To appear in *Performance Metrics for Intelligent Systems Workshop (PerMIS '04).*

Measuring the Performance of Automated Planning Systems

Dana Nau
Department of Computer Science
and Institute for Systems Research
University of Maryland,
College Park, MD 20742, USA
email: nau@cs.umd.edu

Malik Ghallab
LAAS-CNRS
7, Avenue du Colonel Roche
31077, Toulouse, cedex, France
email: Malik.Ghallab@laas.fr

ABSTRACT

In this paper, we describe existing performance measures for automated planning algorithms, and discuss the limitations and biases inherent in those performance measures. We point out the importance of developing a performance measure that explicitly the restrictive assumptions on which a planning algorithm depends, and we propose a composite performance measure based on three factors:

- the scope of the planning algorithm: which set of restrictive assumption are needed and which can be lifted,
- the control knowledge and tuning required for each planning domain,
- the size of the problems that can be solve in a reasonable amount of time in each area of its scope (i.e., for each combination of relaxed assumptions it can handle).

KEYWORDS: automated planning, AI planning, performance measurement

1. INTRODUCTION

Great strides have been made in automated planning during the past few years, and the technology is becoming mature enough to be useful in a variety of demanding applications, ranging from controlling space vehicles such as Deep Space 1 [6] to playing the game of bridge [31]. Successes such as these are creating a great potential for synergy between theory and practice: observing what works well in practice can lead to better theories of planning, and better theories can lead to better performance in practical applications.

Despite this potential, there currently is a substantial gap between theoretical and application-oriented work. The theoretical work tends to be rather narrow in scope, focusing on highly restricted cases such as *classical planning*, with the most common performance measure being the speed of the planner's combinatorial search. The application-oriented work generally depends on *ad hoc* application-specific programming efforts, search techniques, and measures of performance.

For most planning systems, presentations of the planning algorithm may discuss some of the assumptions and restrictions explicitly—but usually the algorithm will also depend on additional assumptions and restrictions that are tacit in the representation rather than explicit. As a consequence, it is often very difficult to judge whether a planning algorithm can be useful for real-world problem solving, and it is often even more difficult to tell whether an application-specific planning algorithm can be generalized to work in anything other than the specific application for which the algorithm has been written. *Better ways are needed to judge the scope and generalizability of planning algorithms and techniques.*

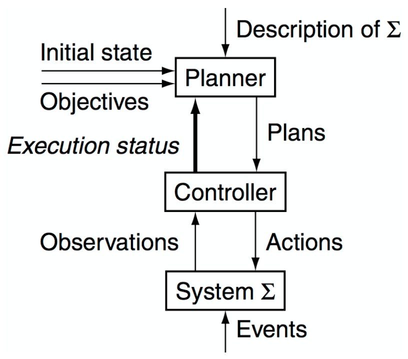

Figure 1: A simple conceptual model for planning. Σ is a state-transition system, as described in the text.

As a step toward meeting that need, we describe a general conceptual model for planning, and use it to classify and discuss the kinds of restrictive assumptions that are often made in automated planning research. We believe that with suitable refinement, such a classification will provide a useful performance measure for automated planning algorithms, by providing a way to give a clearer account of what restrictions a planning algorithm requires.

2. CONCEPTUAL MODEL FOR PLANNING

Since planning is concerned with choosing and organizing actions for changing the state of a system, a conceptual model for planning requires a general model for a dynamic system. This model, shown in Figure 1, includes three components:

- A *state-transition system* Σ that evolves as specified by its state-transition function γ, according to the events and actions that it receives. Σ includes a set S of states, a set A of actions, a set E of events, and a state-transition function $\gamma : S \times A \times E \to 2^S$.

- A *controller*. Given as input the state s of the system (or more generally, some observations that give partial knowledge of the current state), the controller provides as output an action a according to some plan.
  $\eta : S \to O$ that maps S into some discrete set $O = \{o_1, o_2, \ldots\}$ of possible observations. The input to the controller is then the observation
- A *planner*: given as input a description of the system Σ, an initial situation and some objective, it synthesizes a plan for the controller in order to achieve the objective.

The planner's objective can be specified in several different ways.

1. The simplest specification consists of a *goal state* s_g or a set of goal states S_g; the objective is achieved by any sequence of state transitions that ends at one of the goal states.

2. More generally, the objective is to satisfy some condition over the sequence of states followed by the system; for example, one might want to require states to be avoided, states that the system should reach at some point, and states that it should stay in.

3. An alternative specification is through a utility function attached to states, with penalties and rewards, the goal being to optimize some compound function of these utilities, e.g. sum or maximum, over the sequence of states followed by the system.

4. Another alternative is to specify the objective as tasks that the system should perform. These tasks can be defined recursively, as sets of actions and other tasks.

3. RESTRICTIVE ASSUMPTIONS

The conceptual model in the last section was deliberately quite general, in order to provide a starting point for describing a number of restrictive assumptions:

- **Assumption A0 (Finite Σ).** The system Σ has a finite set of states.
- **Assumption A1 (Fully Observable Σ).** The system Σ is *fully observable*, i.e., one has complete knowledge about the state of Σ; in this case the observation function η is the identity function.
- **Assumption A2 (Deterministic Σ).** The system Σ is *deterministic*, i.e., for every state s and event or action u, $|\gamma(s,u)| \leq 1$. If an action is applicable to a state, its application brings a deterministic system to a single other state. Similarly for the occurrence of a possible event.
- **Assumption A3 (Static Σ).** The system Σ is *static*, i.e., the set of events E is empty. Σ has no internal dynamics; it stays in the same state until the controller applies some action.[1]
- **Assumption A4 (Attainment Goals).** The only kind of goal is an *attainment goal*, which is specified as an explicit goal state s_g or a set of goal states S_g. The objective is to find any sequence of state transitions that ends at one of the goal states. This assumption excludes, for example, states to be avoided, constraints on state trajectories, and utility functions.
- **Assumption A5 (Sequential Plans).** A solution plan to a planning problem is a linearly ordered finite sequence of actions.
- **Assumption A6 (Implicit Time).** Actions and events have no duration, they are instantaneous state transitions. This assumption is embedded in state-transition systems, a model that does not represent time explicitly.
- **Assumption A7 (Off-line Planning).** The planner is not concerned with any change that may occur in Σ *while* it is planning; it plans for the given initial and goal states regardless of the current dynamics, if any.

The simplest case, *classical planning*, combines all eight restrictive assumptions: complete knowledge about a deterministic, static, finite system with restricted goals and implicit time. Here planning reduces to the following problem:

> Given $\Sigma = (S, A, \gamma)$, an initial state s_0 and a subset of goal states S_g, find a sequence of actions $\langle a_1, a_2, \ldots, a_k \rangle$ corresponding to a sequence of state transitions $(s_0, s_1, \ldots, s_k)$ such that $s_1 \in \gamma(s_0, a_1)$, $s_2 \in \gamma(s_1, a_2)$, ..., $s_k \in \gamma(s_{k-1}, a_k)$, and $s_k \in S_g$.

Since the system is deterministic, if γ is applicable to s then $\gamma(s,a)$ contains one state s'. To simplify the notation, we will say $\gamma(s,a) = s'$ rather than $\gamma(s,a) = \{s'\}$. For this kind of system, a plan is a sequence $\langle a_1, a_2, \ldots, a_k \rangle$ such that $\gamma(\gamma(\ldots \gamma(\gamma(s_0, a_1), a_2), \ldots, a_{k-1}), a_k)$ is a goal state.

The assumption about complete knowledge is needed only at the initial state s_0, because the deterministic model allows all of the other states to be predicted with certainty. The plan is unconditional, and the controller executing the plan is an *open-loop controller*, i.e., it does not get any feedback about the state of the system.

Classical planning may appear trivial: planning is simply searching for a path in a graph, which is a well understood problem. Indeed, if we are given the graph Σ explicitly then there is not much more to say about planning for this restricted case. However, it can be shown [14] that even in very simple problems, the number of states in Σ can be many orders of magnitude greater than the number of particles in the universe! Thus

[1]The name of this assumption is inaccurate, because the plan is intended precisely to change the state of the system. What the name means is that the system remains static *unless controlled transitions take place.*

it is impossible in any practical sense to list all of Σ's states explicitly. This establishes the need for powerful *implicit* representations that can describe useful subsets of S in a way that both is compact and can easily be searched.

The simplest representation for classical planning is a *set-theoretic* one: a state s is represented as a collection of propositions, the set of goal states S_g is represented by specifying a collection of propositions that all states in S_g must satisfy, and an action a is represented by giving three lists of propositions: preconditions to be met in a state s for an action a to be applicable in s, propositions to assert and propositions to retract from s in order to get the resulting state $\gamma(s, a)$. A plan is any sequence of actions, and the plan solves the planning problem if, starting at s_0, the sequence of actions are executable, producing a sequence of states whose final state is in S_g.

A more expressive representation is the *classical representation*:[2] starting with a function-free first-order language L, a state s is a collection of ground atoms, and the set of goal states S_g is represented by an existentially closed collection of atoms that all states must satisfy. An operator is represented by giving two lists of ground or unground literals: preconditions and effects. An action is a ground instance of an operator. A plan is any sequence of actions, and the plan solves the planning problem if, starting at s_0, the sequence of actions are executable, producing a sequence of states whose final state satisfies in S_g. The *de facto* standard for classical planning is to use some variant of this representation.

4. Classical Planning Versus Planning Applications

For nearly the entire time that automated planning has existed, it has been dominated by research on classical planning. For a while, the dominance was so complete that the term "domain-independent planning system" was used to refer to planning systems whose scope was that of classical planning, as if classical planning were capable of representing all possible planning domains.

In reality, it can be proved [14, Chapters 1–3] that classical planning systems are restricted to a very narrow class of planning domains. This class excludes most problems of practical interest, because most practical planning problems do not satisfy the restrictions of classical planning. Here are a few examples:

- **Process planning for machined parts.** Process planning is an important manufacturing task, and many millions of R&D dollars have been spent to try to automate it [23]. The state space consists of the possible states of the workpiece, including the workpiece geometry and various other parameters. The action space consists of the possible ways to modify the workpiece using machining operations. Both spaces are effectively infinite [17]. The actions have nondeterministic outcomes due to random variations—but in process planning the outcomes usually are approximated deterministically by the use of machining tolerances [9]. The planner must consult with CAD modelers to reason about the workpiece geometry, and must query databases to obtain information about the available machines, tooling, fixturing, and process parameters. With the exception of a few specialized process-planning tasks such as sheet-metal bending [16] and NC toolpath generation [28], generative process planning tools do not work very well and have not achieved significant industrial use. By far the most widely used process-planning tools are those that provide information to help expert humans do the process planning. Other approaches, e.g., [3, 8], illustrate the same trend for planning in other manufacturing applications.
- **Planning declarer play in bridge.** At the beginning of play in a bridge hand, the declarer (the player who chose the trump suit) needs to develop a plan for how to play the hand. The outcomes of the declarer's actions are uncertain, due both to uncertainty about how the opponents will respond and uncertainty about how they *might be able* to respond (since the declarer does not know which opponent holds which cards). A game tree containing all of the possibilities would have about 2.3×10^{24} leaf nodes on the average and about 5.6×10^{44} in the worst case [30, p. 226]. Since most bridge games are over in just a few minutes, it would not be feasible to explore any significant fraction of such a game tree. Instead, techniques have been developed that use various combinations of game-tree search, Monte Carlo simulation, and reasoning about possible strategies [12, 15, 31]. The resulting programs can play better than the average human bridge player, but not as good as the best human players.
- **Ship-movement planning.** Planning the movements of ships is important both commercially and militarily [11]. The state space and action space are effectively infinite: states include positions and velocities of ships, and actions correspond to movements of the ships along various routes. Since movements of different ships may occur concurrently, it is important to make sure they do not interfere with each other. The outcomes and durations of the actions cannot be known with certainty, because of factors such as weather, currents, and the behavior of the ships' operators. Elaborate simulation tools are available to aid in planning ship movements but the planning is still done manually [1]. Similarly, other transportation-planning applications, such as for railways [2], have focused on interactive approaches for planning.

Many other examples could easily be cited; see for example the PLANET repository's "Real-World Planning and Scheduling page" at $\langle$http://vitalstatistix.nicve.salford.ac.uk/planet2$\rangle$.

[2]This has also been called *STRIPS-style* representation), after an early planning system [27] that used a similar representation scheme.

5. Existing Performance Measures

In this section, we do a quick survey of existing performance measures, and draw several conclusions about the limitations of those measures.

5.1. Survey

Performance measures for classical planners. The existence of a standard representation scheme for classical planning has made it relatively easy to develop large collections of planning problems on which different planning algorithms can be compared. In the three international planning competitions that have occurred so far [24, 4, 22], many hundreds of classical planning problems have been generated, from about fifteen different planning domains.[3] The most common performance measures have been *success rate*, *speed*, and *solution size*, i.e., the fraction of problems solved, the CPU time needed to solve them, and the size of the solution found (the latter two are normally measured as a function of the problem size). From these measures, one can get a rough idea of the size of the problems that a planner can solve in a reasonable amount of time.

A partial generalization. The 2002 International Planning Competition [22] included several collections of planning problems that did not satisfy all of the restrictions of classical planning. In these problems, Restrictions A0, A4, and A6 were weakened, by generalizing the planning language to include numeric computations and optimization goals.[4]

Although these generalizations may seem rather modest, they demonstrated some interesting things about the nature of classical planning, as discussed below.

For each of the planners in the competition, the planning engine was problem-independent, and the input for each planning problem included the initial state, the goal or objective to be achieved, and the set of operators for the problem domain. However, the planners varied in terms of how much additional knowledge was made available to them about how to solve problems in the planning domain. The planners in the competition can be classified into three categories:

- **Non-tunable planners.** In these planning systems, the problem input consists solely of the information specified above: initial state, goal or objective, and operators. In the competition, the planners in this class included most, but not all, of the ones that Long and Fox [22] have called "fully automated" planners.

[3]In classical planning, a *domain* is basically a set of planning operators. For each domain it is possible to produce an unlimited number of randomly generated problems by specifying initial and goal states.

[4]In the 2004 International Planning Competition, which was in progress at the time that we wrote this paper, some of the restrictions have been weakened further. For details, see ⟨http://www-rcf.usc.edu/~skoenig/icaps/icaps04/planningcompetition.html⟩.

- **Tunable planners.** Although these planning systems have usually been classified as "fully automated," there are ways to tune them for better performance in a given planning domain. In the 2002 competition, the planners in this class included LPG [13] and FF [18].[5] For LPG, one of the inputs was a setting to optimize its performance for speed, quality, or something in between, and LPG was run with all three settings during the competition. For FF, there were two different versions, both of which were entered in the competition.
- **Domain-configurable planners.** These are planning systems whose input includes detailed information about how to solve problems in the relevant problem domain. Such planners have sometimes been called "hand-tailored" planners [22], but that term is not accurate since the planning engine is domain-independent. They have also been described as "hand-tailorable" [26] or "control-intensive" [5] planners. In the competition, the planners of this type included SHOP2 [26], TLPlan [5], and TALplanner [19].

Performance measures for application-specific planners. For application-specific planning systems, usually the performance measures and the ways of testing them are also application-specific. For example, manufacturing-planning systems are tested on collections of manufacturing-planning problems that are specific to the particular domain in which the planning is done (e.g., see [29]); and in computer bridge [31], there are annual competitions in which performance is measured by playing the programs against each other on a set of bridge hands, using the normal rules for a bridge tournament. These kinds of measures are useful for the application domain at hand, but they are not directly generalizable to other domains.

5.2. Observations

From the survey in the previous section, we can make the following observations.

Observation 1: *There is a tradeoff between the amount of work needed to configure a planner for a domain, and planner's speed and coverage of that domain once it has been so configured.* Here are several examples:

- In the planning competitions, the non-tunable planners were the ones that had the highest running time and solved the fewest planning problems—but configuring a non-tunable planner requires no workwhatsoever, provided that the planner is capable of representing the planning domain.
- In the planning competitions, the tunable planners were faster than the fully automated ones. However, some experimentation may be required to find the settings that give the best overall performance.

[5]Some of the other planners in the competition may also be capable of being tuned, but LPG and FF were the only ones for which results were submitted using more than one setting or version.

- In the planning competitions, the domain-configurable planners solved planning problems several orders of magnitude faster than the others, and solved many problems that were too large for the other planners to solve. However, the domain-configurable planners require a significant amount of up-front work to formulate the domain-specific knowledge that enables them to run so quickly, and this work must be redone each time one switches to a new domain.
- In order to get top-level performance in a specific application domain, it may be necessary to develop a domain-specific planner.[6] However, developing and tuning such planners may require years of work. The resulting planning system may be quite good for its particular application domain, but cannot be used to solve problems in any other domain.

Observation 2: *Performance in classical planning domains does not predict performance in other planning-competition domains.* For example:

- Some of the planning systems were designed, sometimes consciously and sometimes tacitly, with classical planning in mind. These planners did well on classical domains, but on non-classical domains they did not perform very well (if they could be used at all).
- On the other hand, some of the planning systems were designed, from the ground up, to work on non-classical planning domains. These systems generally performed well on both the classical and non-classical domains.

Observation 3: *Performance in planning-competition domains does not predict performance in real-world application.* For example:

- Most of the planning systems in the competition, including both good and bad performers, would not be directly usable in real-world applications, because of restrictions on the kinds of planning problems that they can solve.
- A planner that performed poorly in the 2002 planning competition, IxTetT [20], is used quite successfully for the application of robot motion planning [21], a domain which most of the systems in the competition would be unable to address.
- One of the best performers in the 2002 planning competition, SHOP2 [26], is also proving useful in several application areas. It is developing a user base that includes universities, companies such as Sony, Lockheed Martin, and SIFT, and government laboratories such as NIST and NRL.

From the above observations, we conclude that it is not adequate merely to measure running time and percentage of problems solved. Such figures are not meaningful unless one also knows the class of planning problems over which such performance can be achieved, and how much the performance will be degraded on broader classes of planning problems.

[6]Some examples of such systems include Bridge Baron for computer bridge [31], the Intelligent Bending Workstation for sheet-metal bending [16], and RAX for autonomous spacecraft control [25].

6. A Proposed Performance Measure

In this section, we discuss three different aspects of a planning system's performance that we believe are important to measure: the scope of the problems that the planner can solve, the amount and kind of control knowledge that must be given to the planning system, and the size of the problems that the planning system can reasonably solve.

6.1. Problem Scope

We believe that any useful measure of performance for a planning system needs to include the *scope* of the problems that the corresponding planning algorithm is capable of solving. The set of restrictive assumptions in Section 2 can be used as a basis for defining what this scope is. More specifically:

- **Relaxing Assumption A0 (Finite Σ).** An enumerable, possibly infinite set of states may be needed, for example, to describe actions that construct or bring new objects in the world, or to handle numerical state variables. This brings in some theoretical issues about decidability and termination.
- **Relaxing Assumption A1 (Fully Observable Σ).** If we allow a static, deterministic system to be partially observable, then the observations of Σ will not fully disambiguate which state Σ is in. For each observation o, there may be more than one state s such that $\eta(s) = o$. Without knowing which state in $\eta^{-1}(o)$ is the current state, it is no longer possible to predict with certainty whether an action is applicable and what state Σ will be in after each action.
- **Relaxing Assumption A2 (Deterministic Σ).** In a static but nondeterministic system, each action can lead to different possible states, so the planner may have to consider alternatives. Usually nondeterminism requires relaxing Assumption A5 as well. A plan must encode ways for dealing with alternatives, e.g., *conditional* constructs of the form "do a and, depending on its result, do either b or c", and *iterative* constructs, like "do a until a given result is obtained." Notice that the controller has to observe the state s: here we are planning for a *closed-loop control*.
  If the complete knowledge assumption (Assumption A1) is also relaxed, this leads to another difficulty: the controller does not know exactly the current state s of the system at run-time. A limiting case is *null observability*, where no observations at all can be done at run-time. This leads to a particular case of planning for open-loop control called *conformant planning*.
  Some ways of dealing with nondeterminism are extensions of techniques used in classical planning (such as Graph-based or SAT-based planning), while others are designed specifically to deal with nondeterminism, such as planning based

on Markov Decision Processes (MDPs) [7, 14] or model-checking techniques [10, 14].

- **Relaxing Assumption A3 (Static Σ).** We can easily deal with a dynamic system Σ if it is deterministic and fully observable, and if we further assume that for every state s there is at most one contingent event e for which $\gamma(s, e)$ is not empty, and that e will necessarily occur in s. Such a system can be mapped into the restricted model: one redefines the transition for an action a as $\gamma(\gamma(s, a), e)$, where e is the event that occurs in the state $\gamma(s, a)$.
  In the general model of possible events that may or may not occur in a state and "compete" with actions, a dynamic system is nondeterministic from the view point of the planner even if $|\gamma(s, u)| \leq 1$, u being either an action or an event. Deciding to apply action a in s does not focus the planner's prediction to a single state-transition. Here again, a conditional plan will be needed.
- **Relaxing Assumption A4 (Restricted Goals).** Controlling a system may require more complex objectives than reaching a given state. One would like to be able to specify to the planner an *extended goal* with requirements not only on the final state but also on the states traversed, e.g., critical states to be avoided, states that the system should go through, states it should stay in and other constraints on its trajectories. It may also be desirable to have utility functions to be optimized, e.g., to model a system that must function continuously over an indefinite period of time.
- **Relaxing Assumption A5 (Sequential Plans).** Here, a plan may be a mathematical structure that can be richer than a simple sequence of actions. As examples, one may consider a plan to be a partially ordered set, a sequence of sets, a conditional plan that forces alternate routes depending on the outcome and current context of execution, a "universal plan" or a "policy" that maps states to appropriate actions, or a deterministic or nondeterministic automaton that determines what action to execute depending on the previous history of execution. Relaxing Assumption A5 is often required when other assumptions are relaxed, as we have seen in the case of nondeterministic systems (Assumption A3) or when relaxing Assumptions A1, A3, A4 and A6. Plans as partially ordered sets, or as sequences of sets of actions, are more easily handled than conditional plans and policies.
- **Relaxing Assumption A6 (Implicit Time).** In many planning domains, action duration and concurrency have to be taken into account. Time can also be needed for expressing temporally constrained goals and occurrence of events with respect to an absolute time reference. However, time is abstracted away in the state-transition model.[7] This conceptual model considers actions or events as instantaneous transitions: at each clock tick, the controller synchronously reads the observation for the current state (if needed) and applies the planned action.
- **Relaxing Assumption A7 (Offline Planning).** The control problem of driving a system towards some objectives has to be handled online with the dynamics of that system. While a planner may not have to worry about all the details of the actual dynamics, it cannot ignore completely how the system will evolve. At the least, it needs to check, online, whether a solution plan remains valid, and, if needed, to revise it or replan. Other approaches consider planning as a process that modifies the controller online.

For a detailed presentation of techniques for solving planning problems with various combinations of these restrictions, see [14].

6.2. Control Knowledge

Another important aspect of a planning system's performance is what kind of additional control knowledge (other than just the problem definition) will need to be given to the planning system in order for it to address practical problems. This includes, for example, whether the planner needs such knowledge, how precise and specific to a problem the knowledge needs to be, whether the planner needs to be fine-tuned for different planning domains, and how easily this knowledge can be acquired and formalized. It would be quite difficult to express this feature in precise quantified measurements, but a qualitative assessment of this feature can be made, on the basis of a small set of predefined classes ranging from planners that require no control knowledge to those that require the domain author to do some highly demanding algorithm development.

6.3. Problem Size

A third important aspect of performance is what size of problem a planning system can reasonably solve. For this performance aspect, the traditional measures have been numeric ones, along the lines of "this planner can solve problems of size n in time t" for various values of n and t. This has typically been measured by running the planner on a randomly generated set of planning problems.

Such a performance measure has an obvious appeal, but as we concluded in the preceding section, it also has an important limitation: it is highly biased by theset of benchmark problems on which the planner is tested. If a planning system can solve "toy problems" in which the solution plans contain hundreds or even thousands of actions, this does not necessarily say anything about how well—or even whether—the system can solve more useful classes of planning problems.

A more useful way of measuring performance would be to use several classes of problems, ranging in scope from toy problems to very demanding applications, and measure performance in each class.

[7]Other formalisms, such as *timed automata,* extend state-transition systems by incorporating an explicit representation of time.

6. Conclusion

In this paper, we have described existing performance measures for automated planning algorithms, and have discussed the limitations and biases inherent in those performance measures. We have pointed out the importance of developing a performance measure that explicitly the restrictive assumptions on which a planning algorithm depends—and as initial step toward such a performance measure, we have defined and discussed a list of restrictive assumptions that are common to most automated planning systems. We believe that this list provides an initial step toward developing a *taxonomy of restrictions* that can be used to measure the scope of planning algorithms.

Based on the above considerations, we have proposed a composite performance measure based on three factors:

- the scope of the planning algorithm: which set of restrictive assumption are needed and which can be lifted,
- the control knowledge and tuning required for each planning domain,
- the size of the problems that can be solve in a reasonable amount of time in each area of its scope (i.e., for each combination of relaxed assumptions it can handle).

Several aspects of this performance measure are not yet (or not yet fully) developed, and we hope that this paper will encourage researchers to make the effort needed to develop them.

Acknowledgement

This work was supported in part by the CNRS, France, and in the US by AFRL contract F30602-00-2-0505 and NSF grant IIS0412812. The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

[1] D. W. Aha. Plan deconfliction, repair, and authoring in EDSS. Technical report, Naval Research Laboratory, 2002. Progress report.

[2] J. Allen and G. Ferguson. Human-machine collaborative planning. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, 2002.

[3] R. Aylett, J. Soutter, G. Petley, P. W. H. Chung, and A. Rushton. Ai planning in a chemical plant domain. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 1998.

[4] F. Bacchus. The AIPS '00 planning competition. *AI Magazine*, 22(1):47–56, 2001.

[5] F. Bacchus. The power of modeling—a response to PDDL2.1. *Journal of Artificial Intelligence Research*, 20:125–132, 2003.

[6] D. Bernard, E. Gamble, N. Rouquette, B. Smith, Y. Tung, N. Muscettola, G. Dorias, B. Kanefsky, J. Kurien, W. Millar, P. Nayak, and K. Rajan. Remote agent experiment. ds1 technology validation report. Technical report, NASA Ames and JPL report, 1998.

[7] C. Boutilier, T. L. Dean, and S. Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In Ghallab and Milani, editors, *New Directions in AI planning*, pages 157–171. IOS Press, 1996.

[8] L. Castillo, J. Fdez-Olivares, and A. Gonzlez. Automatic generation of control sequences for manufacturing systems based on partial order planning techniques. *Artificial Intelligence in Engineering*, 4(1):15–20, 2000.

[9] T.-C. Chang. *Expert Process Planning for Manufacturing*. Addison-Wesley, Reading, MA, 1990.

[10] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[11] DOT. An assessment of the u.s. marine transportation system, a report to congress. Technical report, U.S. Department of Transportation, 1999. 103 pages.

[12] I. Frank and D. A. Basin. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100(1-2):87–123, 1998.

[13] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in lpg. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

[14] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[15] M. L. Ginsberg. Partition search. In H. Shrobe and T. Senator, editors, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 228–233, Menlo Park, California, 1996. AAAI Press.

[16] S. K. Gupta, D. A. Bourne, K. Kim, , and S. S. Krishanan. Automated process planning for sheet metal bending operations. *Journal of Manufacturing Systems*, 17(5):338–360, 1998.

[17] S. K. Gupta, W. C. Regli, and D. S. Nau. Manufacturing feature instances: Which ones to recognize? In *ACM Solid Modeling Conference*, 1995.

[18] J. Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[19] J. Kvarnstrm and M. Magnusson. TALplanner in IPC-2002: Extensions and control rules. *Journal of Artificial Intelligence Research*, 20:343–377, 2003.

[20] P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1643–1649, 1995.

[21] S. Lemai and F. Ingrand. Interleaving temporal planning and execution in robotics domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2004. To appear.

[22] D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

[23] M. Mantyla, D. S. Nau, and J. Shah. Challenges in feature-based manufacturing research. *CACM*, 39(2):77–85, 1996.

[24] D. McDermott. AIPS-98 planning competition results, 1998.

[25] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

[26] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, December 2003.

[27] N. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1980.

[28] Parametric Technology Corporation. Pro/ENGINEER NC Sheetmetal. ⟨http://www.ptc.com/appserver/it/icm/cda/icm01_list.jsp?group=2%01&num=1&show=y&keyword=342⟩, 2004.

[29] J. Shah, M. Mantyla, and D. S. Nau, editors. *Advances in Feature Based Manufacturing*. Elsevier/North Holland, 1994.

[30] S. J. J. Smith. *Task-Network Planning Using Total-Order Forward Search, and Applications to Bridge and to Microwave Module Manufacture*. PhD thesis, University of Maryland, 1997.

[31] S. J. J. Smith, D. S. Nau, and T. Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–105, 1998.