Evaluating Planning Through Simulation: An Example Using Phoenix *

Adele E. Howe
Computer Science Department
Colorado State University
Fort Collins, CO 80523
Net: howe@cs.colostate.edu

Introduction

For some time, planning researchers have advocated the use of simulators of "complex" environments, domains or problems [Erman, 1990, Drummond *et al.*, 1990]. Yet, a casual survey of planning papers in the last few AAAI and IJCAI proceedings shows that Blocksworld continues to dominate as the domain of choice. Blocksworld is well-known and simple enough that many properties of planners can be demonstrated by examining the search space rather than simulating performance. As a consequence, we, as a field, are still accumulating experience in using and evaluating planners in simulated environments.

In this paper, I will describe my experiences in using the Phoenix simulator to investigate failure recovery [Howe and Cohen, 1991, Howe, 1992, Howe, 1993]. This paper is *not* intended to be a summary of the state of the art in evaluation or a description of how generally to evaluate AI Planners or to design and run experiments. (Readers interested in these topics are directed to [Cohen and Howe, 1988, Langley and Drummond, 1990, Cohen, 1991, Cohen, 1993].) Instead, the paper discusses the role of simulators in planning research and describes how the Phoenix simulator facilitated and obstructed my research. Additionally, the paper lists some of the pitfalls that I encountered and concludes with advice on how to avoid them.

*This research was conducted at the University of Massachusetts and was supported by a DARPA-AFOSR contract F49620-89-C-00113, the National Science Foundation under an Issues in Real-Time Computing grant, CDA-8922572, and a grant from the Texas Instruments Corporation. I wish also to thank Paul Cohen for his advice and support in conducting this research.

Subject of Study: Failure Recovery in Phoenix

As part of my thesis research [Howe, 1993], I added a failure recovery component to the Phoenix planner, evaluated it, described a procedure for analyzing the behavior of the planner and the new component and then evaluated that procedure. Failure recovery played two roles; it was part of the function of the planner and so a product of the research, but it was also a subject of study and provided data for analyzing the behavior of the planner.

The Phoenix Environment and Simulator

The test environment was a simulation of forest fire fighting in Yellowstone National Park. The goal of forest fire fighting is to contain fires. In this simulation, a single agent, the fireboss, coordinates the efforts of field agents to build fireline, cleared areas of the forest that halt the spread of fire. Fire spread is influenced by weather and terrain. While terrain stays constant, weather changes constantly; agents acting to contain the fire must be prepared for these changes to invalidate their expectations and perhaps cause their actions to fail.

Figure 1 shows the interface to the simulator. The map in the upper part of the display depicts Yellowstone National Park north of Yellowstone Lake. Features such as wind speed and direction are shown in the window in the upper left, and geographic features such as rivers, roads and terrain types are shown as light lines or grey shaded areas. The fireboss and other agents are clustered at a base location just north of the lake.

The Phoenix system comprises a forest fire simulator, an agent architecture, a set of knowledge bases for the different types of agents, and an interface for conducting experiments [Cohen *et al.*, 1989]. The forest fire simulator is tunable through the experiment interface, which allows the experimenter to control the environment and gather data.

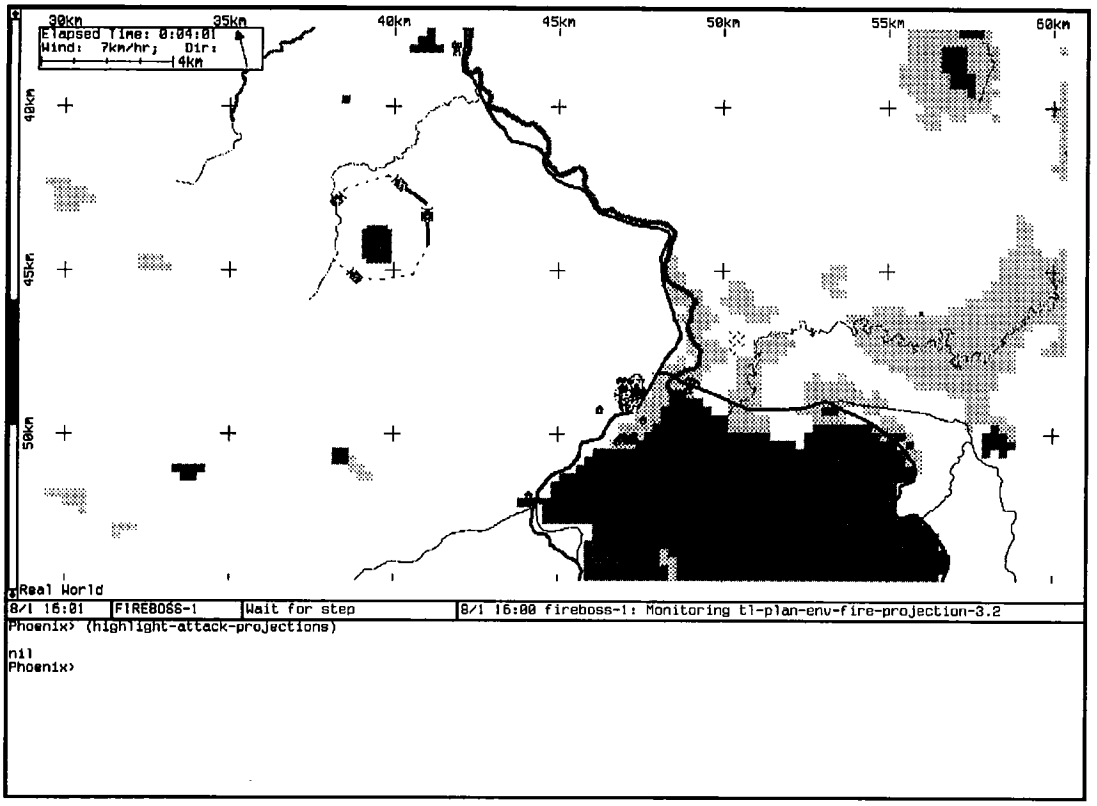

Figure 1: View from Phoenix simulator of bulldozers fighting a fire.

The experimenter can control two aspects of the environment: the initial state and the way the environment changes over time. Initial state includes the number of agents of different types, what they know and how they are initially situated in the world. Environmental change is controlled by scripts, which describe the type, amount, time and frequency of changes to the environment. Any environment parameter can be changed with scripts; typically, the scripts change dynamic features such as wind speed, wind direction, humidity, temperature, moisture content of parts of the park (e.g., as in when it rains), and the location and incidence of fire. Scripts can also specify the collection of time-stamped data at intervals over an experimental trial. Alternatively, the experiment can collect data on the environment and the agents, accessing any process information available, in retrospect, about the execution. For the agents, this includes their long-term memories about what they 'thought', how they acted, and how they perceived the environment.

Failure Recovery and Failure Recovery Analysis

Plan failures are common in some environments. In fact, both execution time and planning time failures are common for agents in Phoenix. For my research, I combined two approaches to limiting the frequency and impact of failures: automated failure recovery and debugging failures. Automated failure recovery both repaired failures and expedited debugging the Phoenix planner using a procedure called *Failure Recovery Analysis* (or FRA).

The automated failure recovery component applies general methods to recover from failures detected by the Phoenix planner. The design of the failure recovery component was based on an expected cost model of failure recovery. The model served as the basis of three experiments in which I tested the assumptions of the model, compared two strategies for selecting recovery methods (one of which was derived from the model), and evaluated the performance of the system after adding two new recovery methods.

Failure recovery analysis is a procedure for analyzing execution traces of failure recovery to discover how the planner's actions might be causing failures. The procedure involves statistically analyzing execution traces for dependencies between actions and failures, mapping those dependencies to plan structures, and explaining how the structures might produce the observed dependencies. I evaluated the procedure by applying it to explain

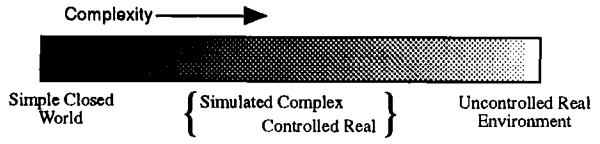

Figure 2: Spectrum of Complexity of Domains for Planning

some results of the experiments in which I evaluated automated failure recovery and by analyzing the Phoenix data for FRA's sensitivity to noise and size of the execution traces.

Role of the Simulator

Although Blocksworld is the most common planning domain, planners have been constructed for many different domains from simple and easily described to complicated and hard to understand fully. Figure 2 positions the different domains on a spectrum. Blocksworld is the best known example of a constrained closed world domain. At the other end of the spectrum are uncontrolled, "real" environments that either involve the normal complexity of everyday human life or are particularly challenging even for humans; the JPL Planetary Rover [Miller *et al.*, 1982] and the Transportation Planning Problem for USTRANSCOM [Day, 1992] exemplify such domains. Between the almost trivial and almost impossible lies both simulated complex environments and controlled real environments. Examples of simulated complex environments include Phoenix and Tileworld [Pollack and Ringuette, 1990]; an example of a constrained real environment is an underwater remotely-piloted robot in a test tank [Bonasso, 1991].

The Phoenix simulator was integral to my research on failure recovery. First, the simulator provided a platform in which to test the feasibility of the design for failure recovery. I designed a failure recovery component and modeled it; the simulator helped demonstrate the utility of the failure recovery design and was used to test the assumptions of the model. Second, the simulator allowed the collection of data from 30,000 simulated hours of fire fighting (without burning a single tree). The failure recovery research involved many pilot studies and four experiments, which together required three weeks of execution time on a TI Explorer. Without the simulator, I could not have run enough trials to test statistical significance and would never have noticed long-term trends in the data, trends that motivated the design of Failure Recovery Analysis. Third, the experiment interface expedited testing hypotheses because it grants considerable control to the experimenter. In each experiment, I tested hypotheses about the performance of failure recovery under varying conditions; the experiment interface allowed me to specify which aspects of the experiment should be changed and which should remain the same.

Advantages of Using a Simulator

The obvious advantage of a simulator is that the experimenter is in control. One can define the conditions that should hold in the environment to test one's hypothesis or evaluate the performance of a system. Additionally, instrumentation is easily obtained by adding hooks to the planner or simulator.

Embedding the planner in a simulated environment with an experiment interface supports "soup to nuts" inquiry. Many stages of experimentation can be automated, from running trials and collecting data to partially interpreting the results. With the facilities in the simulator, I wrote code to run multiple trials of the experiment, gather data, process the data and perform basic statistical analyses on it.

Another advantage of simulation is that it expedites comparing the behavior of the system over time and across environment conditions. I was able to gather execution data on the long term effects of planning actions and to search for common behaviors in differing conditions of the environment.

Simulators are portable domains. We can copy software from one system to another and expect that it will perform roughly the same. Simulators offer us the chance to replicate and compare results on shared metrics.

The possibilities for replication and the ability to define whatever metrics we want mean that it should be possible to generalize results beyond a single simulator. When we can look inside the environment to see what it is doing, we should also be able to disregard superficial differences, such as between forest fire fighting and oil spill containment, and recognize the commonalities in approach.

Disadvantages of Using a Simulator

Although my research could not have progressed without a simulator, using Phoenix did have its downside. One major problem with complex simulation is that so much effort can be expended acquiring domain-specific knowledge that any results obtained from the planner may be specific to the domain. In other words, it can become difficult to attribute credit for the system's performance: was the algorithm responsible or was the knowledge base? I found that my approach to failure recovery worked for Phoenix, but at present, I can only argue, based on the requirements of the approach and similarities in the characteristics of the environments, that it will work in other environments.

The most often cited problem with simulators is that they are not the real world, but are only approximations of it. In particular, simulators tend to avoid dealing with the realities of noisy sensors and faulty effectors [Gini, 1988]. Additionally, they do not capture richness of sensation or true parallelism and so can mislead us about what will work in "real" environments.

Simulators depend on their hardware and software platforms. The major problem with this dependence is that it can be difficult to predict the effects of changes in hardware or software. I determined early in my pilot studies that I needed to run the experiments on machines with exactly same configuration (hardware, operating system and application software) because the same experiments took longer to run and the results did not look comparable on other "similar" machines.

Simulators may give a false feeling of control. The effect of the platform is one factor that may cause hidden differences. Other factors that I had difficulty controlling were: system stability, programming errors that led to collecting different data than I had intended, and updates to the software. When experimenting with a new system, one should expect a lot of bugs and system crashes to occur; in fact, with Phoenix, we sometimes spent days fixing the system after installing modifications to make Phoenix stable enough to run hours of experiments without crashing. Also, instrumentation is not immune to programming bugs. The hardest aspect to control was the installation of new software: patches and updates to the platform and to the simulator. My experiments were run over a period of several months; in that time, I could not prevent programmers from fixing bugs and support people from installing updates. In at least one case, I know that an update introduced a bug in the planner. The bug produced data that was dramatically different from a previous experiment and made comparing results from the two experiments a questionable activity. As it happened, that experience had the positive effect that I realized I could detect bugs in the program by simply looking at the data I had collected, but I paid for the serendipity by a loss in generality of other results.

Exploiting the Advantages While Avoiding the Pitfalls

Simulators provide a remarkable flexibility and degree of control for evaluating planners. The problem is that it is so easy to dash off experiments that the results can be uninterpretable. As in any experimental activity, one needs to have clearly stated hypotheses and needs to ensure that the experiment is indeed testing those hypotheses under adequately controlled conditions. Based on my experiences, I offer the following suggestions for avoiding some of the pitfalls of experimentation with a complex simulator.

First, pilot studies are essential! Bugs need to be worked out of both the planner and the experiment itself. Typically, the simulator will create situations that the programmers never thought to or did not have the time to test. The experiment may sound good on paper, but may not be adequately controlled. I ran three pilot studies to test the simulated conditions of my experiment and still had to run the first experiment twice because of corrupted data.

Second, the planner and the simulator should be isolated from modifications. Unless you can be certain that no one can change the system (planner and simulator) during the experiment, the results from one controlled experiment may not actually be the same as those in the "same" experiment run later.

Third, dependent and independent variables should be selected carefully and understood thoroughly. Because of a difference in the representation of different actions, I implemented the instrumentation for one dependent variable with a test for the type of action. I thought the difference was largely syntactic, but in fact, analyzing the data showed that the value of the dependent variable depended, oddly enough, on another dependent variable and did not measure what I had intended it to measure.

Fourth, experiments based on concrete hypotheses are much easier to understand and critique than exploratory experiments. Exploratory experiments are important for helping narrow the hypotheses, but the results should be viewed as preliminary.

Simulators have been invaluable in identifying canonical problems and in making ideas concrete. However, their flexibility and ease of use can lead us astray. In particular, we need to learn how to factor in the contribution of knowledge to the performance of systems, how to formulate and test hypotheses in simulated environments and how to generalize our results beyond a single planner in a single simulator.

References

Bonasso, R. Peter 1991. Underwater experiments using a reactive system for autonomous vehicles. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA. American Association for Artificial Intelligence. 794–800.

Cohen, Paul R. and Howe, Adele E. 1988. How evaluation guides AI research. *AI Magazine* 9(4):35–43.

Cohen, Paul R.; Greenberg, Michael; Hart, David M.; and Howe, Adele E. 1989. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine* 10(3).

Cohen, Paul R. 1991. A survey of the eight national conference on artificial intelligence: Pulling together or pulling apart? *AI Magazine* 12(1):16–41.

Cohen, Paul R. 1993. *Empirical Methods in Artificial Intelligence.* In Preparation.

Day, David S. 1992. Acquiring search heuristics automatically for constraint-based planning and scheduling. In Hendler, J., editor 1992, *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS92)*, San Mateo, CA. Morgan Kaufmann Publishers, Inc. 45–51.

Drummond, Mark E.; Kaelbling, Leslie P.; and Rosenschein, Stanley J. 1990. Collected notes from the benchmarks and metrics workshop. Artificial Intelligence Branch FIA-91-06, NASA Ames Research Center.

Erman, Lee D. 1990. Intelligent real-time problem solving (IRTPS): Workshop report. Technical Report TTR-ISE-90-101, Cimflex Teknowledge Corp.

Gini, Maria 1988. Automatic error detection and recovery. Computer Science Dept. 88-48, University of Minnesota, Minneapolis, MN.

Howe, Adele E. and Cohen, Paul R. 1991. Failure recovery: A model and experiments. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA. 801–808.

Howe, Adele E. 1992. Analyzing failure recovery to improve planner design. In *Proceedings of the Tenth National Conference on Artificial Intelligence.* 387–393.

Howe, Adele E. 1993. *Accepting the Inevitable: The Role of Failure Recovery in the Design of Planners.* Ph.D. Dissertation, University of Massachusetts, Department of Computer Science, Amherst, MA.

Langley, Pat and Drummond, Mark 1990. Toward an experimental science of planning. In Sycara, Katia P., editor 1990, *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control.* Morgan Kaufmann Publishers, Inc. 109–114.

Miller, David P.; Desai, Rajiv S.; Gat, Erann; Ivlev, Robert; and Loch, John 1982. Reactive navigation through rough terrain: Experimental results. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA. 823–828.

Pollack, Martha E. and Ringuette, Marc 1990. Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eight National Conference on Artificial Intelligence*, Boston, MA. 183–189.