensors. In our approach planner decisions that depend on sensory information are deferred. As soon as sensory data become available the planning activity is resumed.

Doyle [Doyle, Atkinson and Doshi, 1986] uses sensors to verify the execution of a plan. The sensor requests are generated after the plan has been produced by examining the preconditions and postconditions of each action in the plan. Domain dependent verification operators map assertions to perception requests and expectations. Since perception requests are actions that could have preconditions, the planner is used to modify the original plan to guarantee that the preconditions are established. If the expectations are not satisfied by the perception the plan is repaired using predefined fixes. The entire process is done before executing the plan.

Our work has been inspired, in part, by the recent work of [Turney and Segre, 1989]. The system they present, SEPIA, alternates between improvising and planning. It addresses sensing errors, control errors, and modeling errors. Their example is a traveling salesperson problem with time constraints at every place to be visited. The set of rules suitable for firing contains rule instances whose preconditions and constraints have been met, but whose sensor requests have yet to be evaluated. Since sensing is assumed to be expensive, the system fires the rule instance with the fewest sensor requests first. The cost of a rule is proportional to the number of sensor requests it contains. The planner is interrupted when the cumulative cost exceeds its budget. The quality of the heuristic improvisation strategy has the most significant effect on the quality of the solution (both with the simple improvisation strategy and with SEPIA). This seems to suggest that it is more important to develop good heuristics than to develop a highly sophisticated planner.

Dean [1987] recognized the complexity of solving realistic planning problems and suggested heuristic approaches to decompose a task into independent subtasks that are easier to solve. He suggested using a library of strategies applicable to a set of tasks instead of a library of plans.

The need to plan with incomplete information raises interesting theoretical issues in finding an appropriate balance between the time spent to plan and the time spent to get additional information. Hsu [1990] proposes a method for planning with incomplete information. She shows that if the information available to the planner is not sufficient to produce a plan, then no amount of planning will help find the optimal solution. The idea is to generate a "most general partial plan" without committing to any choice of actions not logically imposed by the information available at that point. An anytime algorithm is then used to chose the appropriate action on the current partial plan when the system has to act. She defines a PERCEPT to be a (possibly partial) description of the world. Percepts are saved to form HISTORIES. A history prescribes or prohibits some actions, allowing the refinement of a partial plan. Finally, a plan is a mapping from histories to actions. Instead of using the most general partial plan she introduces the notion of effective partial plan. Conceptually an effective partial plan is a huge table where each entry contains a perceptual history and a set of actions. This resembles universal plans and is probably impractical unless powerful domain heuristics can be used to prune the search space.

6 Further Work and Conclusions

We are currently extending our strategy study to a 3-box world where the robot must bolt three boxes with three different bolts using three different wrenches.[5] Preliminary results suggest that as the problem becomes more complicated, Continue Elsewhere will begin to outperform Stop and Execute. This is due to the fact that BUMP with the Stop and Execute strategy is unable to plan more than one sensor operation ahead. This is too shortsighted for complex problems. More interestingly, preliminary results also suggest that neither of the general-purpose strategies are very good at avoiding failures, and that more specialized, domain-dependent strategies such as Sense Before Closing may be necessary.

To conclude, we have adapted a conventional planner to do deferred planning. This planner can then be used for problems where there is insufficient information at planning time to develop a complete plan. We have developed several strategies for deciding which plan goals to defer, and we are studying the performance of these strategies. In the 2-box study, the Stop and Execute strategy seems to perform (slightly) better than the other two strategies. The 3-box study is still in progress.

References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: an implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, Seattle, Washington, July 1987. American Association for Artificial Intelligence.

[Brooks, 1982] Rodney A. Brooks. Symbolic error analysis and robot planning. *International Journal of Robotics Research*, 1(4):29–68, 1982.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[Dean, 1987] Thomas Dean. Intractability and time-dependent planning. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, eds. M. Georgeff and A. Lansky. Morgan Kaufmann, San Mateo, California, 1987.

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, Minneapolis, Minnesota, August 1988. American Association for Artificial Intelligence.

[Dean and Kanazawa, 1988] Thomas Dean and Keiji Kanazawa. Probabilistic temporal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 524–528, Minneapolis, Minnesota, August 1988. American Association for Artificial Intelligence.

[Doyle, Atkinson and Doshi, 1986] R. J. Doyle, D. J. Atkinson, and R. S. Doshi. Generating perception requests and expectations to verify the execution of plans. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 81–87, Philadelphia, Pennsylvania, August 1986. American Association for Artificial Intelligence.

[Drummond, 1989] Mark Drummond. Situated control rules. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, May 1989. Morgan Kaufmann.

[Drummond and Bresina, 1990] Mark Drummond and John Bresina. Anytime synthetic projection: maximizing the probability of goal satisfaction. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, Massachusetts, August 1990. American Association for Artificial Intelligence.

[Durfee and Lesser, 1989] Edmund H. Durfee and Victor R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 58–64, Philadelphia, Pennsylvania, August 1986. American Association for Artificial Intelligence.

[5]There are, of course, problem instances where two or more bolts have the same size, but these are easier for the planner and therefore less interesting.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189-208, 1971.

[Georgeff and Lansky, 1987] 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, Washington, July 1987. American Association for Artificial Intelligence.

[Hayes-Roth, 1987] Barbara Hayes-Roth. Dynamic control planning in adaptive intelligent systems. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, pages 4-1–4-7, Arlington, Virginia, 1987.

[Henderson and Shilcrat, 1944] Tom Henderson and Esther Shilcrat. Logical sensor systems. *Journal of Robotics*, 1(2): 169–193, 1984.

[Hsu, 1990] Jane Yung-jen Hsu. Partial planning with incomplete information. AAAI Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments, March 1990.

[Kaelbling, 1988] Leslie P. Kaelbling. Goals as parallel program specifications. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 60–65, Minneapolis, Minnesota, August 1988. American Association for Artificial Intelligence.

[McDermott, 1978] Drew V. McDermott. Planning and acting. *Cognitive Science*, 2:71-109, 1978.

[Nilsson, 1989] Nils Nilsson. Action networks. In *Proceedings of the Rochester Planning Workshop*, pages 21-52, Rochester, New York, October 1988. University of Rochester.

[Nilsson, 1989] Nils J. Nilsson. Teleo-reactive agents. Draft Paper, Stanford Computer Science Department, September 1989.

[Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Schoppers, 1987] Marcel J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1039–1046, Milano, Italy, August 1987. International Joint Committee on Artificial Intelligence.

[Turney and Segre, 1989] Jennifer Turney and Alberto Segre. A framework for learning in planning domains with uncertainty. Technical Report TR 89-1009, Cornell University, May 1989.

[Wilkins, 1989] David E. Wilkins. Can AI planners solve practical problems? Technical Note 468, SRI International, Menlo Park, July 1989.

[Wilkins, 1988] David E. Wilkins. *Practical Planning – Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, San Mateo, California, 1988.