A Framework for Learning in Planning Domains with Uncertainty

Jennifer Turney
Alberto Segre*

TR 89-1009
May 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This research is supported by the Office of Naval Research grant N0014-88-K-0123.

A Framework for Learning in Planning Domains with Uncertainty

Jennifer Turney and Alberto Segre[1]

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

May 16, 1989

Attempts to apply classical planning techniques to realistic environments have met with two major difficulties. The first is that of *average-case inadequacy*. As one might expect, the (worst-case) computational characteristics of planning problems are ugly at best; a system that is to operate on problems of any reasonable size must rely on heuristics to reduce the average-case complexity of the problem. The second difficulty is that of *uncertainty*, or what to do when the real world doesn't work as expected.

We have shown in earlier work that particular machine learning techniques are a viable means of addressing the problem of average-case inadequacy [Segre88] in domains without uncertainty. This paper describes a planner operating in a realistic environment that is intended to support the same kind of learning. We report some preliminary experimental results comparing our planner with other approaches to planning in the presence of uncertainty.

[1] Support for this research was provided by Office of Naval Research grant N0014-88-K-0123.

1. Introduction

Most AI research in planning falls into the classical planning framework. A *classical planner* produces either a *linear plan* (e.g., an ordered set of operators) or a *nonlinear plan* (a partial ordering of operators with causal dependencies implicitly encoded in the ordering) for use by an *executive* in transforming the initial state of the world into a final state consistent with a prespecified goal.

1.1. Classical Planning

Research in the classical planning paradigm has always been carried out in environments without uncertainty. Planners such as STRIPS [Fikes72] and NOAH [Sacerdoti77] relied on both the *closed-world hypothesis* (essentially providing the planner with perfect knowledge of the world) and the *STRIPS assumption* (allowing the planner to know all operator effects completely *a priori*) to develop a static plan for later execution. Given these assumptions, the planner has the ability to project its perfect world description forward in accordance with its operators as it searches for a solution. Dealing with uncertainty is left to the executive; typically it would monitor the unfolding plan, reinvoking the planner should the world stray from the planner's original projections.

The tendency to diverge from the planner's projections is exacerbated by the presence of uncertainty in the domain. Any system that is to plan for tasks in a real-world environment must be able to *sense* the environment, *effect* changes in that environment, and *model* the environment. The classical planning framework assumes that the agent possesses perfect versions of these capabilities; a more realistic framework for planning would introduce uncertainty in each aspect.

(1) *Sensor error* occurs when the value returned by the sensor does not properly reflect the real world. For example, a sensor reading may be used to determine the position of an object for a robot system. The position may not be correct due to poor resolution of the sensor system itself.

(2) *Control error* occurs when an effector operating in the real world does not behave as expected. This might occur when a robot arm is asked to move to a given point in Cartesian space; the tolerances in the arm may prevent this position from being reached exactly.

(3) *Model error* occurs when the system's internal model of the world state diverges from the actual world. A robot system might model a block as an exact one inch cube; reality might be different since the block could not possibly be machined to arbitrary tolerances.

Clearly, classical planners are not equipped to deal with any of these types of uncertainty; they are also no better at addressing the problem of average-case inadequacy. Most planning research is carried out in microworld domains, where it is feasible to search the entire space without having to worry about efficiency. When classical planners *are* brought to bear on realistically-sized environments, heuristics must be used to restrict the search space.

A good example of a classical planning system adapted for use in a real-world domain is the SIPE planner [Wilkins88]. SIPE incorporates both domain-dependent and domain-independent heuristics to keep its search space within realistic resource bounds. By carefully crafting these heuristics, the system developer manually addresses the average-case inadequacy problem.

SIPE is unusual; most classical planning research ignores both problems. The TWEAK planner [Chapman85] makes no claims about its average-case performance; TWEAK incorporates no heuristics and makes no attempt to deal with uncertain environments. TWEAK is *complete* in the sense that it finds a solution to a planning problem if a solution exists, and *correct* in that the solution found will be guaranteed to work within TWEAK's idealized world. Note that if there is no solution to a given problem, TWEAK may never terminate. Chapman claims that the planning problem is undecidable, and that no upper bound can be placed on the amount of time necessary to find a solution. This makes the need for heuristics to provide average-case adequacy even more urgent.

Our previous work on *explanation-based learning* demonstrates how this particular machine-learning technique can be used to address the problem of average-case inadequacy [Segre88]. The ARMS system observes and analyzes solutions proposed by an external agent in order to improve its own planning performance. Thus a *learning-apprentice system* such as ARMS can be seen as a resource-limited planner, much like SIPE, augmented by a learning component that automatically acquires an operational set of search-control heuristics.

Unfortunately, the ARMS system still relies on simplifying assumptions about the world. In particular, ARMS is set in a robot assembly domain where all sources of uncertainty are ignored. If we wish to apply these techniques to more realistic domains, we must face the problem of uncertainty.

1.2. Improvisation

Recent work [Brooks87] suggests that the proper way to act in uncertain domains is often not to plan at all. It is argued that *creatures* built using this approach exhibit seemingly planful behavior by combining low-level hardwired behaviors and reacting to sensory input. This kind of *improvisation* is a notable departure from the smart planner/dumb executive framework of classical planning research. It is similar in spirit to other recent work in *situated activity* [Agre87]; the automated Pengo-player reacts to situations its sensors relate in the same manner the creatures react to sensory input from their world. Patterns of seemingly planful activity emerge from reactivity just as they emerge from improvisation.

There are two motivating factors behind the work in improvisation and situated activity. The first is the need to deal with uncertainty without resorting to probabilistic models. It is simply not possible to obtain the necessary information to support probabilistic modeling of the environments. Secondly, since the goal is real-time behavior, classical planning is just too time-consuming.

Regardless of motivation, either form of improvisatory behavior has serious shortcomings. While a creature may successfully act in real-time in an uncertain world, there is no getting away from the painstaking process of designing the low-level behaviors that combine to produce the overall behavior. This process is just like the process of designing heuristics for SIPE, with one additional disadvantage. The low-level behaviors which combine to produce the emergent behavior of the system are not accessible to the system itself.

Without at least some sort of memory, purely improvisatory systems correspond essentially to a gradient descent algorithm operating in a space defined by the initial state specification, sensory inputs, and the low-level behaviors.[2] Lacking the ability to reason about its environment dooms the system to be forever reactive. We would like our system to learn. Explanation-based learning techniques generally rely on a declarative knowledge representation as opposed to procedural hardwiring [Segre89]. Classical planning, notwithstanding all of its shortcomings, at least has the advantage of making indirect goal-achieving behavior possible.

1.3. Combining Planning and Improvisation

It is perhaps more reasonable to view the classical planning and improvisation approaches as two ends of a spectrum. The key to achieving effective real-world planning then becomes locating the quiescent point on the spectrum. It is this view which has motivated the current experiment; we have implemented an integrated planning and improvisation system we call SEPIA, for *S*ituated *E*xecution of *P*lanning and *I*mprovisation in *A*lternation. In this system, the role of the executive is extended to that of an improvisor acting within constraints imposed by a planner. The two components are allowed to execute in turn, under the principle that time spent planning will result in more directed behavior on the part of the improvisor, though too much planning may well be counterproductive.

2. The SHOPPER Domain

For the purpose of this paper, we will focus on the following domain (called SHOPPER) which, while being relatively easy to understand, can be altered to contain any of the elements of uncertainty that we wish to explore. The domain consists of some number of *shops* whose locations in the plane are fixed, along with a single *client*. The client has the ability to move within the plane and to make purchases at shops. If we take as our goal the task of visiting each shop while minimizing distance traveled, this first version of the problem is equivalent to the traveling salesperson optimization problem. Our planarity constraint serves to classify the problem as NP-hard.[3]

---

[2] In [Brooks88] we note the addition of memory to an improvisatory creature yields a form of goal-directedness; in this case "return to home" behavior.

[3] While there are heuristic solutions to the travelling-salesperson problem, the best known only guarantee a path no worse than 50% longer than the optimal distance.

We modify the travelling-salesperson problem by associating an opening, closing, and waiting time with each shop. A successful visit to a shop, or *purchase*, consists of the client remaining at an open shop for the duration specified by the shop's waiting time. If the shop closes before the waiting period has ended, the visit is not successful. The goal for this domain is to successfully visit a maximum number of shops, within the time limit imposed by the closing times of the shops.[4]

The client is equipped with a sensor that can be queried to yield the distance from one shop to another, or from the client's current location to a shop. The client has a second sensor that, when queried by the client, yields the waiting time at a given shop. Sensor error is introduced by allowing values returned by the sensors to deviate from the actual value. We can also introduce model error by considering the opening and closing times of the servers to be approximations (note that these values cannot be determined by sensors). Finally, we note that control error can be expressed as a discrepancy between the straight-line distance between two points and the actual distance travelled by the client, assuming that velocity remains constant.

3. The SEPIA System

The SEPIA system consists of two components, a *planner* and an *improvisor*, each operating in turn. A *control strategy* determines when to interrupt a component, transferring control to the other component. The two components communicate via a modified *assumption-based truth-maintenance system* (ATMS) [de Kleer86] that is initialized with the goal specification as well as a partial description of the initial state.

The planner is a non-linear planner that posts constraints on plan step orderings. Constraints are established by domain-specific rules available to the planner; when a rule can be instantiated and applied is determined by examining the contents of the ATMS. The constraints posted by the rule are also recorded in the ATMS, thus keeping track of what assumptions led to a particular constraint being posted.

The ATMS is augmented by the notion of a *current context* that serves to focus the attention of the planner. The current context is simply a set of assumptions that, along with all of the constraints deducible from this assumption set, represent the current partial plan.

Changes to the current context are fed to a RETE pattern matching network [Forgy82] which is used to maintain the conflict set of rule instances suitable for firing. Each rule is composed of *preconditions*, *constraints* on the possible instantiations of those preconditions, *sensor requests*, and *results*. The preconditions and constraints are compiled into the RETE net, and the resulting rule instance set is ordered by number of sensor requests that must be evaluated to fire the rule. Note that this ordering choice is

[4] The waiting time constraint could be reexpressed by adjusting the opening and closing times to reflect the narrower time slot in which the client can successfully visit the shop. This would be analogous to what roboticists call the configuration-space approach to obstacle avoidance.

arbitrary, following a heuristic of preferring rules which do not introduce new uncertain values into the system.

A rule is selected from the conflict set, and, if its sensor requests evaluate as true, the results of the rule are added to the ATMS. If at any time the current context becomes contradictory, a domain-specific contradiction-handling method is invoked to revise it, constructing a new, consistent current context. Naturally, this may in turn cause changes to the conflict set.

Each sensor request represents a query or a relation on queries of the world's current state. The results of a sensor request, which may depend on other sensor requests, are cached in the ATMS. The ATMS is always checked before making a new query; assuming that sensor requests are expensive, this ensures that no sensor request is made more than once.

The improvisor is free to act on the world, relying on a predefined *heuristic strategy* to govern its behavior. At any given time, the improvisor consults its heuristic strategy to determine what it should be doing, the only constraint being to respect the orderings imposed by the planner (as stored in the ATMS). As the improvisor acts on the world, the changes effected are recorded in the ATMS. These changes may affect the truth value of propositions already stored in the ATMS, causing, for example, the correctness of values returned by earlier sensor requests to be called into question. This may in turn affect the current context, thus focusing the planner's attention (when it eventually regains control) elsewhere.

3.1. SEPIA and Uncertainty

Uncertainty in SEPIA is handled by the ATMS. Every state descriptor obtained from a sensor reading or specified as part of the initial state description is considered to be an assumption. If and when the assumption is shown to be inconsistent, SEPIA revises its behavior accordingly.

The only certain information is posted by the improvisor. When the improvisor successfully completes a purchase, it is recorded in the ATMS as a premise. As more and more visits are made, ordering constraints which previously were justified by assumptions become justified by premises and cannot be defeated. The contradiction-handling method will never retract premises, only assumptions.

This method of dealing with uncertainty, along with the sensor-value caching scheme described above, permits the planner to treat the world as if there were no uncertainty; yet the planner can recover if it makes the wrong assumption without affecting conclusions that do not rely on the faulty assumption in the first place. This is fairly simplistic, but without some *a priori* knowledge about the characteristics of the environment's uncertainty, it is difficult to do any better.

3.2. Control Strategy

The performance of the system is critically dependent on its control strategy. Our current prototype has a simple control strategy. Each component is resource-limited: the planner is charged according to

the cost associated with each rule it fires, receiving a fixed budget for firings per planning cycle; the improvisor is limited to a constant number (30) of time units before interruption. In addition, the planner is allotted planning time while the client waits at a store, thus providing the system with a primitive concurrency simulation.

Our experience has been that the quality of the system's plan depends a great deal on the values of these resource-limiting parameters. We are currently experimenting with less heavy-handed control strategies. It should be possible to have the system initially plan until, for example, the conflict set size stabilizes. From then on, the improvisor should relinquish control only while waiting for service in a queue. Additionally, should the contradiction-handling method be invoked, the planner should again be permitted to plan until the conflict set size stabilizes.

3.3. Conflict Resolution

The RETE net maintains a conflict set of rule instances. Recall that this conflict set basically contains rule instances whose preconditions and constraints have been met, while their sensor requests have yet to be evaluated. The current prototype maintains this list ordered according to number of sensor requests necessary to fire the rule; the system will fire the rule instance with the least sensor interaction first. Some of the sensor requests in the rule may already be cached, while others will have to be established via calls to (possibly faulty) sensors.

Other conflict resolution strategies would naturally affect the performance of the planner. These resolution strategies are by nature domain-dependent; as we port SEPIA to other domains we expect to have to establish different resolution strategies.

3.4. Equating Planning Resources with Real Time

In classical planning, all planning work is done off-line before execution. This lack of real-time activity is exactly one of the issues improvisation is meant to address. In contrast, SEPIA interleaves planning and improvisation; thus the question arises of how to establish a single metric for measuring both planning time and execution time.

Assuming that planning is *not* free, there must be some way to decide how much real time is consumed by each rule firing. For the purpose of this paper, we have assumed that each rule's cost is fixed at a level proportional to the number of sensor requests it contains. The planner is interrupted when the cumulative cost exceeds its budget (currently set at 10 units). As we implement new control strategies such as the one mentioned above, a cost in time units will have to be associated with each rule.

4. Experimental Results

In this section, we describe the results obtained on randomly generated SHOPPER worlds of five and ten servers by several different techniques, including SEPIA.

(1) *Classical planning.* This system does an exhaustive search through the space of all shop visit orderings for the best performing plan. The system bases its projections on the data provided by faulty sensors. No execution monitoring is in effect; the executive applies the plan derived before beginning the trial. Note that the time for planning, while notably high, is not considered; search is assumed to be free.

(2) *Improvisation.* This system is most akin to Brooks' creatures. The improvisor accepts a functionally-specified heuristic strategy as an argument. This strategy corresponds to the hardwired control of Brooks' modules. Several strategies were provided; these strategies can become quite complex. In this paper we report data collected on runs using the *Stupid* and *Greedy* strategies. The *Stupid* strategy visits the servers in numeric order. The *Greedy* strategy visits the closest as yet unvisited server still open at selection time, taking into account estimated travel time.

(3) *SEPIA* was tested using a library of simple schemas, including one for naive projection, and both of the same heuristic strategies used in the improvisation system trials. The control strategy for interrupting planning and/or improvisation was fixed; a set number of rules were fired first, followed by improvisation. Thereafter, planning occurred whenever the client was waiting at a shop.

For each of the trials, control and sensor error were enabled, but no model error was introduced. A reasonable effectiveness metric for this domain would be number of purchases divided by time of execution. For the purposes of this paper, we are not interested in planning cost, but rather only execution cost and effectiveness; in other work we note that there are other interesting metrics for comparing this kind of system, especially when a learning algorithm is involved [Segre87].

Table 1 summarizes the effectiveness of each of the five systems. The classical planning system was not run on the SHOPPER world of ten servers due to its high planning cost.

5. Discussion

As expected, uncertain environments force the abandonment of classical planning techniques. The state-space search was not only too expensive to run on larger worlds, but simply performs poorly in the presence of sensor and control error, even in smaller worlds.

The more interesting comparison seems to be between the improvisation strategy *a la* Brooks and the SEPIA system. Both of these systems are running the identical improvisor; SEPIA is simply augmented by a planning component and a control strategy to manage the two components. As was expected, the quality of the heuristic improvisation strategy had the largest effect on the quality of the either system's solution. The more domain-specific information was procedurally encoded in the heuristic, the better the

Table 1
Experimental Results

	5 Server World				10 Server World			
	Visits	Purchases	Distance	Time	Visits	Purchases	Distance	Time
	Classical Planning							
mean	6	2.8	258.0	285.8	*planning exceeds CPU time limit*			
s	0	0.32	8.80	7.96				
	Improvisation with *Stupid* heuristic							
mean	10.40	2	236.76	264.76	10	3	458.96	505.84
s	2.33	0	9.63	10.11	0	0	14.68	15.42
	SEPIA with *Stupid* heuristic							
mean	9.16	2	235.08	254.48	10	3	464.88	492.72
s	2.38	0	10.94	11.74	0	0	14.23	17.88
	Improvisation with *Greedy* heuristic							
mean	3.04	2.72	144.12	167.24	4.24	3.16	137.96	200.28
s	0.20	0.45	12.37	10.19	0.59	0.46	22.60	19.35
	SEPIA with *Greedy* heuristic							
mean	3.68	3.24	151.08	162.08	5.16	4.12	177.16	194.76
s	0.55	0.43	11.99	11.99	0.67	0.71	28.43	27.45

performance.[5]

The interesting result is that SEPIA significantly outperforms the improvisor in the *Greedy* heuristic case, both in terms of average number of purchases and in the metric of purchases per unit time proposed above. Recalling that the total time spent on the shop task includes time spent planning, it is also significant that SEPIA completed its task in less time on average.

These preliminary experimental results are encouraging. We are currently experimenting with different control strategies, and we are also adding an explanation-based learning component [Segre89] to SEPIA in the SHOPPER domain. The learning component should be able to increase the efficiency of the planning component, thus enabling less time to be spent planning for the same results.

A version of SEPIA has been implemented which relies on a constraint-propagating truth maintenance system similar to that described in [McAllester82], and work is in progress to incorporate a learning element as described above. Finally, SEPIA is being ported to a two-player wargame domain with sensor, control and model error.

---

[5] The greedy and stupid strategies reported here are the opposite ends of the spectrum. We have run the improvisor, as well as SEPIA, using a wide variety of heuristic improvisation strategies.

References

[Agre87] P. E. Agre and D. Chapman, "Pengi: An Implementation of a Theory of Activity", *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, 268-272 .

[Brooks87] R. A. Brooks, "Planning is Just a Way of Avoiding Figuring Out What to Do Next", Working Paper 303, MIT Artificial Intelligence Laboratory, Cambridge, MA, September 1987.

[Brooks88] R. A. Brooks, "Herbert: A Second Generation Mobile Robot", AI Memo 1016, MIT Artificial Intelligence Laboratory, Cambridge, MA, January 1988.

[Chapman85] D. Chapman, "Planning for Conjunctive Goals", Technical Report 802, MIT Artificial Intelligence Laboratory, Cambridge, MA, November 1985.

[de Kleer86] J. de Kleer, "An Assumption-based TMS", *Artificial Intelligence 28*, 2 (March 1986), 127-162.

[Fikes72] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans", *Artificial Intelligence 3* (1972), 251-288.

[Forgy82] C. L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *Artificial Intelligence 19* (1982), 17-37.

[McAllester82] D. A. McAllester, "Reasoning Utility Package User's Manual, Version One", Memo 667, MIT Artificial Intelligence Laboratory, Cambridge, MA, April 1982.

[Sacerdoti77] E. Sacerdoti, *A Structure for Plans and Behavior*, American Elsevier, Menlo Park, CA, 1975, 1977.

[Segre87] A. M. Segre, "On the Operationality/Generality Trade-Off in Explanation-Based Learning", *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, 242-248.

[Segre88] A. M. Segre, *Machine Learning of Robot Assembly Plans*, Kluwer Academic Publishers, Norwell, MA, March 1988.

[Segre89] A. Segre and C. Elkan, "Not the Last Word on EBL Algorithms", Technical Report Cornell Un Department of Computer Science, 1989.

[Wilkins88] D. E. Wilkins, *Practical Planning*, Morgan Kaufmann Publishers, San Mateo, CA, 1988.