Toward a Foundation for Evaluating AI Planners

Technical Note 471

August 7, 1989

By: Nabil A. Kartam
Department of Civil Engineering
University of Maryland
and

David E. Wilkins
Artificial Intelligence Center
Computer and Information Sciences Division

This paper describes research peformed while the first author was at the Department of Civil Engineering, Stanford University. This research is funded by the Stanford Construction Institute, the Stanford Center for Integrated Facility Engineering, SRI International, and the National Science Foundation, grant MSME-87-16608.

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Contents

List of Figures

Abstract

There exists a large body of Artificial Intelligence (AI) research on generating plans, i.e., linear or nonlinear sequences of actions, to transform an initial world state to some desired goal state. However, much of the planning research to date has been complicated, ill-understood, and unclear [Cha87]. Only a few of the developers of these planners have provided a thorough description of their research products, and those descriptions that exist are usually unrealistically favorable since the range of applications for which the systems are tested is limited to those for which they were developed. As a result, it is difficult to evaluate these planners and to choose the best planner for a different domain. To make a planner applicable to different planning problems, it should be domain independent. However, one needs to know the circumstances under which a general planner works so that one can determine its suitability for a specific domain.

This paper presents criteria for evaluating AI planners; these criteria fall into three categories: (1) performance issues, (2) representational issues, and (3) communication issues. This paper also assesses four nonlinear AI planners (NOAH, NONLIN, SIPE, and TWEAK).

Acknowledgments

This paper describes research performed while the first author was at the Department of Civil Engineering, Stanford University. This research is funded by the Stanford Construction Institute, the Stanford Center for Integrated Facility Engineering, SRI International, and the National Science Foundation, grant MSME-87-16608. Their support is gratefully acknowledged. The authors also greatly appreciate the willingness of many AI planning researchers: Earl Sacerdoti, Ken Currie, Austin Tate, and David Chapman, who took the time to explain their complex planners. Raymond Levitt and Boyd Paulson contributed much by commenting on a draft of the paper.

1 Introduction

There exists a large body of Artificial Intelligence (AI) research on planning. Planning is defined as generating sequences of actions that will achieve given goals in a domain complex enough that the appropriateness and consequences of the actions depend upon the world states in which they are to be executed. While much planning research has been theoretical (Georgeff provides a good overview [Geo87]), there are a number of implemented planning systems. Choosing one of these systems to do planning in a specific domain is a difficult problem because much of the planning research to date has been complicated, ill-understood, and unclear.

To use a domain-independent planner for planning in a specific domain, one should make sure that the planner will meet the characteristics of this domain and hence one should know the conditions under which the planner works, the quality of plans produced, and the ease of formalizing problems for the planner. In this paper, we provide criteria for evaluating AI planners and then carry out an assessment of four AI planners based upon these criteria. We also provide an overview of some of the approaches for evaluating various domain-independent AI planners (literature review, communication with the developers of AI planners, experimentation), the problems encountered in such evaluation, and the ways to overcome them.

We have chosen four nonlinear, domain-independent AI planners to assess relative to our evaluation criteria: NOAH [Sac77], NONLIN [Tat77], SIPE [Wil88], and TWEAK [Cha87]. These four were chosen because they are widely known in the AI planning community, they have been well studied and documented, and their authors were willing to cooperate. Because the planners could not always be assessed based on a study of the published literature, it was necessary to rely on communication with their developers (via electronic mail, meetings, and correspondence). This need prevented a comprehensive survey of planning systems; in fact, some of our criteria can be hard to assess even by the author of a particular planning system.

NOAH, NONLIN, and SIPE are, respectively, further developments of what has been called the classical AI planning paradigm [Wil88], while TWEAK was one of the first implemented planners to have a solid theoretical foundation. NOAH, developed in 1975, was the first system to produce nonlinear, hierarchical plans and allow planning variables. While avoiding some of the weaknesses of linear planners such as HACKER [Sus75], NOAH could not avoid producing incorrect plans because of the complexity introduced by nonlinearity and planning variables. Sacerdoti therefore greatly expanded the idea of using techniques for modifying plans and referred to them as *plan critics*. Plan critics were used to solve the problems introduced by nonlinearity, e.g., that of concurrent actions interacting. However, there were still several limitations in NOAH. It did not backtrack, so could only find a solution if it happened to guess correctly at every choice point. The implementation of hierarchical planning was confusing and could produce incorrect plans. However, the major deficiency is the limited expressive power of NOAH's formalism. The system did not permit deduction of context-dependent effects, thus making actions unacceptably hard to describe. Even though planning variables were allowed, NOAH did not allow quantifiers over these variables or constraints on them.

NONLIN, written around 1977, further extended NOAH. The most important additions were backtracking and more extensive plan critics to modify already created plans. The more powerful critics required that the system keep track of why various goals had been inserted in the plan. While this permitted more problems to be solved, the expressiveness of the underlying formalism was not appreciably increased.

SIPE, developed during the early and mid 1980s, extends the expressiveness of the underlying formalism significantly. It has extended the classical AI planning paradigm further than any other system, being the first classical planner to employ a deductive causal theory to represent and reason about different world states, to reason about resources, to post and use constraints, and to do interesting replanning. SIPE can be run interactively or automatically, has a graphical interface, and has been applied in several problem domains. Unlike most AI planning research, the design of SIPE has taken heuristic adequacy (efficiency) as one of its primary goals.

TWEAK is a constraint-posting planner that employs dependency-directed backtracking. It is not as expressive as SIPE, e.g., it does not allow context-dependent effects or reasoning about resources. However, these restrictions stem from the primary design goal of TWEAK, which was to implement a planner with a sound theoretical foundation. TWEAK is clearly and precisely defined and, unlike the other three systems, is provably correct.

2 Issues in Evaluating AI Planners

One way to evaluate AI planners is by using information in the published literature. Two major types of information are available in the literature: descriptive information and analytic studies, although the latter are rare. The problem in using information in the published literature is that few of the developers of these planners have provided a thorough description of their research products, and their descriptions are usually unrealistically favorable. Of the dozen or more nonlinear planners developed and published since 1975, only one planner (TWEAK) has a solid theoretical foundation; the others employ heuristic techniques. Thus it is almost impossible to determine the mathematical properties of these other planners. Furthermore, few of these planners have been implemented on real problems, and thus the literature alone provides no information about their limitations in addressing real-life planning problems. Most AI planners developed thus far were not intended for use outside the research laboratory.

It is thus often necessary to contact the people close to the development of these planners to get more detailed information for evaluating them. One can communicate with the developers of AI planners through meetings, electronic mail, and/or correspondence. A second way to evaluate planners is through direct experimentation when the software is available. For example, experimentation can provide counterexamples to the claims made by authors of the planners.

In this paper, our evaluation of AI planners was based on criteria that fell into three categories: (1) performance issues, (2) representational issues, and (3) communication issues, as shown in Figure 1. Performance and representational issues are important from the point of view of the technical ability of the planner. They address the question: What are the kinds of problems that the planner is technically capable of solving and how well does it solve them? These issues received the most consideration in this research. Communication issues are less important but matter considerably in the case of an unsophisticated user. For each criterion, we then assessed four nonlinear AI planners: NOAH, NONLIN, SIPE, and TWEAK.

2.1 Performance Issues

Performance issues are the most important issues when evaluating a planner. They are at the heart of planning. In this research, we investigated five performance criteria: soundness, completeness, optimality, efficiency, and search control.

2.1.1 Soundness

A planner is said to be sound if all the plans that it produces are correct. A plan produced by a planner is correct if the execution of the plan, starting from the initial state of the world, transforms the world to a goal state. We start with a noncontradictory initial state of the world, and we are supposed to make our way to a noncontradictory final state of the world via various intermediate world states that will exist after planned actions are executed. If any of the intermediate world states produced by the planner are inconsistent, or the final world state is inconsistent, then the planner is unsound. Another way in which a planner can be unsound is if it produces a plan that does not result in the final desired world state. Such a planner fails because it produces a plan that does not achieve its goal. A formal analysis is needed to determine conclusively the soundness of a planner. However, it is sometimes possible to demonstrate that a planner is unsound through experimentation.

Lifschitz [Lif87] demonstrated that STRIPS does not necessarily produce correct plans and that STRIPS, therefore, is not a sound planner. STRIPS correctly solves only a restricted subset of problems allowed by its representation language. One reason why STRIPS is not sound is that it depends on the STRIPS assumption to deal with the frame problem. In STRIPS we have predicates that describe the initial state of the world and actions that modify this initial state by adding and deleting predicates from it. Unfortunately, STRIPS allows universally quantified

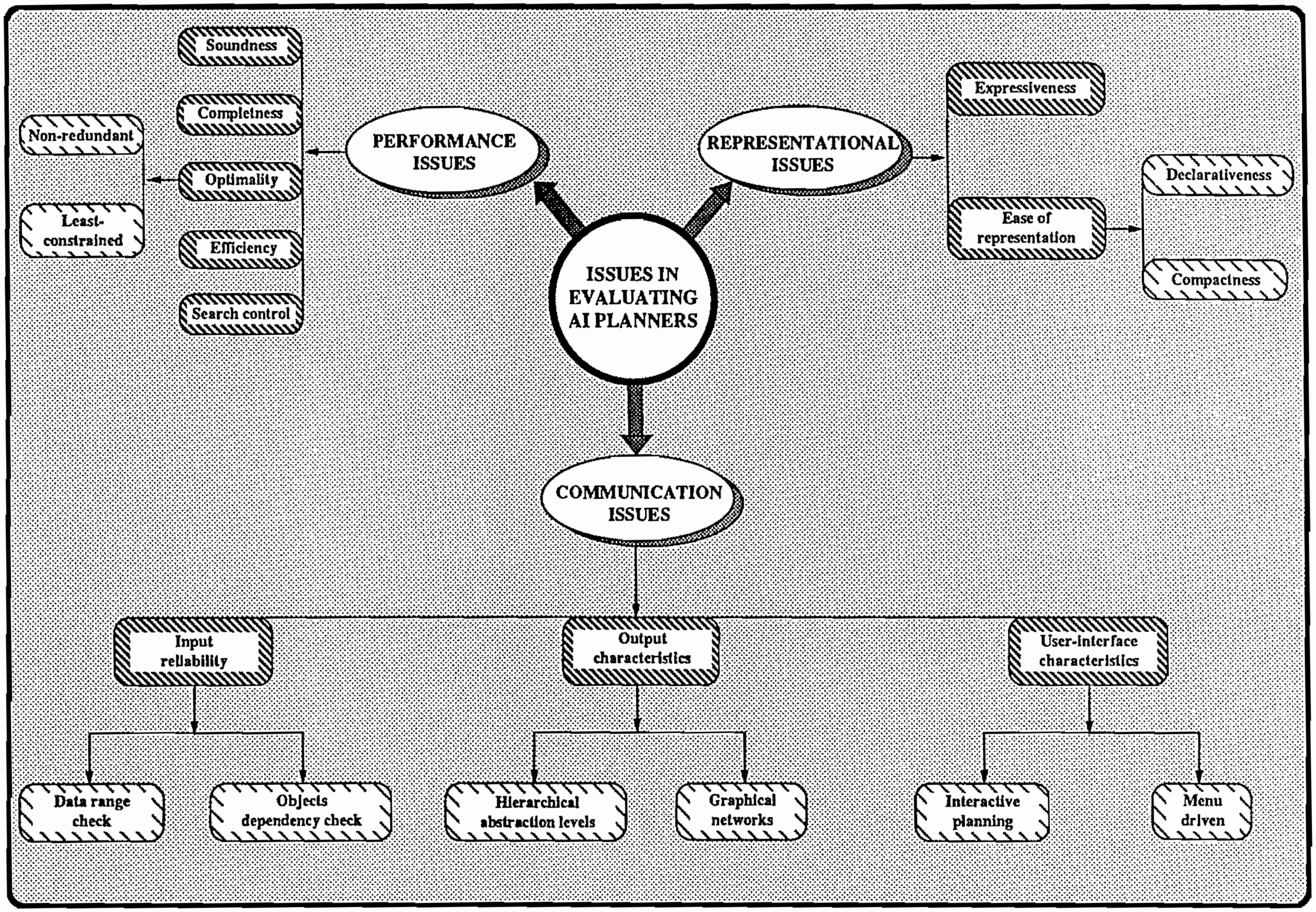

Figure 1: Criteria for evaluating AI planners

statements which cannot be properly updated by its add and delete mechanism. To make STRIPS sound is not difficult; it requires placing restrictions on the use of universally quantified statements. These restrictions have been adhered to in practice during use of STRIPS.

TWEAK, on the other hand, was specified rigorously and has been proven formally to be sound. NOAH, NONLIN, and SIPE do not share this property. In order to be practically useful, all three of these systems incorporate mechanisms and heuristics for circumventing exponential problems that result in invalid plans being temporarily produced [Wil88]. These systems rely on plan critics that check for and correct problems in these invalid plans. These critics are applied at certain intervals. One could view this as doing the work of enforcing validity every so often instead of at every primitive step. However, the heuristics, representations, and critics are complicated and not precisely defined, thus making any formal proof of soundness in these systems a huge undertaking. It would be our guess that none of these three systems are sound exactly as implemented, but that with some effort — including, perhaps, more precisely defining the representations — one could write a critic that analyzed a plan in enough depth to determine whether it was sound. Unsound plans would then trigger a failure just as any other critic which finds an insoluble problem; this would cause backtracking until the planner produced a sound plan.

In SIPE's case, the lack of a soundness proof has not been a problem in practice. The system has been applied successfully to several domains, including the blocks world, travel planning, movement of aircraft on a carrier deck, control of a mobile robot, generation of project plans for construction, and scheduling of process lines in a manufacturing environment [Wil89]. Even if a system is unsound, it can still be useful. For example, most Prolog implementations are unsound [Llo84] because, for reasons of efficiency, they omit the occur check in the unification algorithm.

2.1.2 Completeness

A planner is said to be complete if it will produce a correct plan whenever a feasible plan exists. This means that the planner must consider all alternatives that may lead to a solution. Choices that do not lead to a solution have to be excluded so that the retained plans are correct. There are three types of choice points while generating a plan: ordering links, variable instantiations, and operator choices.[1] The planner can choose to insert an ordering link between two previously unordered actions as part of its plan generation. When there is no concrete justification to linearize the two activities, it may be best to refrain from introducing an ordering link to preserve the freedom to execute them in any order, one before the other or both simultaneously.

[1]An operator is a planner's representation of an action and is used to transform one world state into another world state.

Similarly, it may be best to refrain from choosing variable instantiations until the planner knows which instantiation will allow it to arrive at the goal. The same is true of operator choices. A planner which avoids committing to these choices for as long as possible is referred to as a *least-commitment* planner. However, a planner must often make the above three choices without concrete justification so as to generate a plan; in such cases, the system must be able to backtrack and try other choices in order to be complete.

The next requirement to ensure completeness of a planner is that the planner not be trapped in infinite branches during the search while finite solutions still exist and have not been discovered. For example, if there are infinite branches in the search space, a planner that uses the pure depth-first search technique may not find the finite solution, even if one exists, because it is exploring an infinite branch. Therefore, almost all AI planners that use a depth-first search technique apply it to a certain depth limit, then backtrack. If they do not find any plans at the current depth, then they try again with a deeper depth limit. This technique is called depth-first iterative deepening (DFID) and is generally better than the breadth-first technique, which necessarily searches all plans; the DFID technique can often discover the right plan without looking at many others. As in the case of soundness, it may be possible to show that a planner is incomplete by providing a counterexample.

TWEAK is both sound and complete, given its representational constructs. It is currently the only nonlinear planner that has been proven to be complete. NOAH never attempted to be complete in that it does not keep track of alternative choice points. NOAH makes a commitment to operator choices and variable instantiations before the choice of operator or variable value is reduced to a single one, and it cannot backtrack. To be complete, a planner can choose an operator without saving a backtracking point when only a single operator is applicable, and a planner should instantiate a variable without saving a backtracking point when only one value of the variable is left. If several operators are applicable, then a complete planner must at least preserve all sound options. There may be cross dependencies between these options, in which case dependency-directed backtracking, which still preserves all the sound options but reduces the search space, is preferable.

SIPE is not complete. SIPE backtracks over operator choices, but not over ordering links and variable instantiations. However, SIPE's use of constraints generally allows the planner to delay variable instantiations until enough information is available. The user can control the insertion of ordering links in his domain specification [Wil89], so incompleteness has not been a problem in practice. NONLIN may be complete, but this would be difficult to prove because of the heuristic nature of the system. The crucial question is whether the critics try all possible choices of ordering links.

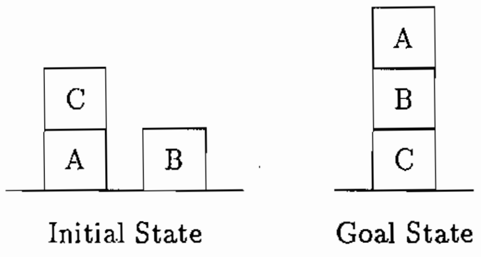

Figure 2: The Sussman Anomaly

2.1.3 Optimality

When considering optimality, we are concerned with some measure of the quality of a produced plan. We will refer to a final plan that reflects the constraints given by the domain- specific knowledge (with no unnecessary constraints) as a nonredundant, least-constrained, correct plan in this research. This plan is considered the optimal product of a planner. Such a plan allows the user the greatest freedom in selecting among the free choices without violating the correctness of the plan in any way, i.e., any set of actions that are in parallel can be executed simultaneously or in any order based on other constraints such as time, cost, preferences, resources, etc.

A plan is redundant if a set of actions and their associated links can be removed without preventing the plan from achieving its goal. Most AI researchers refer to a plan that is not redundant as an optimal plan, because no action stated in the plan needs to be undone, i.e., there is no unnecessary action in the plan. Most AI planners have been tested for what is called the Sussman Anomaly block-stacking problem, depicted in Figure 2. A nonoptimal plan of this problem produced by STRIPS consists of the following five actions:[2]

Unstack (c,a); Stack (a,b); Unstack (a,b); Stack (b,c); Stack (a,b).

On the other hand, the optimal plan of the Sussman Anomaly problem, which can be obtained from NONLIN, TWEAK, or SIPE, consists of only three actions:

Unstack (c,a); Stack (b,c); Stack (a,b).

There are three types of choice points while generating a plan: ordering links, variable instantiations, and operator choices. We will focus on ordering links here, because it is important not to overconstrain the ordering of a plan. For example, the duration of a plan is determined by the durations of the activities on the critical path of the plan. An overconstrained plan could result in the inclusion of an additional activity on this critical path and thus result in a longer

[2]Stack (x, y): means stack object x on object y. Unstack (x,y): means unstack object x from object y. ";" is the concatenation operator.

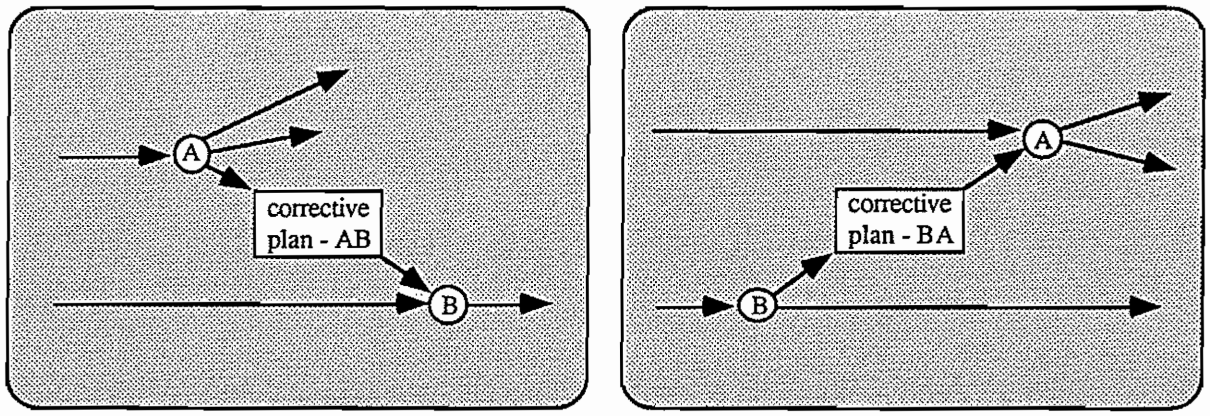

Figure 3: Alternative Orderings of Activities

project time. Thus, the plan that a planner produces should not have unnecessary ordering constraints. Unfortunately, the matter of not having unnecessary ordering constraints in a plan is subtle; the theory has not been formulated in graph theory or elsewhere. In addition to having a least-constrained plan, we need to make sure that we still have a correct plan. Any plan with actions that are unordered with respect to each other will often not be correct until the parallel interactions are analyzed and corrected [Wil88]. In this section, we discuss the difficult issues behind having a least-constrained, correct plan and how we deal with them.

One difficulty has to do with comparing alternatives for ordering activities. Suppose we have the situation shown in Figure 3. Two activities in two sub-plans interact. These two activities cannot be performed in parallel, but either one can come before the other, with a corrective sequence constraint interposed. Both plans are correct and least constrained, but we cannot say which has more parallelism. The duration of the activities could make the critical path of either of them shorter. Thus if we consider only the logical sequence of activities without their durations, comparison is difficult.

On the other hand, if we know that these two activities interact but that the only way to achieve one of them, say A, is to first achieve the other, say B, then it is correct to impose the order in which B must come before A. This particular constraint is necessary, and the plan produced remains correct and least constrained. Another type of difficulty arises when there are multiple instances of achieving an intermediate condition. Suppose that after activity 1 the predicates p, q, and r become true, after activity 2 the predicates s and t become true, after activity 3 the predicates p, r, and s become true, and after activity 4 the predicates q and t become true. Then activity 5 requires as preconditions that the predicates p, q, r, s, and t be true before execution. We could choose to make activity 5 come after activities 1 and 2 or make it come after activities 3 and 4 as shown in Figure 4. We cannot always judge which resulting

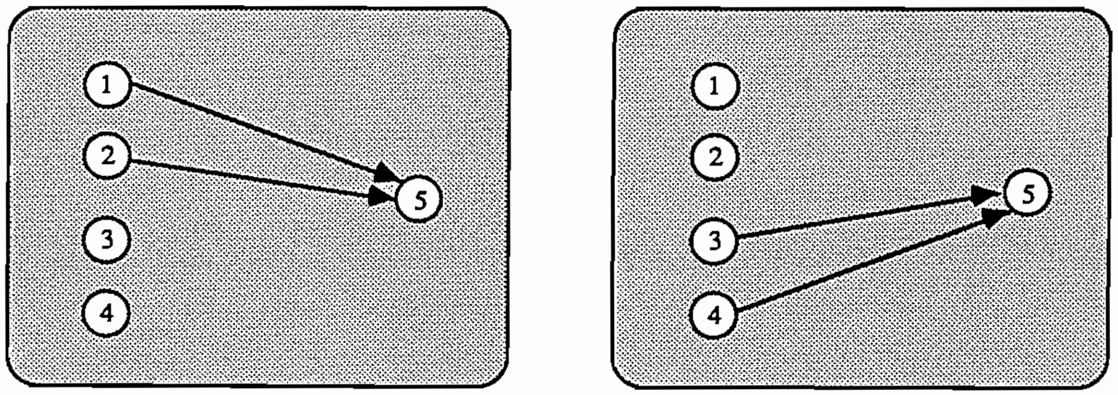

Figure 4: Selection of Ordering Links among Alternatives

plan has more parallelism.

Perhaps the most difficult problem behind producing a least-constrained, correct plan is dealing with interaction among activities that are unordered with respect to each other. If two actions of a plan are unordered, an interaction is defined to happen when a predicate occurs as a precondition of one action and the same predicate occurs as an effect of the other action. If the predicate has the same sign in both actions, the interaction may be helpful, otherwise it may be harmful. The two well-known methods for avoiding such interactions are choosing certain variable instantiations and imposing ordering constraints to achieve a least constrained, yet correct, plan.

For domains with no harmful interaction among subplans, it is possible to tell when a plan has no unnecessary ordering constraints. The algorithm we propose is as follows: Take a plan produced by the planner and check that the immediate predecessors of each activity x are the activities that make the unique predicates that are preconditions of x true (after instantiation). Given the conditions we have stated, there should be no other immediate predecessors.

The criterion of least-constrained and correct plans is difficult to evaluate. Experimentation may help to determine if particular problems exist with the implementation of a planner. But to show that a planner produces plans that are least-constrained and correct requires a careful and complete formal analysis that cannot be done with heuristic planning tools such as SIPE.

2.1.4 Efficiency

Most AI planners are implemented in different computer languages and run on different machines; therefore, an efficiency comparison of these varied planners is not really meaningful. For example, NOAH is implemented in QLISP; NONLIN is implemented in POP-2; TWEAK is implemented in dependency-directed LISP; and SIPE is implemented in Symbolics Common Lisp. Hence, we can base a simplistic evaluation only on gross measures, e.g., running time of a given software on a specific hardware. Some planners can, in practice, solve large problems in a few minutes, while other planners may take hours for simple problems and be prevented from addressing large problems because of their combinatorics. Thus, while efficiency results cannot be backed up with meaningful formal analyses, they are important in practice. Timing results can be obtained for specific planning problems, both small and large, by running them on implemented planners. This comparison will provide an idea of the practicality of using these planners based on the current state of the art. It will not show us worst-case behavior, only the performance of particular planners for a narrow class of problems. Unfortunately, the efficiency of planning algorithms has not been extensively reported in the literature.

SIPE is considerably more efficient than NOAH, NONLIN and TWEAK, based on system performance in solving actual problems. It is not possible to prove lower bounds on computation for any input because the representation provided by SIPE is powerful enough to express inefficient constraints. Furthermore, some algorithms inside the planner address exponential problems. In practice, efficient performance depends upon axiomatizing one's domain so as to take advantage of SIPE's heuristics and algorithms.

We briefly summarize some performance results reported for SIPE [Wil89]. Problems in an extended block-world (i.e., one that permits more than one block to be on top of another and poses problems such as "get some red block on top of some blue block") are solved by SIPE in 1 second. SIPE generated plans for an indoor mobile-robot world consisting of five rooms connected by a hallway, the robot itself, and various objects. The world is described by 222 predicate instances and 50 operators which use four levels of abstraction. The planner produces primitive plans that provide actual commands for controlling the robot's motors. This low level of abstraction requires the planner to generate hundreds of goal nodes to generate one plan, yet SIPE takes only about 20 seconds to formulate such a plan completely, or 9 seconds for an executable plan if the planner intermingles planning and execution. SIPE has also been used in a manufacturing domain on the problem of producing products from raw materials on process lines under production and resource constraints. The initial world state is described by 2100 predicate instances: to the best of our knowledge, this is considerably larger than any problem previously solved by an AI planning system. Producing a plan that schedules dozens of orders on six production lines with around 20 separate product runs, with their corresponding needs for different resources and materials, requires less than 4 minutes on a Symbolics 3645. To produce one such plan with no backtracking requires the generation of 1100 action and goal nodes (at all planning levels).

It is hard to make efficiency comparisons, because it appears that no AI planning systems have been tried on problems of similar complexity, in most cases because such problems cannot be effectively handled. NOAH, NONLIN, and probably TWEAK are not expressive enough to handle either the robot or manufacturing problems solved by SIPE. FORBIN [DFM88] would be expressive enough but is described as being slow on even simple problems. Drummond describes his NEWT planner [DC88] as not solving block-world problems, because of memory limitations, until a "temporal-coherence" heuristic was added to the system. With this heuristic, the system was able to solve simple block-world problems, allowing only one block on top of another, in "a matter of minutes." Both NEWT and FORBIN made concessions in their design to gain efficiency.

TWEAK did not have heuristic adequacy as one of its design goals and would probably be less efficient than FORBIN and NEWT. The planning algorithm of TWEAK requires that it perform breadth-first search, which implies a combinatorial explosion in the number of options investigated. While it is true that under the restricted representations of TWEAK it takes a polynomial amount of work to check if a given plan satisfies the modal truth criterion, TWEAK itself needs an exponential amount of work to find a valid solution.

There is one point of comparison outside of block-world problems. Both NONLIN and SIPE have been applied to a house-building problem. This problem is simple and was solved by NONLIN in 20 seconds [Tat76]. SIPE solves the problem in 5 seconds, but these times mean very little and can be ascribed to hardware differences. An important difference, however, is that SIPE's input is much more readable and concise than that of NONLIN; in fact, NONLIN had significantly less work to do to solve the problem because of the additional input information it required.

It is also useful to analyze how different planning algorithms affect efficiency. Many constructs, e.g., resource reasoning, hierarchical planning, and constraints, have been proposed as means to speed up planning. However, there is no theoretical justification for the claimed performance improvement. The combinatorial analysis of complicated planning systems, although theoretically possible, is practically unrealistic.

There is one technique, planning at differenct abstraction levels, for which experimental evidence exists for performance improvement. Experiments with SIPE in the mobile-robot domain indicate that planning hierarchically at different levels of abstraction did speed up the planning process significantly [Wil88].

Nonlinear planners using the least-commitment approach and plan critics can avoid searching much of the search space. These planners develop a network of partially ordered actions and goals and assume that unordered goals and actions can be achieved in parallel unless interactions dictate the introduction of ordering links. Thus, NOAH, NONLIN, and SIPE develop plans in a depth-first manner and use critics to detect interaction problems and then fix them by choosing variable instantiations and/or imposing ordering constraints. The critics guide the planner toward more beneficial directions of the search. A depth-first search with heuristic critics brings the planner to the solution much more quickly in the average case than TWEAK's breadth-first search does.

SIPE has other algorithms for aiding performance. It extends the critics approach by employing resource reasoning as part of the domain representation. In SIPE, if some of the variables associated with an action are declared as resources,[3] then the planner will not permit another unordered action to use those resources. This resource reasoning not only helps in making domain-specific knowledge easy to represent and understand, but it also enhances the efficiency of the system. Classical planners that use the critics approach, such as NOAH or NONLIN, will detect harmful interactions only after the entire plan has been expanded and the critics have been applied. Even then, conflicts between uninstantiated variables might not be detected, since only in an attempt to instantiate them would an actual conflict arise. In SIPE, it is known which resources an operator needs before the operator is applied, so conflicts can be detected even before the plan is expanded. This can result in choosing operators that do not produce conflicts, thereby pruning the search space and hence enhancing the efficiency of the planning system.

Another advantage in SIPE is its ability to separate side effects from main effects. Consequently, SIPE recognizes and resolves interactions dealing with the main effects of nodes, i.e., interactions between two side effects are ignored. This method greatly reduces the computational burden by not requiring the system to resolve conflicts that generally do not matter anyway.

2.1.5 Search Control

To be complete, a planner must search, although the search space can vary depending on the design of the planner. For example, the planner might search a space of partial plans or a space of world states. Figure 5 depicts a search space without making a commitment about what the space is.

[3]Resources in SIPE refer to reusable resources which cannot be shared among parallel activities.

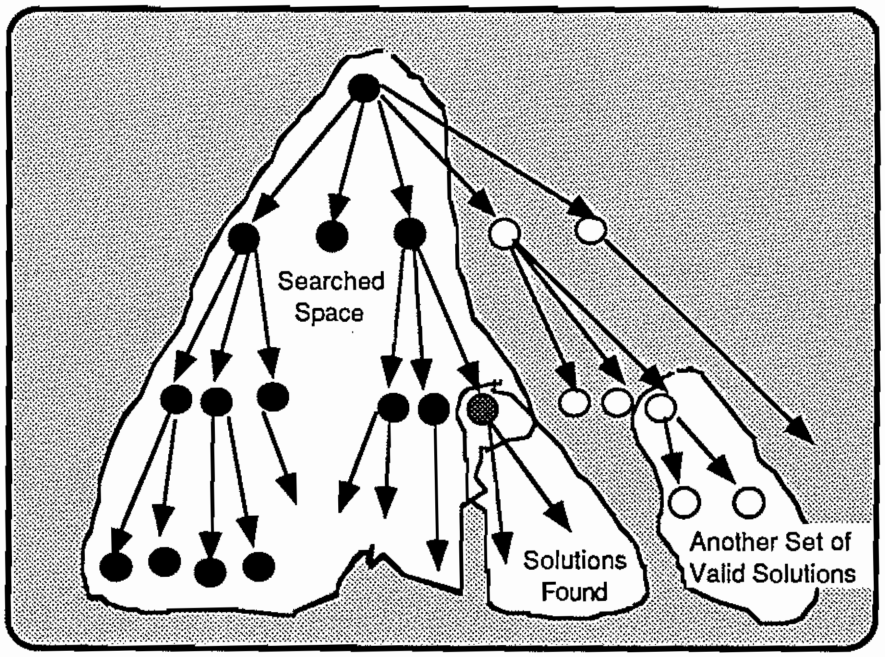

Figure 5: Search Space Tree

Search control is one of the least-understood aspects of domain-independent planning, and there has been little progress made in intelligent search control. It has been argued that intelligent search control will be domain specific [Wil88]. For example, consider the heuristic of using objects that are already being used in the plan. This is a fairly domain-independent concept that was incorporated into NOAH and other planners. However, this idea still involves domain knowledge. In the house-building domain, it is desirable to use the same piece of lumber to support both the roof and the sheetrock on the walls. But in another domain, this may not be a good strategy. On the space shuttle, one might want different functions to be performed by different objects so that the plan will be more robust and less vulnerable to the failure of any one object.

The four systems we are comparing differ markedly in their search control regimes. Many planning systems do not even search the important space of operator applications; rather, they rely on picking the correct operator every time a goal is expanded. Such planners include NOAH and FORBIN. NONLIN, TWEAK, and SIPE all use some form of backtracking.

Chronological backtracking often backtracks to choices that will also fail. Dependency-directed backtracking attempts to backtrack to a choice that had some role in the eventual failure of the proposed solution. The latter is beneficial when the search space is composed of independent subspaces, so that only one subspace needs to searched. If all parts of the search space are highly interdependent, dependency-directed backtracking may be more expensive because of the cost of maintaining dependencies. Thus, the usefulness of this technique depends again on the domain. TWEAK and NONLIN use dependency-directed backtracking. One problem with this technique involves the complexity of determining dependencies as the planner's representation becomes more powerful. With SIPE's complicated truth criterion, constraints, and deduced effects, the truth of a formula may depend on many things in the domain, and these dependencies may change every time a constraint is added to any variable in the system.

SIPE's only automatic search is a simple depth-first search, although the system does provide for the interleaving of planning and execution. This search has been built on basic mechanisms, contexts and choice points, that facilitate the ability to encode domain-specific search strategies. For example, it would be easy to implement a best-first search in SIPE if the user were to provide a function to evaluate the utility of a partial plan. SIPE also provides a powerful enough representation to encode search-control knowledge within operators. Unlike its predecessors, SIPE is designed to allow interaction with users throughout the planning and plan execution processes. The user is able to watch and, as desired, to guide or control the planning process.

2.1.6 Summary of Performance Issues

In summary, performance issues include soundness, completeness, optimality, efficiency, and search control. TWEAK has been proven to satisfy the first three criteria; however, it is too inefficient to solve practical problems. The problem-solving approaches of NOAH, NONLIN, and SIPE are based on heuristics. Therefore, they are more efficient but are not provably sound or complete. Little progress has been made in search control for domain-independent planning, in part because searching algorithms and heuristics need to be domain dependent.

2.2 Representational Issues

A major concern during the development of any planner must, of course, be the representational capacity of the planner, i.e., the question of what kind of planning problems can be represented in the system. Among the types of knowledge about which a planner might be concerned are knowledge about objects, time, and actions, and the constraints on them and among them. It would be ideal if a planner could represent any planning problem whatsoever, but that is not a practical objective, because greater representational power leads to a concomitant increase in the complexity of planning. Clearly, there is a tradeoff.

We will discuss two types of representational issues: expressiveness and ease of representation. Expressiveness delimits the fundamental ability of a system; e.g., first-order predicate calculus can describe a wide class of problems, but it may be awkward to describe them in this manner or to provide an algorithm for solving the problem described in this manner. Different planning problems could require different types of representational constructs (such as frames, rules, and logic). Representational constructs and planning algorithms interact. It is possible for a planner to become unsound if the representational range is increased, as when universally quantified statements are permitted in STRIPS. Conversely, if we limit the representational range, a planner may become sound. In some cases, we can determine the presence of specific representational constructs by reviewing the literature, although we may not be able to determine their effects on either planner completeness, correctness, or efficiency.

Ease of representation is not a minor issue; it is typically difficult to attain. There are various considerations with respect to ease of a representation. First, the representation ought to match the way the user thinks about the domain as closely as possible. This means that the user has to do little translation from the way he or she thinks about the problem to the input required. A declarative representation is almost certainly required to aid the user in preparing, understanding, and modifying his or her input in a natural language format. Second, the representation should be compact. A useful measure of compactness is hard to define, but a simple metric is the number of symbols used. It is clear, however, that a system which requires input of information that can be deduced, or requires the same information in different forms, is not compact. Only relative comparison of inputs for the different planners would be meaningful. One method of improving the compactness of representation is an object frame hierarchy. In this hierarchy, properties of objects can be inherited and do not need to be entered again, thus reducing encoding redundancy. This means that less input is needed to describe the objects of planning. Another method of reducing the amount of input is deduction. Examples are the deductive operators in SIPE. These are essentially rules for deducing that additional things are true about the world from facts that are true now. The user thus has less to worry about in providing complete and consistent input, and hence the reliability (a communication issue) is improved as well. Neither the object hierarchy nor the deductive operators necessarily increase the representational capacity of the planner, but both increase representational convenience and clarity. Their presence in a planning package can easily be determined from a review of the literature.

In NOAH, domain knowledge had to be encoded procedurally in a LISP-like language called "SOUP" (Semantics Of User's Problem). Similarly, NONLIN has a "Task Formalism," which allows for a declarative representation of domain knowledge. In SIPE, the user can represent domain-specific knowledge in a declarative, hybrid formalism, i.e., knowledge with a static nature (invariant) in frames and that of dynamic nature (changeable) in first-order logic. In discussing

Characteristics	NOAH	NONLIN	SIPE	TWEAK
Expressiveness:				
1. the use of variables (unspecified objects)	x	x	x	x
2. ability to specify constraints on objects			x	
3. ability to specify contingent subnetworks			x	
4. ability to specify preferences in operators		x	x	
Ease of representation:				
1. declarative input specifications		x	x	x
2. use of inheritance to facilitate input			x	
3. use of deduction			x	

Table 1: Comparison of Representational Issues among Four AI Planners

knowledge representation, Parsaye and Chignell stated in their book *Expert Systems for Experts*: "Predicate logic provides a powerful system for reasoning and once combined with frames, allows a flexible method of knowledge representation and inference." [PC88], p. 161. In addition, SIPE provides a means for stating constraints over variables, which permits partial description of objects. TWEAK, NOAH, and NONLIN do not allow the specification of similar constraints on variables.

Table 1 compares four AI planners (NOAH, NONLIN, SIPE, and TWEAK) with regard to desirable representational characteristics.

2.3 Communication Issues

Communication issues involve user interaction and analysis of the output of the planning process. Below, we discuss the needs of the user in communicating his problem in the input language of the planner and the computer system's capability to present the output information needed by the user. It is important that the user be able to formulate his problem easily and nonredundantly.

2.3.1 Input Reliability

An important communication issue is reliability of user input. Given that users can make mistakes, it is important that the input be checked, or easily can be checked, for errors. Input that has a nonchecked redundancy could result in the introduction of errors during entry as well as during modifications. One characteristic of reliability is that the data values fall into the correct range. Another characteristic is that the dependencies among the objects should be preserved. To ensure this preservation of dependencies, a check should be made whenever possible. For example, in the construction domain, usually the number of footings should not exceed the number of columns and the number of decks should not exceed the number of beams. Such semantic conditions for error-checking represent a layer of additional knowledge. Most current planner front-ends (including NOAH, NONLIN, SIPE, and TWEAK) do not have facilities to check the correctness of input knowledge, nor do they have truth maintenance checks on the consistency of the input world models. Consistency checks would require a full theorem prover, but other checks, e.g., type checking for the arguments of predicates, could be added inexpensively.

2.3.2 Output Characteristics

The utility of a planner depends strongly on the quality of its output. If the output needs extensive transformation to alllow interpretation, then it is less useful. The language of the output should be the language of the user whenever possible, for the reasons discussed in the previous section. Spurious items should be removed or explained. For example, construction plans can involve hundreds of resources and thousands of activities. Therefore, it is important for the planner to have an output interface that:

- Provides the user with an abstract overview of the plan.
- Allows the user to see the portion of the plan that is relevant.
- Presents the plan in a way that readily allows the user to interpret it.

The first two criteria can be met by adopting a hierarchical planning approach that represents both objects and actions of a domain at different abstraction levels. Of our four planners, SIPE is the only one with a graphical interface that meets these criteria. Regarding the third criterion, one of the best-known ways of presenting a plan is a graphical network of activities in which nodes and their logical relationships flow from left to right (or top to bottom). It would be helpful if the grain size of the displayed plan could be adjusted and the user could browse over the plan. Transformations and user-driven modification of the displayed plan are most desirable. In this way, the user can adjust the display according to his needs. The output of a planner can be presented in various ways. The most widespread is a plan listing of actions and their sequential relationships. Another way is with a directed activity graph. In one type of activity graph termed a precedence diagram, the activities to be performed are given as nodes and the

Characteristics	NOAH	NONLIN	SIPE	TWEAK
Hierarchical planning:				
1. multiple abstraction levels	x	x	x	
Graphical networks:				
1. plan listings	x	x	x	x
2. activity graph display (a network)			x	
3. concurrent alternative solutions			x	
4. contingent subnetworks display			x	
5. execution of an existing plan			x	

Table 2: Comparison of Output Characteristics among Four AI Planners

ordering links as edges. The reverse approach, termed an arrow diagram, places activities on the arcs and labels the nodes as events.

Table 2 compares four AI planners (NOAH, NONLIN, SIPE, and TWEAK) with regard to desirable output characteristics.

2.3.3 User Interface Characteristics

User-friendliness is a subjective measure: it depends on the sophistication of the user. However, we know that a graphical system is often more user-friendly than a nongraphical one. The less the user needs to learn and remember in order to operate the system, the more user friendly it is; therefore, a menu-driven system is often more user-friendly than a command- driven system. A shorter setup time is also desireable. Since the 1960s, many researchers have realized that man-machine interactive systems might help overcome some of the shortcomings of traditional planning systems [Pau72]. The user interface was often neglected in AI planners. Most AI planners, including NOAH and TWEAK, are run primarily in batch mode without user interaction. Tate recognized the importance of incorporating an interactive interface in NONLIN; he stated: "At present the system is run mostly 'stand-alone' without user interaction. However, for the use to which it will be put in the planning, a joint AI/OR approach project, it is envisaged that a user will aid in the choice of nodes to expand and linearizations to make, etc." [Tat76], p. 26. SIPE has an elegant interactive man-machine interface: the user can watch and, if he desires, guide or control the planning process through mouse-sensitive menu selection to aid in choosing

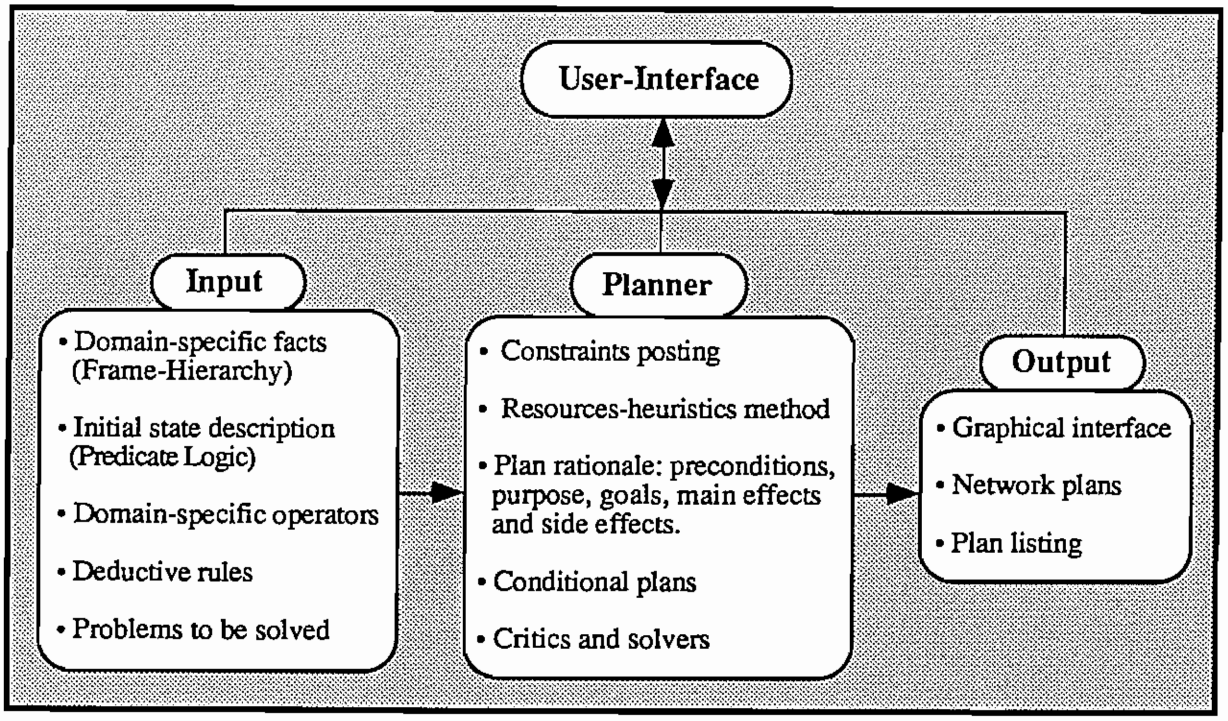
User-Interface
Input
• Domain-specific facts (Frame-Hierarchy)
• Initial state description (Predicate Logic)
• Domain-specific operators
• Deductive rules
• Problems to be solved
Planner
• Constraints posting
• Resources-heuristics method
• Plan rationale: preconditions, purpose, goals, main effects and side effects.
• Conditional plans
• Critics and solvers
Output
• Graphical interface
• Network plans
• Plan listing

Figure 6: SIPE's Models: Interactive-Heuristic AI Planner

nodes to expand with a specific operator, orderings to make, instantiations to specific objects, etc. Such user interaction with the planner permits the solving of larger problems than would have been possible with fully automatic planning techniques and is also useful for debugging. Figure 6 shows SIPE's improvement over previous AI planners.

3 Conclusion

We presented a set of general criteria for evaluating AI planners to verify the usefulness of a planner for a specific domain. These criteria, shown in Figure 1, cover the important aspects of a planning system from the standpoint of choosing one for an application. The significance of each criterion depends on the characteristics of the specific domain in which the planner will be used. For example, in the construction domain, it is important to obtain plans in their least-constrained format, but it is less crucial to obtain such plans quickly, i.e., efficiency is not of the highest priority. Similarly, simple domains such as the block world may not stress expressiveness or output characteristics.

We assessed four nonlinear domain-independent planners (NOAH, NONLIN, SIPE, and TWEAK) for each criterion based on a study of the published literature and communication with their developers. We did not attempt an in-depth formal analysis of these general AI planners because (1) of the more than one dozen nonlinear planners developed and published to date, only one planner (TWEAK) has a solid theoretical foundation, and (2) the formal analysis of these complicated AI planners is unrealistic in practice.

A quick summary of our assessment is in order. NOAH fell short of the other planners along every criteria; this is not surprising as it was the first nonlinear planner implemented. It is not sound or complete, does not search, is not very expressive, and has poor input-output characteristics. NONLIN improves the input characteristics of NOAH, provides a backtracking search, and slightly improves expressiveness over NOAH. SIPE is much further advanced than NOAH, NONLIN, and TWEAK in the areas of efficiency, expressiveness, ease of representation, input-output characteristics, and user interface characteristics. This is not surprising since NOAH and NONLIN are predecessors of SIPE, and TWEAK did not have these characteristics as design goals.

TWEAK involves the development of logical formalisms that are theoretically sound and complete but that have little hope of expressive or efficient implementation for real-life planning problems. TWEAK is the only one of the four planners that is provably sound and complete. Its development contributed greatly to the understanding of planning. Among the four planners SIPE represents the state of the art in engineering while TWEAK represents the state of the art in the formal theory of planning.

Because of the difficulty in determining how planning systems meet our criteria, e.g., it cannot be done by literature review only, we have limited our analysis to four well-known planners, although other more recent planners have been mentioned. FORBIN is probably more expressive than SIPE but is considerably less efficient. NEWT provides new mechanisms for domain-independent search control, but even with these mechanisms, it is also considerably less efficient than SIPE. Neither of these systems has been reported to have a graphical interface or a graceful user interface. O-PLAN [CT85] is a descendant of NONLIN that addresses many of the latter's weaknesses. The O-PLAN implementation was not ready for experimentation when this research was conducted, but it appears to be comparable to SIPE in many areas. While the literature provides no data on O-PLAN's efficiency, it appears to be more expressive than SIPE and to have comparable output characteristics, user interface characteristics, and ease of representation.

References

[Cha87] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[CT85] K. Currie and A. Tate. O-plan: control in the open planning architecture. Technical Report AIAI-TR-12, University of Edinburgh, Edinburgh, Scotland, 1985.

[DC88] M. Drummond and K. Currie. Exploiting temporal coherence in nonlinear plan construction. *Computational Intelligence*, 4(4):341–348, 1988.

[DFM88] T. Dean, R. J. Firby, and D. Miller. Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 4(4):381–398, 1988.

[Geo87] M.P. Georgeff. *Planning.* Annual Reviews, Inc., Palo Alto, California, 1987.

[Pau72] B.C. Paulson Jr. Man-computer concepts for planning and scheduling. *ASCE Journal of the Construction Division*, 98(2), 1972.

[Lif87] V. Lifschitz. On the semantics of STRIPS. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop.* Morgan Kaufmann, San Mateo, California, 1987.

[Llo84] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, New York, 1984.

[PC88] K. Parsaye and M. Chingell. *Expert Systems for Experts.* John Wiley, Inc., New York, 1988.

[Sac77] E. D. Sacerdoti. *A Structure for Plans and Behaviour.* Elsevier, North Holland, New York, 1977.

[Sus75] G.J. Sussman. *A Computer Model of Skill Acquisition.* Elsevier, North Holland, New York, 1975.

[Tat76] A. Tate. Project planning using a hierarchical nonlinear planner. Department of Artificial Intelligence Report 25, Edinburgh University, 1976.

[Tat77] A. Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–893, Cambridge, Massachusetts, 1977.

[Wil88] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[Wil89] David E. Wilkins. Can AI planners solve practical problems? Technical Report 468, Artificial Intelligence Center, SRI International, 333 Ravenswood Avenue, Menlo Park, California, July 1989.