The utility of planning

Shlomo Zilberstein
University of Massachusetts
Department of Computer Science
Lederle Graduate Research Center, Box 34610
Amherst, MA 01003-4610
shlomo@cs.umass.edu

Abstract

Evaluation and comparison of existing planning systems is hard because they disagree on the fundamental role of planning, on evaluation metrics, and on the notion of success and failure. This paper suggests a decision-theoretic approach to evaluate planning systems that generalizes the role of planning in intelligent systems. The planner is viewed as a source of information that is used by an execution architecture in order to select actions. A planner is only as good as the effect it has on the performance of an operational system. Our approach calls for a clear separation between the planning component and the execution architecture and for evaluation of planning systems within the context of a well-defined command, planning and execution environment. The evaluation is based on the expected utility of the domain histories that are induced by the planning component.

1 Evaluating planning systems

The task of evaluating planning systems becomes exceedingly hard as one tries to measure their comprehensive value. The evaluation task becomes even harder when it is used to study the qualitative structure[1] of planning systems. At the same time, one can trivialize the evaluation by looking at a limited set of isolated performance measures. In this section we examine the various aspects of planning systems that complicate their evaluation.

1.1 Base-level evaluation

Base-level evaluation of planning systems refers to analyzing the performance of the planner using a variety of possible metrics. A number of reasons make this type of evaluation insufficient for comparison purposes.

First, there is no unified view of what constitutes a planning task. As a result, one must compare planners that essentially address different problems. In the early days of AI there used to be a more unified view of planning. For a long time planning has been strongly associated with problem solving and was formulated using domain *states*, *operators*, and *goals*. The planning problem, generally stated, was to find a set of operators whose execution (subject to some ordering constraints) will change a given initial state to a state satisfying the goals. Over the years, this approach has been proven unsatisfactory and oversimplified. In particular, it ignores the execution process embedded in any real planning system, the possibility of partial goal satisfaction, and recovery from failures. More recent implementations of planning systems adopted a variety of different and sometimes incompatible definitions of the problem, making it hard to compare and evaluate systems. How can PRS-style planning, for example, be compared with BPS-style planning? How can an adaptive planner that generates new plans be compared with a planner that only retrieves existing plans?

Another difficulty in evaluating planning stems from the multiple dimensions that characterize any planning system. These dimensions include correctness, the quality of solutions, the failure rate, the resources required to generate plans, robustness of the planner, graceful degradation, and user friendliness. Some of these performance measures are hard to evaluate individually. It is much harder to compare systems that have different strengths and weaknesses along those dimensions. For example, suppose that system A generates reliable plans (failure rate = 2%) within 60 seconds, and system B generates less reliable plans (failure rate = 10%) within 25 seconds. How can one determine which system is "better"? Obviously, much more information about the nature of the task and the implications of failure is needed in order to compare these two systems. When one considers a dozen of different performance measures and a system operating in a complex, non-deterministic environment, evaluation becomes very hard.

In some domains, such as medical diagnosis and treatment, human performance is used as the ultimate benchmark. This complicates the evaluation of base-level performance due to subjective human judgment, inconsistency of human knowledge, and the complexity of modeling human decision-making processes.

The complexity of the evaluation process requires the use of both analytical and experimental tools. The complexity of domains such as air traffic scheduling, and the multidimensional evaluation criteria make it almost impossible to model and analyze the behavior of planning systems using analytical tools alone. Moreover, analytical tools are normally used to derive worst case, *asymptotic* complexity, while a superior planning approach should be selected based on *expected* performance with respect a particular probability distribution of problem instances. As a result, empirical methods will play an important role in this process.

1.2 Additional performance metrics

Besides the above base-level performance measures, there is a set of additional metrics that describe important features of planning systems. One set of measures evaluates the design of planners. These measures include generality over problem domains, scalability, ease of implementation and ease of analysis. As researchers who are primarily interested in revealing the qualitative structure of planning systems, we have only limited interest in planning systems that exhibit good performance in one particular domain but are hard to analyze or apply to different domains. Understanding and evaluating

[1] The term *laws of qualitative structure* was used first by Al Newell and Herb Simon [10] in their Turing Address to describe the qualitative generalizations and regularities that have become the most significant, lasting results of scientific research.

the design are at least as important as the evaluation of base-level performance. The ultimate goal of evaluation should be to understand the structure of a planner and to answer such fundamental questions as: what makes a system successful in a certain environment and how do different performance measures interact.

As the focus of attention moves from base-level performance to understanding the qualitative structure of planning systems, qualitative evaluation becomes more important and more feasible than quantitative evaluation. The main goal of such evaluation is to identify the features of planning systems that enable them to cope with resource limitations, uncertainty, and failure. Understanding the implications of these factors on the design of planning systems is much more important than evaluating the particular performance of a planning system on a single problem.

To summarize, evaluation of planning systems is a hard process since the success of a planner can be measured along many different dimensions. Evaluation is especially hard when used to understand the structure of a planner that performs well in a certain set of domains.

The rest of this paper addresses two major aspect of evaluation: the need for a unified view of planning and a unified approach to evaluation of planning systems. Section 2 examines the role of planning in intelligent agents. Section 3 argues that planners should be evaluated within the context of a complete system and with respect to a *particular* execution architecture. Section 4 outlines a decision-theoretic framework that generalizes the role of planning and facilitates evaluation and comparison of different planning paradigms. Finally, Section 5 summarizes our approach.

2 The role of planning revisited

Evaluation and comparison of planning systems can be simplified if we redefine and generalize the role of planning in intelligent systems. Planning is a deliberation process aimed at helping an agent to *act* intelligently in the world. Various mechanisms have been used in planning systems to achieve this goal. The following list summarizes some of these mechanisms:

1. Plan synthesis using domain knowledge and such principles as means-ends analysis and precondition achievement have been used since the early days of AI. These principles have been widely used by planning systems such as STRIPS [3], NOAH [11] and their descendents. Plan synthesis reduces future deliberation and helps the agent achieve its goals .

2. Projection of possible future world states using domain knowledge. Such projections can improve performance by taking actions that correspond to anticipated events and by taking actions that can avoid anticipated problems. These principles are used in stochastic planners such as [4, 7].

3. Forming constraints on future action improve performance by reducing future deliberation and focusing future search on feasible plans. As suggested in [12], similar constraints can be derived with respect to any aspect of a plan including protection conditions, resources, authority requirements, spatial constraints, etc.

4. Retrieval of existing plans from a library, or case-based planning [6], can further reduce run-time deliberation and utilize external knowledge and expertise.

5. Causal and temporal reasoning can improve performance by selecting actions that have maximal anticipated effect on the level of goal achievement.

Regardless of the planning techniques that are used, the information provided by the planner combined with an appropriate execution architecture increases the agent's performance. In that sense, planners are only as good as their contribution to the performance of the agent.

Analyzing and evaluating planning systems based on the quality of the behavior that they produce may seem quite natural, but this criterion has not been the primary component of evaluation in the past. Instead, planning systems have been widely evaluated based on correctness or optimality of the solutions, deemphasizing the role of planning in performance improvement. The hidden assumption was that an agent needs a correct (or sometimes "optimal") plan and that executing a correct plan always leads to maximal goal achievement. But this assumption hardly holds in any real-world situation. Soundness and completeness of planners are important properties, but they provide little insight into the practical value of a planning system.

Overall performance on the other hand is a metric that summarizes many of the measures that were introduced in the previous section such as the quality of solutions, the failure rate, the resources required to create the plan, and robustness of the planner. In order to develop a methodology to evaluate overall performance and better understand different planning systems, we must first recognize some basic facts regarding the role of planning in intelligent systems. The main issues that need to be explicitly addressed are discussed below.

2.1 The goal of planning

Planning helps agents to choose actions intelligently. The ultimate goal of an agent is not to derive "optimal" plans, or even "correct" ones, but rather to transform the world into a desired state and thus perform a certain task. One can describe such goals using time-dependent utility functions that define the desirability of certain world configurations. Time-dependent utility functions extend the traditional notion of goals (by allowing partial goal satisfaction) and the traditional notion of deadline (by allowing gradual loss of utility as a function of time). Given such a utility function, a planner should be evaluated within the context of a complete working system. Other properties of planners such as complexity, correctness, and completeness are very important but they are insufficient for overall evaluation or comparison purposes. In most cases we cannot equate "correct" planning with good agent performance.

2.2 Planning as a deliberation process

Planning is a reasoning process and as such it does not immediately change the external world or achieve the ultimate goals of an agent. As a result, one must view the outcome of a planning process as a potentially valuable piece of *information*. The value of this information depends on three factors:

1. The objective quality of the plan that reflects the quality of the solution to the problem defined by the initial planning conditions.

2. The time at which the information that the planner produced becomes available to the system and the extent of change in the domain.

3. The capability of the agent to interpret the information produced by the planner and to exploit it effectively.

Standard decision-theoretic techniques can be used to determine the value of planning both analytically and experimentally [9, 14].

2.3 Planning as a resource-bounded activity

The various factors that determine the value of a plan are not independent. In particular, the deliberation time required to improve the quality of a plan will normally degrade the overall utility of the agent. Hence, it is useful to analyze planning as a resource-bounded activity and to develop planning systems that can trade off planning quality for deliberation time. Since in many domains the value of continued planning is context dependent, a utility maximizing approach must rely on constant, real-time monitoring of the planning process.

Dean, Horvitz, Russell, Zilberstein and others have shown that anytime algorithms offer a simple means by which an agent can trade off decision quality for deliberation time. In addition, efficient models have been developed for composition and monitoring of systems that are composed of anytime algorithms [13].

2.4 Planning versus problem-solving

Historically, planning has been closely associated with problem solving and similar search techniques have been widely used in both areas. However in many "real world" environments planning is much more than problem solving. The difference does not relate to the resource constraints and real-time nature of planning – similar considerations apply to real-time problem solving. Besides the richer representation of operators and goals in planning, the main difference between planning and problem solving is the temporal scope. In particular, a planning problem can contain much more than a single problem solving episode. In some cases, the planning component amounts to a sequential problem solving process. But other intelligent agents are designed to operate in an environment over an extended period of time. Their long term goal may be "keep all the machines in this room working" or "keep track of all the targets in region N". Translating such a high level goal into action may not have a fixed solution that can be derived and implemented. Achieving such goals requires an ongoing situation assessment, planning, and action. Such systems may be even harder to evaluate, but in many ways they better represent practical problem domains.

To summarize, planning is much more than problem solving. In order to evaluate various planning systems we need to modify the narrow definition of planning as "a partially ordered set of operators to be used in reaching a goal". Instead, planning should be viewed as a resource-bounded, reasoning activity that provides useful information to an execution architecture in selecting actions. The overall utility of a planner is determined by the effect it has on the agent behavior.

3 Execution architectures

In this section we argue that planning systems should be designed and evaluated with respect to a particular *execution architecture*. In many existing planning paradigms the problem domain is well specified but the execution architecture is either trivial or ill-specified. In other systems, the execution architecture becomes the central component and planning is virtually nonexistent. For example, in STRIPS style planning and many of its descendants, it is assumed that execution is the obvious process of carrying out the actions one by one subject to some ordering constraints. In other systems, based on condition-action rules or a blackboard architecture, the execution architecture is more complex, but planning (in the very general sense defined above) does not exist. In reality, planning must be accompanied by an execution architecture that can translate the plan into individual actions, monitor their execution, recognize various types of failures, recover from "simple" failures, and otherwise rely on replanning.

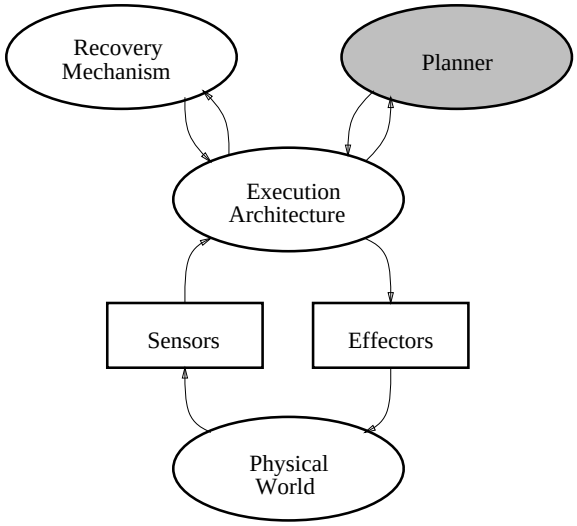

Figure 1: In practice the planning component interacts with the execution architecture, not with the physical world

The execution architecture communicates and interacts with the three major entities shown in Figure 1. The first entity is the *physical world* that it manipulates. The primary task of the execution architecture is to monitor (using sensory input) the execution of actions (using certain manipulators) that change the physical world. The second entity is a *recovery mechanism* that can suggest ways to recover from certain "simple" failures. Recovery is normally performed using a simple planning mechanism or a reactive component whose goal is to achieve the desired effect of the failed action. The third entity is a planner. The execution architecture sends to the planner information about the current state and receives from the planner information on how to act intelligently to achieve the goals. We deliberately avoid in this paper (except for the purpose of giving examples) the discussion of particular representations of states, operators, and plans. Different representations may be used by different systems, but the general role of the execution architecture remains the same over different planning approaches.

3.1 An example of an execution architecture

To demonstrate the notion of an execution architecture, we describe below a simple example. This execution architecture has been used in an application that is also described.

The plan segments that the execution architecture receives from the planner are abstract plans in the sense that each step may require a number of primitive actions and in the sense that the order of the primitive actions is not fully specified. Therefore, the execution architecture needs to map each step of the plan into a sequence of primitive actions. The execution of each primitive action is controlled and monitored

1. While there exists an active task do
   (a) Retrieve the next abstract plan segment from the planner
   (b) Map the plan into concrete base-level actions
   (c) Execute each primitive action
   (d) When a primitive action fails, use a local recovery mechanism to achieve the intended effect of the action
   (e) When a plan segment fails, inform the planner

Figure 2: A simple execution architecture

by the execution architecture. The control of execution determines such run-time parameters as resource consumption and execution speed. The monitoring function verifies that the execution is successful. In case of a failure, the recovery mechanism is used to find an alternate set of primitive actions that can achieve the desired effect of the failed action. As a last resort the execution architecture can inform the planner that the execution of a plan segment has failed.

We have used a similar execution architecture in [14] to control a simulated mobile robot whose sensing and planning components were implemented as anytime algorithms. In this application, a robot is situated in a simulated, two dimensional environment with random obstacles. The robot does not have an exact map of the environment but it has a vision capability that allows it to create an approximate map. The environment is represented by a matrix of elementary positions. The robot can move between adjacent cells of the matrix at a varying speed which affects the execution time of the plan as well as the energy consumption. When the simulation starts, the robot is presented with a certain task that requires it to move to a particular position and perform a certain job. Associated with each task is a reward function that determines the value of the task as a function of completion time. The system is designed to control the movement of the robot, that is, to determine its direction and speed at each point of time while maximizing the overall utility. The overall utility depends on the value of the task (a time dependent function), and on the amount of energy consumed in order to complete it.

Path planning is performed using a coarse-to-fine search algorithm that allows for unresolved path segments. In order to make it an anytime algorithm, we vary the abstraction level of the domain description. This allows the algorithm to find quickly a low quality plan and then repeatedly refine it by replanning a segment of the plan in more detail.

Figure 3 shows an instance of the problem domain. The thick lines represent obstacles. The particular map that is shown is the map that was available to the planner and includes sensing noise. The thin dotted lines show the abstract plan that was sent for execution. It includes a path from the lower left corner to the upper right corner. The thick dotted line shows the actual path followed by the robot when controlled by the above execution architecture. Note that no replanning was necessary since the recovery mechanism – an obstacle avoidance algorithm – was sufficient in this domain to "fix" any plan error. Finally, the path quality is the ratio between the length of the shortest path (that takes much longer to plan) and the actual path followed by the robot. An utility maximizing approach to control the planning component is

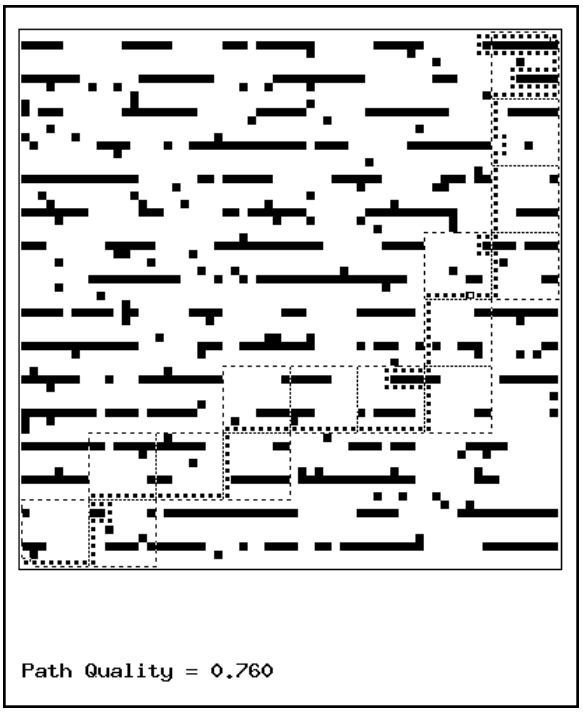

Figure 3: Execution of abstract plans with noisy sensors

described in [14].

3.2 A separate execution architecture

There are many advantages to specifying and developing a planning system with respect to a particular execution architecture rather than developing a complex planning system that combines planning and execution. The rest of this section summarizes the benefits of this approach.

Simplifying the design of planners

A well-defined execution architecture simplifies the development of the planner by narrowing down the scope of the planning task. Once the execution architecture is responsible to the safe operation of the system, the planner can operate under arbitrary assumptions. Those assumptions may lead to some probability of execution failure and replanning, but as long as they contribute to performance in the average case, they will have positive utility.

Facilitating "fair" comparison of planners

There is a large variability between planning systems in the type of monitoring and recovery mechanisms that are supported by the execution architecture. These mechanisms can have a major effect on the performance of the whole system. In particular, a "poor" planner can exhibit good performance when it is supported by a good execution architecture and vice versa. Therefore, analysis and comparison of planners that operate with the same execution architecture is much more objective and meaningful.

Modularity

The separation of the planner from the execution architecture leads to a modular approach with all the standard benefits that modularity introduces into system development.

Integration of different planning paradigms

A separate execution architecture facilitates integration of different planning paradigms that do not share the same knowledge representation and reasoning techniques. For example, one can integrate a more reactive rule-based planner with a deliberative search-based planner. As long as the planners can individually communicate with the execution architecture and can be individually controlled by a meta-level monitor, they can be integrated into one system.

Refining design choices

Separation of the planner from the execution architecture allows for a more refined set of design choices. For example, the principle of commitment to goals that has been examined recently by the planning community can be applied to the planner or to the execution architecture leading to totally different solutions and performance gains.

Uniform scalability

Another advantage of separation is its support of a uniform approach to system expansion. A planner and an execution architecture of one level become, together, the execution architecture of a higher level. The "primitive" actions of the high level system are the typical tasks that can be assigned to the low level system.

Robustness

Finally, separation of the planner and the execution architecture leads to a robust system since the execution architecture alone is responsible for the integrity of the system and for recovery from failures. Planners are normally much harder to debug and verify so it is advantageous to rely on the robustness of the execution architecture.

3.3 Planning as a goal refinement process

An important consequence of the above system decomposition is that a plan is no longer restricted to base-level actions. The information that a planner sends to the execution architecture is interpreted as advice that has value in the sense of statistical expectations, but it does not guarantee a solution to the problem[2]. The execution architecture can use the plan to select actions, but it is not committed to the plan. Planning becomes a continuous goal refinement process that translates a certain goal into a set of more operational, short term, obvious-to-achieve goals. Similar to the approach taken by some abstract planners, we view any plan consisting of a set of actions and ordering constraints as a set of *goals* to achieve the expected outcomes of those actions under the specified constraints. This approach leaves to the execution architecture the decision of what to do (possibly substituting one primitive action for another as needed). In other words, planning translates long term, high-level goals into more primitive, intermediate goals. This approach to planning as a goal refinement process has a number of advantages. First, it further simplifies the planning problem as its outcome is interpreted as intelligent advice. Second, it facilitates the notion of anytime planning since an iterative goal refinement process can be stopped at various points along the way and provide useful (but incomplete) advice to the execution architecture. Obviously, the advice provided after short deliberation may be less detailed and less helpful than a complete plan, but it may be the right thing to do under real-time pressure.

[2] A similar approach has been proposed by Agre and Chapman's plans-as-communications theory [1]. It has been experimentally verified in the ATLANTIS architecture [5].

4 History-based evaluation

We propose to analyze and evaluate alternative planning systems with respect to the domain *histories* that they induce and the utilities of such histories. A domain history is an extension of the notion of a state. It captures the change that occurred in the domain over time. Instead of considering individual states or possible worlds, one can analyze sequences of states or histories.

Histories facilitate a better formal treatment of the properties of planning systems. In particular, they allow the utility of the planner to be defined as a function of the actual solution cost in terms of time and other resources, the solution quality, and the rewards and penalties that affect plan execution. In some situations the quality of the solution is a property of the whole history rather than a property of the goal state alone. For example, when a robot is required to maintain at all times enough fuel to be able to return home. In addition, using histories one can reason about the utility (and other properties) of planning systems that operate over extended periods of time, possibly over unbounded time frames.

In evaluating a planner, we assume a given execution architecture, E, and a recovery mechanism, R. Moreover, we assume that the execution architecture may include some primitive mechanisms to achieve the goals, normally using a reactive approach that produces actions as an immediate response to sensory input. A history, H, of the physical world, W, is a sequence of world states that result from (a) change in the environment that is determined by the nature of the domain, and (b) the actions performed by the agent. We assume a given utility function $U(H)$ that determines the utility of histories.

The utility of a planner P at a given world state, S_0, is defined as:

$$U'(P|S_0) = \sum_H U(H) Pr(H|S_0, E[R,P]) - \sum_H U(H) Pr(H|S_0, E[R])$$

where $E[R]$ represents the operation of the execution architecture with the recovery mechanism and $E[R,P]$ represents the operation of the execution architecture with the recovery mechanism and the planner. The utility of a planner represents the expected gain of utility due to active planning over all the possible world histories. The utility of a planner in the domain D is the expected utility of the planner with respect to all the possible initial states:

$$U''(P) = \sum_{S_0} U'(P|S_0) Pr(S_0)$$

This approach to evaluation of planning systems has several advantages. First, it can be applied to any planning system and any planning paradigm. Second, it allows the researcher to pose some important questions in a well-defined, uniform way. For example, given a set of planners, one can check which one is better in a given domain. Given a set of domains,

one can determine which ones can be handled efficiently by a particular planner. All the "standard" properties of planners including computational complexity, correctness, completeness, and real-time operation are factored automatically into this evaluation formula. And most importantly, no particular planning paradigm, from situation calculus and theorem proving to decision-theoretic search, has an absolute advantage over the others. No paradigm can be ruled out a priori.

The main value of this evaluation approach lies in its generality and uniformity. It treats the planner exactly as a planner should be treated – as a performance improvement deliberation process. Some may question the applicability of this framework given the complexity of the representation and reasoning about histories. But the same argument can be applied to world states. Computer programs that represent and reason about states capture only a small fraction of the contents of a real world state. The same principle applies to world histories. In practice, a small set of variables may be sufficient to summarize the aspects of a history that determine the utility of planning. The boundary between states and histories may not be that obvious. In fact, some existing systems keep history features as part of their state representation.

5 Conclusion

We have examined the difficulties in evaluating and comparing planning systems. Since planning systems will continue to disagree on the fundamental role of the planner and on base-level evaluation metrics, we propose to compare systems in the context of a complete command, planning and execution environment. Moreover, we propose to compare planners with respect to a particular execution architecture since the execution architecture may have a major effect on the overall utility of a planner. Finally, we use the expected utility of domain histories as a unified evaluation criterion.

But as we noted earlier, evaluation of planning systems has another goal beyond summarizing base-level performance. The discovery of the laws of qualitative structure of planning systems remains the ultimate goal. A fair and unified comparison approach is only the first step.

Acknowledgements

The participants of the AAAI-94 Workshop on Comparative Analysis of AI Planning Systems have provided useful comments on an earlier draft of this paper. This work was partially supported by the University of Massachusetts under a Faculty Research Grant and by the National Science Foundation under grant IRI-9409827.

References

[1] P. E. Agre and D. Chapman. What are plans for? *Robotics and Autonomous Systems*, 6:17–34, 1990.

[2] P. R. Cohen. *Empirical Methods for Artificial Intelligence*, MIT Press, Forthcoming.

[3] R. E. Fikes and N. J. Nilsson. STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence*, Vol. 2, 1971.

[4] T. Dean, L. Kaelbling, J. Kirman and A. Nicholson. Planning with Deadlines in Stochastic Domains, *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 574–579, Washington, DC, 1993.

[5] E. Gat. Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots. *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 809–815, San Jose, California, 1992.

[6] K. J. Hammond. Explaining and Repairing Plans that Fail, *Artificial Intelligence*, 45:173–228, 1990.

[7] S. Hanks. Practical Temporal Projection, *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 158–163, 1990.

[8] S. Hanks, M. Pollack and P. R. Cohen. Benchmarks, Testbeds, Controlled Experimentation, and the Design of Agent Architectures, *AI Magazine*, 14(4):17–42, 1993.

[9] R. A. Howard. Information value theory, *IEEE Transactions on Systems Science and Cybernetics*, SSC-2(1):22–26, 1966.

[10] A. Newell and H. A. Simon. Computer science as empirical inquiry: Symbols and search, *Communications of the ACM*, 12:113–126, 1976.

[11] E. D. Sacerdoti. The Non-Linear Nature of Plans, *Proceedings of the International Joint Conference on Artificial Intelligence*, Tibilisi, USSR, 1975.

[12] A. Tate. Representing Plans as a Set of Constraints - the <I-N-OVA> Model, *ACM SIGART Bulletin*, Vol. 6, No. 1 (this issue), 1995.

[13] S. Zilberstein. Operational Rationality through Compilation of Anytime Algorithms, Ph.D. dissertation, (also Technical Report No. CSD-93-743), Computer Science Division, University of California, Berkeley, 1993.

[14] S. Zilberstein and S. J. Russell. Anytime sensing, planning and action: A practical model for robot control, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, pp. 1402–1407, 1993.

[15] S. Zilberstein and S. J. Russell. Optimal Composition of Real-Time Systems, *Artificial Intelligence*, forthcoming.