AI Magazine Volume 28 Number 4 (2007) (© AAAI)

Current Trends in Automated Planning

Dana S. Nau

■ Automated planning technology has become mature enough to be useful in applications that range from game playing to control of space vehicles. In this article, Dana Nau discusses where automated-planning research has been, where it is likely to go, where he thinks it should go, and some major challenges in getting there. The article is an updated version of Nau's invited talk at AAAI-05 in Pittsburgh, Pennsylvania.

In ordinary English, *plans* can be many different kinds of things, such as project plans, pension plans, urban plans, and floor plans. In automated-planning research, the word refers specifically to *plans of action*:

> representations of future behavior ... usually a set of actions, with temporal and other constraints on them, for execution by some agent or agents.[1]

One motivation for automated-planning research is theoretical: planning is an important component of rational behavior—so if one objective of artificial intelligence is to grasp the computational aspects of intelligence, then certainly planning plays a critical role. Another motivation is very practical: plans are needed in many different fields of human endeavor, and in some cases it is desirable to create these plans automatically. In this regard, automated-planning research has recently achieved several notable successes such as the Mars Rovers, software to plan sheet-metal bending operations, and Bridge Baron. The Mars Rovers (see figure 1) were controlled by planning and scheduling software that was developed jointly by NASA's Jet Propulsion Laboratory and Ames Research Center (Estlin et al. 2003). Software to plan sheet-metal bending operations (Gupta et al. 1998) is bundled with Amada Corporation's sheet-metal bending machines such as the one shown in figure 2. Finally, software to plan declarer play in the game of bridge helped Bridge Baron to win the 1997 world championship of computer bridge (Smith, Nau, and Throop 1998).

The purpose of this article is to summarize the current status of automated-planning research, and some important trends and future directions. The next section includes a conceptual model for automated planning, classifies planning systems into several different types, and compares their capabilities and limitations; and the Trends section discusses directions and trends.

Conceptual Model for Planning

A conceptual model is a simple theoretical device for describing the main elements of a problem. It may fail to address several of the practical details but still can be very useful for getting a basic understanding of the problem. In this article, I'll use a conceptual model for planning that includes three primary parts (see figure 4a and 4b, which are discussed in the following sections: a *state-transition system*, which is a formal model of the real-world system for which we want to create plans; a *controller*, which performs actions that change the state of the system; and a *planner*, which produces the plans or policies that drive the controller.

Figure 1. One of the Mars Rovers.

State-Transition Systems

Formally, a *state-transition system* (also called a *discrete-event system*) is a 4-tuple $\Sigma = (S, A, E, \gamma)$, where

$S = \{s_0, s_1, s_2, \ldots\}$ is a set of states;

$A = \{a_1, a_2, \ldots\}$ is a set of *actions*, that is, state transitions whose occurrence is controlled by the plan executor;

$E = \{e_1, e_2, \ldots\}$ is a set of *events*, that is, state transitions whose occurrence is not controlled by the plan executor;

$\gamma : S \times (A \cup E) \rightarrow 2^S$ is a state-transition function.

A state-transition system may be represented by a directed graph whose nodes are the states in S (for example, see figure 5). If $s' \in \gamma(s, e)$, where $e \in A \cup E$ is an action or event, then the graph contains a *state transition* (that is, an arc) from s to s' that is labeled with the action or event e.[2]

If a is an action and $\gamma(s, a)$ is not empty, then action a is *applicable* to state s: if the plan executor executes a in state s, this will take the system to some state in $\gamma(s, a)$.

If e is an event and $\gamma(s, e)$ is not empty, then e may *possibly* occur when the system is in state s. This event corresponds to the internal dynamics of the system, and cannot be chosen or triggered by the plan executor. Its occurrence in state s will bring the system to some state in $\gamma(s, e)$.

Given a state-transition system Σ, the purpose of planning is to find which actions to apply to which states in order to achieve some objective, when starting from some given situation. A *plan* is a structure that gives the appropriate actions. The objective can be specified in

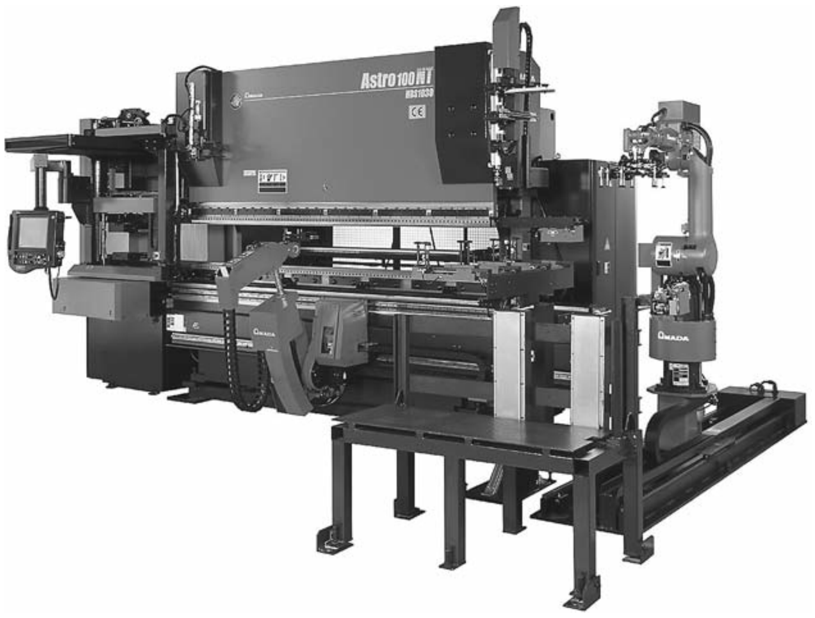

Figure 2. A Sheet-Metal Bending Machine.

(Figure used with permission of Amada Corporation).

several different ways. The simplest specification consists of a *goal state* s_g or a set of goal states S_g. For example, if the objective in figure 5 is to have the container loaded onto the robot cart, then the set of goal states is $S_g = \{s_4, s_5\}$. In this case, the objective is achieved by any sequence of state transitions that ends at one of the goal states. More generally, the objective might be to get the system into certain states, to keep the system away from certain other states, to optimize some utility function, or to perform some collection of tasks.

Planners

The planner's input is a *planning problem*, which includes a description of the system Σ, an initial situation and some objective. For example, in figure 5, a planning problem P might consist of a description of Σ, the initial state s_0, and a single goal state s_5.

The planner's output is a plan or policy that solves the planning problem. A *plan* is a sequence of actions such as

<take, move1, load, move2>.

A *policy* is a partial function from states into actions, such as

$\{(s_0$, take), $(s_1$, move1), $(s_3$, load), $(s_4$, move2)$\}$.[3]

The aforementioned plan and policy both solve the planning problem P. Either of them, if executed starting at the initial state s_0, will take Σ through the sequence of states $\langle s_1, s_2, s_3, s_4, s_5\rangle$.[4]

In general, the planner will produce actions that are described at an abstract level. Hence it may be impossible to perform these actions without first deciding some of the details. In many planning problems, some of these details include what resources to use and what time to do the action.

What Resources to Use. Exactly what is meant by a *resource* depends on how the problem is specified. For example, if Σ contained more than one robot, then one approach would be to require the robot's name as part of the action (for example, move1(robot) and move1(robot2)), and another approach would be to consider the robot to be a resource whose identity will be determined later.

What Time to Do the Action. For example, in

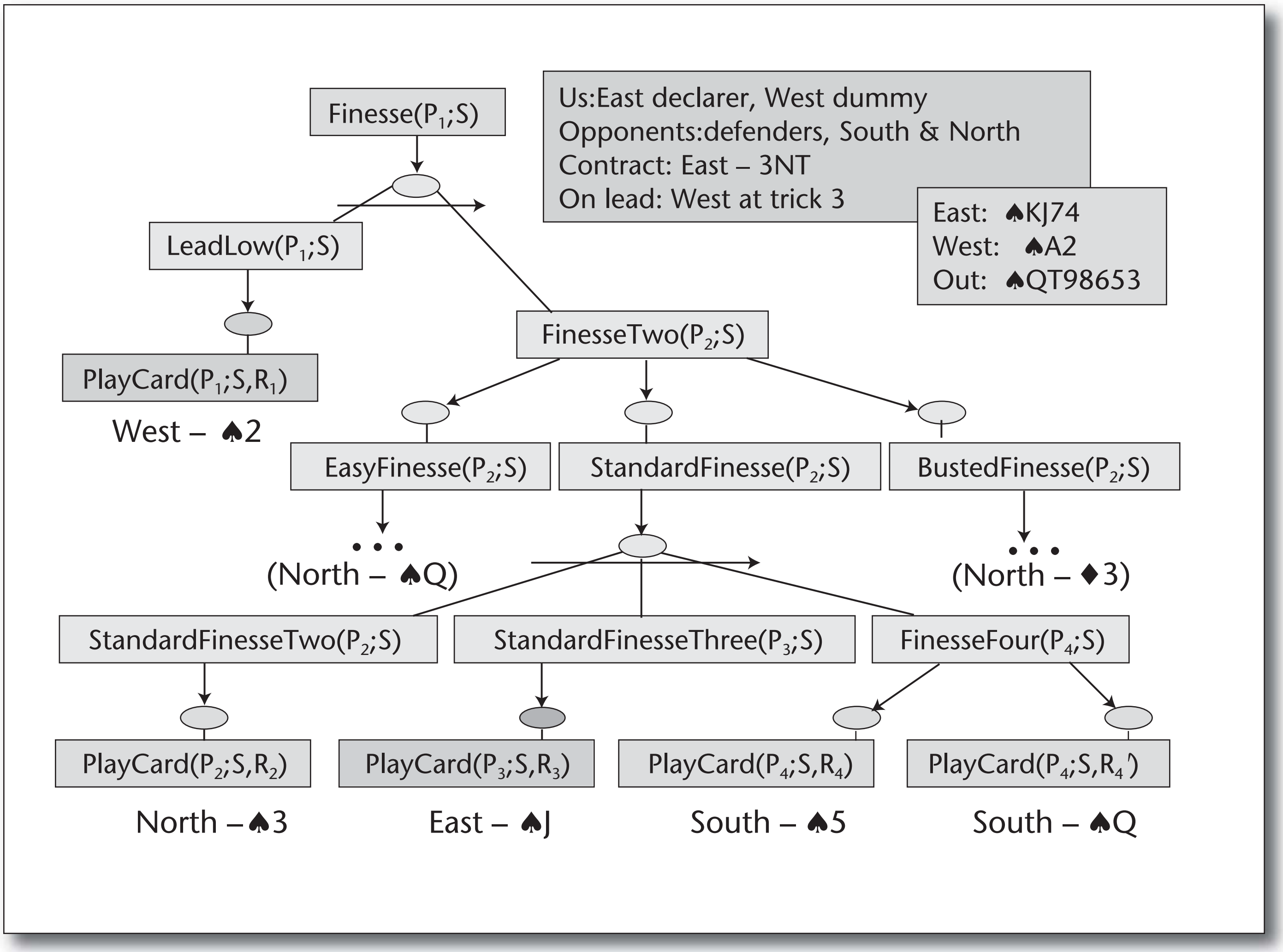

Figure 3. A Hierarchical Plan for a Finesse in Bridge.

order to load the container onto the robot, we might want to start moving the crane before the robot arrives at location1, but we cannot complete the load operation until after the robot has reached location1 and has stopped moving.

In such cases, one approach is to have a separate program called a *scheduler* that sits in between the planner and the controller (see figure 4c), whose purpose is to determine those details. Another approach is to integrate the scheduling function directly into the planner. The latter approach can substantially increase the complexity of the planner, but on complex problems it can be much more efficient than having a separate scheduler.

Controllers

The *controller*'s input consists of plans (or schedules, if the system includes a scheduler) and observations about the current state of the system. The controller's output consists of actions to be performed in the state-transition system.

In figure 4, notice that the controller is *online*. As it performs its actions, it receives *observations*, each observation being a collection of sensor inputs giving information about Σ's current state. The observations can be modeled as an observation function $\eta : S \to O$ that maps S into some discrete set of possible observations. Thus, the input to the controller is the observation $o = \eta(s)$, where s is the current state.

If η is a one-to-one function, then from each observation o we can deduce exactly what state Σ is in. In this case we say that the observations provide *complete* information. For example, in figure 5, if there were a collection of sensors that always provided the exact locations of the

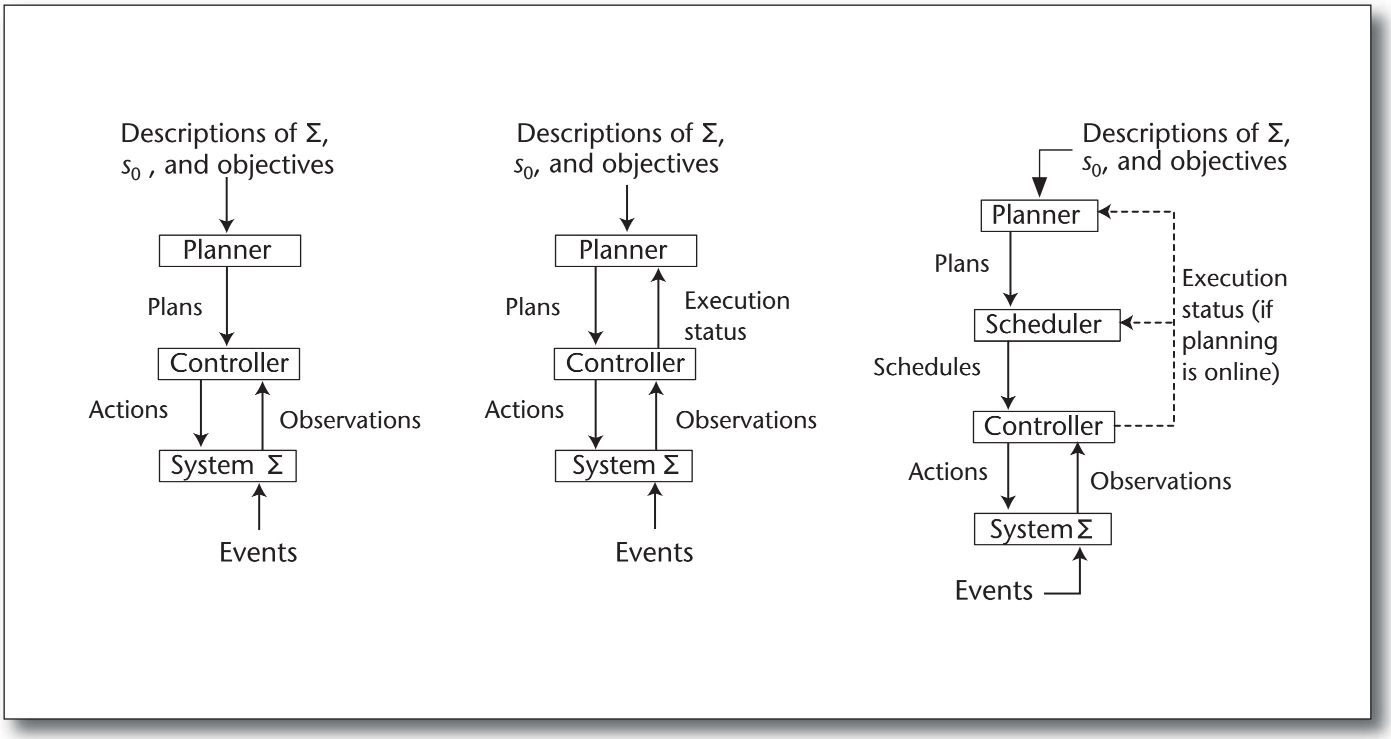

Figure 4. Simple Conceptual Models for (a) Offline Planning, (b) Online Planning, and (c) Planning with a Separate Scheduler.

robot and the container, then this sensor would provide complete information—or, at least, complete information for the level of abstraction used in the figure.

If η is not a one-to-one function, then the best we can deduce from an observation *o* is that Σ is in one of the states in the set $\eta^{-1}(o) \subseteq S$, and in this case we say that the observations provide *incomplete* information about Σ's current state. For example, in figure 5, if we had a sensor that told us the location of the robot but not the location of the container, this sensor would provide incomplete information.

Offline and Online Planning

The planner usually works *offline*, that is, it receives no feedback about Σ's current state. Instead, it relies on a formal description of the system, together with an initial state for the planning problem and the required objective. It is not concerned with the actual state of the system at the time that the planning occurs but instead with what states the system may be in when the plan is executing.

Most of the time, there are differences between Σ and the physical system it represents—for example, the state-transition system in figure 5 does not include the details of the low-level control signals that the controller will need to transmit to the crane and the robot. As a consequence, the controller and the plan must be robust enough to cope with differences between Σ and the real world. If the differences can become too great for the controller to handle from what is expected in the plan, then more complex control mechanisms will be required than what we have described so far. For example, the planner (and scheduler, if there is one) may need feedback about the controller's execution status (see the dashed lines in the figure), so that planning and acting can need be interleaved. Interleaving planning and acting will require the planner and scheduler to incorporate mechanisms for supervision, revision, and regeneration of plans and schedules.

Types of Planners

Automated planning systems can be classified into the following categories, based on whether—and in what way—they can be configured to work in different planning domains: domain-specific planners, domain-independent planners, and domain-configurable planners.

Domain-specific planners are planning systems that are tailor-made for use in a given planning domain and are unlikely to work in other domains unless major modifications are made to the planning system. This class

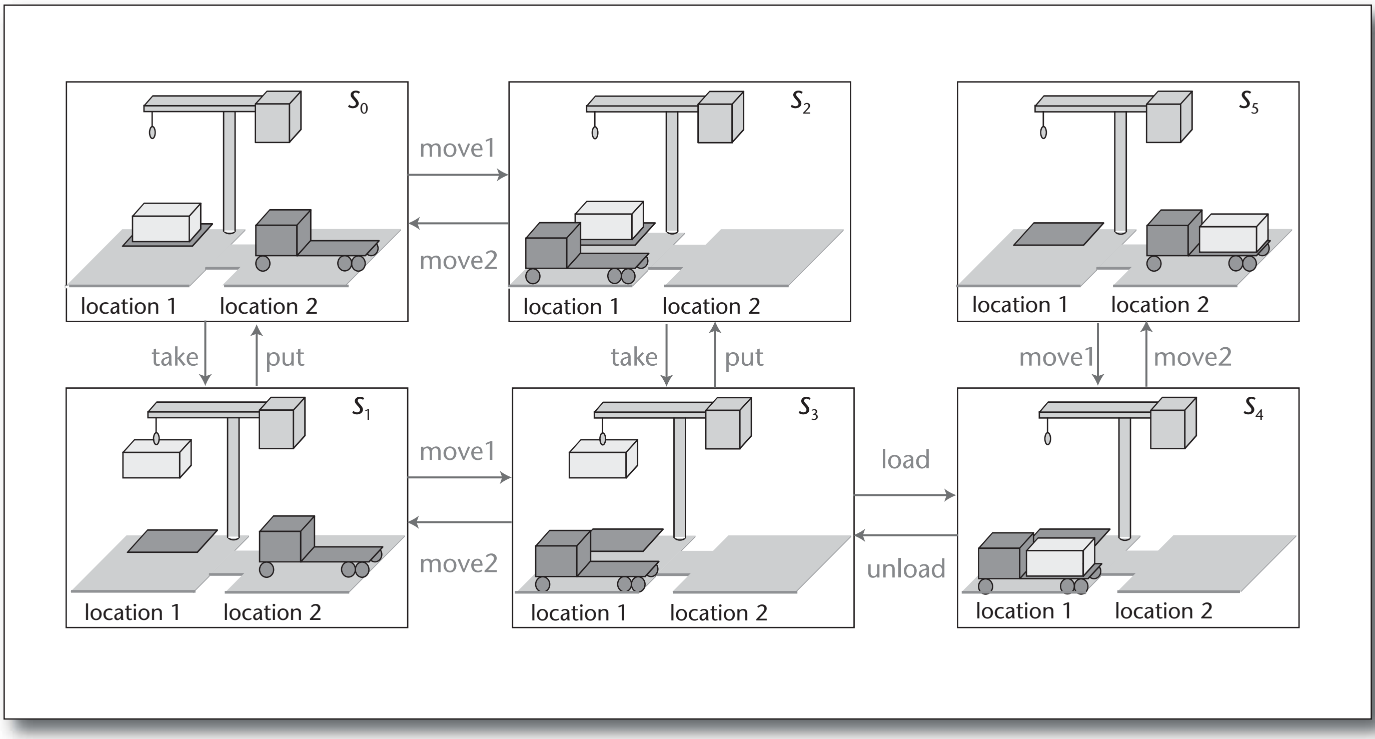

Figure 5. A State-Transition System for a Simple Domain Involving a Crane and a Robot for Transporting Containers.

includes most of the planners that have been deployed in practical applications, such as the ones depicted in figures 1–3.

In *domain-independent planning systems,* the sole input to the planner is a description of a planning problem to solve, and the planning engine is general enough to work in any planning domain that satisfies some set of simplifying assumptions. The primary limitation of this approach is that it isn't feasible to develop a domain-independent planner that works efficiently in every possible planning domain: for example, one probably wouldn't want to use the same planning system to play bridge, bend sheet metal, or control a Mars Rover. Hence, in order to develop efficient planning algorithms, it has been necessary to impose a set of simplifying assumptions that are too restrictive to include most practical planning applications.

Domain-configurable planners are planning systems in which the planning engine is domain-independent but the input to the planner includes domain-specific knowledge to constrain the planner's search so that the planner searches only a small part of the search space. Examples of such planners include *HTN Planners* such as O-Plan (Tate, Drabble, and Kirby 1994), SIPE-2 (Wilkins 1988), and SHOP2 (Nau et al. 2003), in which the domain-specific knowledge is a collection of methods for decomposing tasks into subtasks, and *control-rule planners* such as TLPlan (Bacchus and Kabanza 2000) and TALplanner (Kvarnström and Doherty 2001), in which the domain-specific knowledge is a set of rules for how to remove nodes from the search space.[5]

The next two sections are overviews of domain-independent and domain-configurable planners. This article does not include a similar overview of domain-specific planners, since each of these planners depends on the specific details of the problem domain for which it was constructed.

Domain-Independent Planning

For nearly the entire time that automated planning has existed, it has been dominated by research on domain-independent planning. Because of the immense difficulty of devising a domain-independent planner that would work well in *all* planning domains, most research has focused on *classical planning domains*, that is, domains that satisfy the following set 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*, that is, 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*, that is, 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*, that is, the set of events *E* is empty. Σ has no internal dynamics; it stays in the same state until the controller applies some action.[6]

Assumption A4 (Attainment Goals). The only kind of goal is an *attainment goal*, which is specified as an explicit goal state 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 the state-transition model, which 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.

In summary, classical planning requires 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 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 $s_k \in s_g$.

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 (Ghallab, Nau, and Traverso 2004) 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 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 is executable, producing a sequence of states whose final state is in S_g.[7]

A more expressive representation is the *classical representation*: 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 is 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.

Classical Planning Algorithms

In the following paragraphs, I provide brief summaries of some of the best-known techniques for classical planning: plan-space planning, planning graphs, state-space planning, and translation into other problems.

Plan-Space Planning. In plan-space planning, the basic idea is to plan for a set of goals $\{g_1, \ldots, g_k\}$ by planning for each of the individual goals more-or-less separately, but maintaining various bookkeeping information to detect and resolve interactions among the plans for the individual goals. For example, a simple domain called the Dock Worker Robots domain (Ghallab, Nau, and Traverso 2004), which includes piles of containers, robots that can carry containers to different locations, and cranes that can put containers onto robots or take them off of robots. Suppose there are several piles of containers as shown in state s_0 of figure 6, and the objective is to rearrange them as shown in state s_g. Then a plan-space planner such as UCPOP (Penberthy and Weld, 1992) will produce a partially ordered plan like the one shown in the figure.

Planning Graphs. A *planning graph* is a structure such as the one shown in figure 7. For each *n*, level *n* includes every action *a* such that at level *n* – 1, *a*'s preconditions are satis-

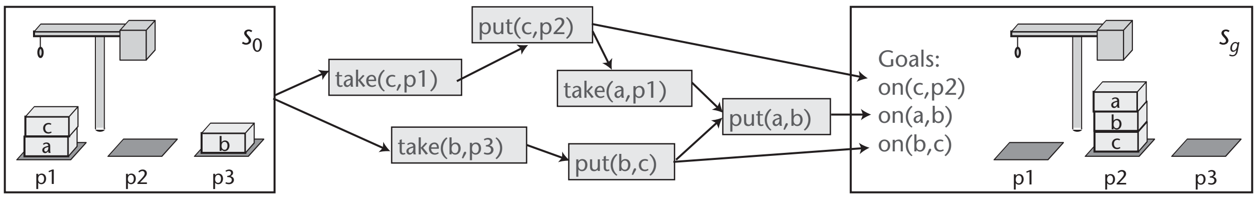

Figure 6. During the Operation of a Plan-Space Planner, Each Node of the Search Space is a Partial Plan like the One Shown Here.

On the left-hand side is the initial state s_0, on the right-hand side is the goal state s_g, and in the middle is a collection of actions and constraints. Planning proceeds by introducing more and more constraints or actions, until the plan contains no more flaws (that is, it is guaranteed to be executable and to produce the goal.

fied and do not violate certain kinds of mutual-exclusion constraints. The literals at level n include the literals at level $n - 1$, plus all effects of all actions at level n. Thus the planning graph represents a relaxed version of the planning problem in which several actions can appear simultaneously even if they conflict with each other. The basic GraphPlan algorithm is as follows:

```
For $n = 1, 2, \ldots$ until a solution is found,
    Create a planning graph of $n$ levels.
    Do a backwards state-space search from the
    goal to try to find a solution plan, but restrict
    the search to include only the actions in the
    planning graph.
```

The planning graph can be computed relatively quickly (that is, in a polynomial amount of time), and the restriction that the backward search must operate within the planning graph dramatically improves the efficiency of the backward search. As a result, GraphPlan runs much faster than plan-space planning algorithms. Researchers have created a large number of planning algorithms based on Graphplan, including IPP, CGP, DGP, LGP, PGP, SGP, TGP, and others.[8]

State-Space Planning. Although state-space search algorithms are very well known, it was not until a few years ago that they began to receive much attention from classical planning researchers, because nobody knew how to come up with a good heuristic function to guide the search. The breakthrough came when it was realized that heuristic values could be computed relatively quickly by extracting them from relaxed solutions (such as the planning graphs discussed earlier). This has led to planning algorithms such as HSP (Bonet and Geffner 1999) and FastForward (Hoffmann and Nebel 2001).

Translation Into Other Problems. Here, the basic idea consists of three steps. First, translate the planning problem into another kind of combinatorial problem—such as satisfiability or integer programming—for which efficient problem solvers already exist. Second, use a satisfiability solver or integer-programming solver to solve the translated problem. Third, take the solution found by the problem solver and translate it into a plan. This approach has led to planners such as Satplan (Kautz and Selman 1992).

Limitations of Classical Planning

For nearly the entire time that automated planning has existed, it has been dominated by research on classical planning. In fact, for many years the term *domain-independent planning system* was used almost synonymously with classical planning, as if there were no limitations on what kind of planning domains could be represented as classical planning domains. But since classical planning requires all of the restrictive assumptions in the Domain-Independent Planning section, it actually is restricted to a very narrow class of planning domains that exclude most problems of practical interest. For example, Mars exploration (figure 1) and sheet-metal bending (figure 2) satisfy none of the assumptions in the

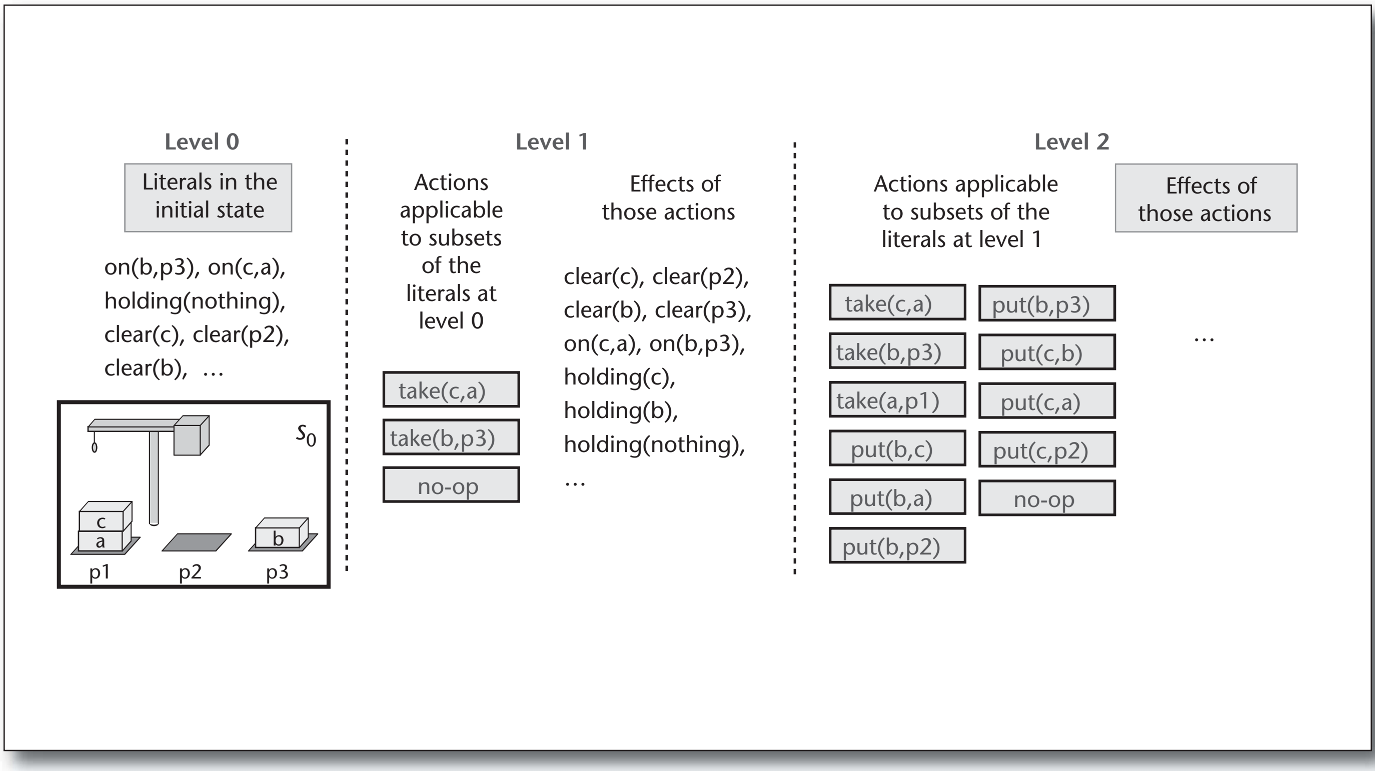
Level 0
Literals in the initial state
on(b,p3), on(c,a), holding(nothing), clear(c), clear(p2), clear(b), …
s_0
c
a
b
p1
p2
p3
Level 1
Actions applicable to subsets of the literals at level 0
take(c,a)
take(b,p3)
no-op
Effects of those actions
clear(c), clear(p2), clear(b), clear(p3), on(c,a), on(b,p3), holding(c), holding(b), holding(nothing), …
Level 2
Actions applicable to subsets of the literals at level 1
take(c,a)
take(b,p3)
take(a,p1)
put(b,c)
put(b,a)
put(b,p2)
put(b,p3)
put(c,b)
put(c,a)
put(c,p2)
no-op
Effects of those actions
…

Figure 7. Part of a Planning Graph.

The figure omits several details (for example, mutual-exclusion relationships). The 0'th level contains all literals that are true in the initial state s_0. For each *i*, level *i* + 1 includes all actions that are applicable to the literals (or some subset of them) at level *i* and all of these actions' effects.

Domain-Independent Planning section, and the game of bridge (figure 3) satisfies only A0, A2, and A6.

On the other hand, several of the concepts developed in classical planning have been quite useful in developing planners for nonclassical domains. This will be discussed further in the Directions for Growth section.

Domain-Configurable Planners

Domain-specific and domain-configurable planners make use of domain-specific knowledge to constrain the search to a small part of the search space. As an example of why this might be useful, consider the task of traveling from the University of Maryland in College Park, Maryland, to the LAAS research center in Toulouse, France. We may have a number of actions for traveling: walking, riding a bicycle, roller skating, skiing, driving, taking a taxi, taking a bus, taking a train, flying, and so forth. We may also have a large number of locations among which one may travel: for example, all of the cities in the world. Before finding a solution, a domain-independent planner might first construct a huge number of nonsensical plans, such as the following one:

> Walk from College Park to Baltimore, then bicycle to Philadelphia, then take a taxi to Pittsburgh, then fly to Chicago, ….

In contrast, anyone who has ever had much practical experience in traveling knows that there are only a few reasonable options to consider. For traveling from one location to another, we would like the planner to concentrate only on those options. To do this, the planner needs some domain-specific information about how to create plans.

In a domain-specific planner, the domain-specific information may be encoded into the planning engine itself. Such a planner can be quite efficient in creating plans for the target domain but will not be usable in any other planning domain. If one wants to create a planner for another domain—for example, planning the movements of a robot cart—one will need to build an entirely new planner.

In a domain-configurable planner, the planning engine is domain independent, but the input to the planner includes a *domain description*, that is, a collection of domain-specific

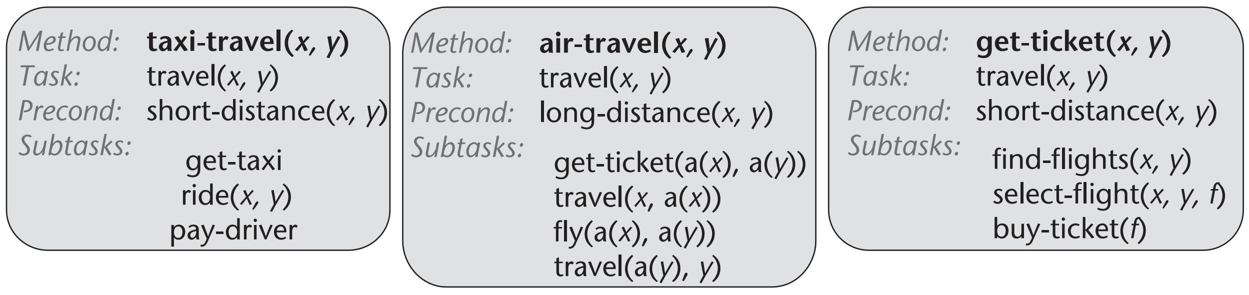

Figure 8. HTN Methods for a Simple Travel-Planning Domain.

In this example, the subtasks are to be done in the order that they appear.

knowledge written in a language that is understandable to the planning engine. Thus, the planning engine can be reconfigured to work in another problem domain by giving it a new domain description.

The existing domain-configurable planners can be divided into two types: hierarchical task network (HTN) planners and control-rule planners. These planners are discussed in the following two sections.

HTN Planners

In a hierarchical task network (HTN) planner, the planner's objective is described not as a set of goal states but instead as a collection of *tasks* to perform.[9] Planning proceeds by decomposing tasks into subtasks, subtasks into subsubtasks, and so forth in a recursive manner until the planner reaches *primitive* tasks that can be performed using actions similar to the actions used in a classical planning system. To guide the decomposition process, the planner uses a collection of *methods* that give ways of decomposing tasks into subtasks.

As an illustration, figure 8 shows two methods for the task of traveling from one location to another: *air travel* and *taxi travel*. Traveling by air involves the subtasks of purchasing a plane ticket, traveling to the local airport, flying to an airport close to our destination, and traveling from there to our destination. Traveling by taxi involves the subtasks of calling a taxi, riding in it to the final destination, and paying the driver. The preconditions specify that the *air-travel* method is applicable only for long distances and the *taxi-travel* method is applicable only for short distances. Now, consider again the task of traveling from the University of Maryland to LAAS. Since this is a long distance, the *taxi-travel* method is not applicable, so we must choose the *air-travel* method. As shown in figure 9, this decomposes the task into the following subtasks: (1) purchase a ticket from IAD (Washington Dulles) airport to TLS (Toulouse Blagnac), (2) travel from the University of Maryland to IAD, (3) fly from IAD to TLS, and (4) travel from TLS to LAAS.

For the subtasks of traveling from the University of Maryland to BWI and traveling from Logan to MIT, we can use the *taxi-travel* method to produce additional subtasks as shown in figure 9.

HTN-planning research has been much more application-oriented than most other AI-planning research. Domain-configurable systems such as O-Plan (Tate, Drabble, and Kirby 1994), SIPE-2 (Wilkins 1988), and SHOP2 (Nau et al. 2003) have been used in a variety of applications, and domain-specific HTN planning systems have been built for several application domains (for example, Smith, Nau, and Throop [1998]).

Control-Rule Planners

In a control-rule planner, the domain-specific information is a set of rules describing conditions under which the current node can be pruned from the search space. In most cases, the planner does a forward search, starting from the initial state, and the control rules are written in some form of temporal logic. Don't read too much into the name *temporal logic*, because the logical formalisms used in these planners provide only a simple representation of time, as a sequence of states of

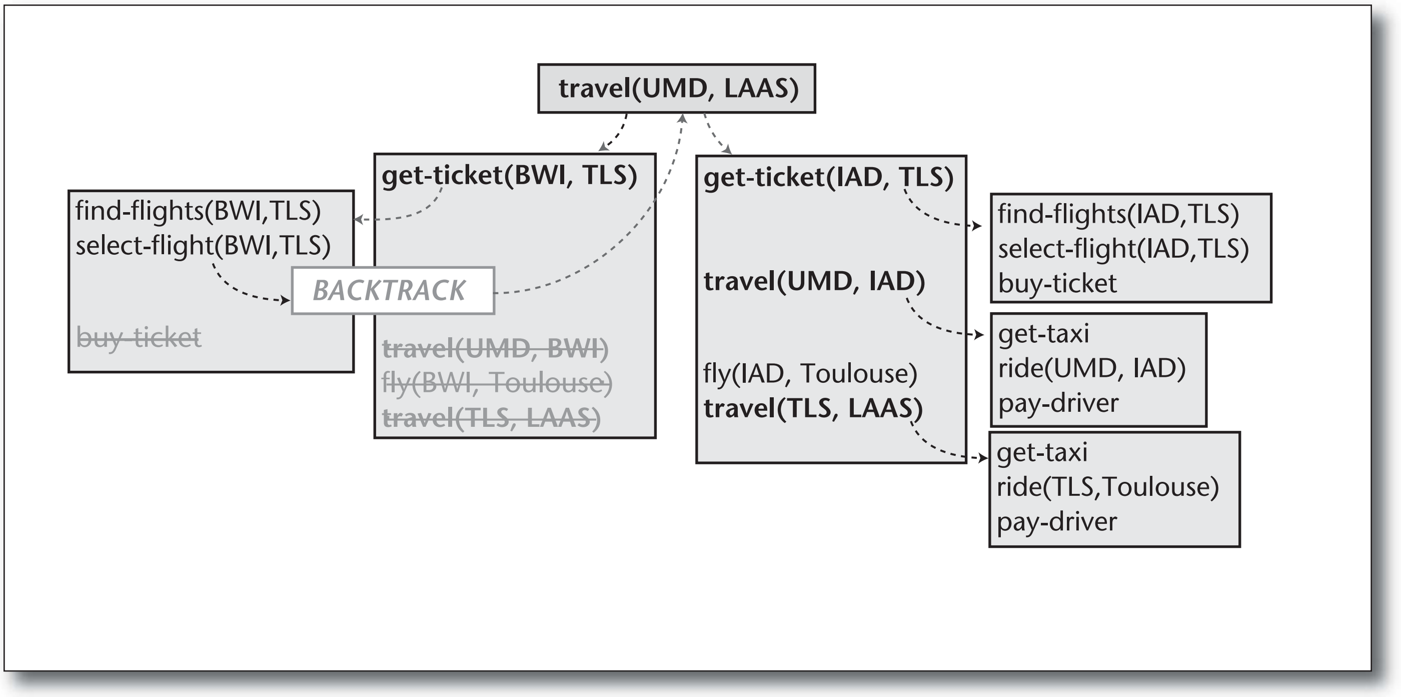

Figure 9. Backtracking Search to Find a Plan for Traveling from UMD to LAAS.

The downward arrows indicate task decompositions; the upward ones indicate backtracking.

the world s_0 = the initial state, $s_1 = \gamma(s_0, a_1)$, $s_2 = \gamma(s_1, a_2)$, … where a_1, a_2, … are the actions of a plan.

As an example, the planning problem of figure 5 is in a simple planning domain called the Dock Worker Robots domain (Ghallab, Nau, and Traverso 2004), which includes piles of containers, robots that can carry containers to different locations, and cranes that can put containers onto robots or take them off of robots. In this domain, suppose that the only kind of goal that we have is to put various containers into various piles. Then the following control rule may be useful:

> Don't ever pick up a container from the top of any pile *p* unless at least one of the containers in *p* needs to be in another pile.

Here is a way to write this rule in the temporal logic used by TLPlan:

$$\Box\ [\mathrm{top}(p, x) \wedge \neg\exists[y{:}\mathrm{in}(y, p)]\ \exists[q{:}\ \mathrm{GOAL}(\mathrm{in}(y, q)]\ (q \neq p) \Rightarrow \bigcirc\ (\neg\exists[k{:}\mathrm{holding}(k, x)])\ ]$$

What this rule says, literally, is the following:

> In every state of the world, if *x* is at the top of *p* and there are no *y* and *q* such that (1) *y* is in *p*, (2) there's a goal saying that *y* should be in *q*, and (3) $q \neq p$, then in the next state, no crane is holding *x*.

If a planner such as TLPlan reaches a state that does not satisfy this rule, it will backtrack and try a different direction in its search space.

Comparisons

The three types of planners — domain-independent, domain-specific, and domain-configurable — compare with each other in the following respects (see table 1): (1) effort to configure the planner for a new domain, (2) performance in a given domain, and (3) coverage across many domains.

Effort to Configure the Planner for a New Domain. Domain-specific planners require the most effort to configure, because one must build an entire new planner for each new domain—an effort that may be quite substantial. Domain-independent planners (provided, of course, that the new domain satisfies the restrictions of classical planning) require the least effort, because one needs to write only definitions of the basic actions in the domain. Domain-configurable planners are somewhere in between: one needs to write a domain description but not an entire planner.

Performance in a Given Domain. A domain-independent planner will typically have the worst level of performance because it will not take advantage of any special properties of the domain. A domain-specific planner, provided that one makes the effort to write a good one, will potentially have the highest level of performance, because one can encode domain-specific problem-solving techniques directly

	Up Front Human Effort	Performance	Coverage
Highest	Domain Specific	Domain Specific	Configurable
	Configurable	Configurable	Domain Independent
Lowest	Domain Independent	Domain Independent	Domain Specific

Table 1. Relative Comparisons of the Three Types of Planners.

into the planner. A sufficiently capable domain-configurable planner should have nearly the same level of performance, because it should be possible to encode the same domain-specific problem-solving techniques into the domain description that one might encode into a domain-specific planner.[10]

Coverage across Many Domains. A domain-specific planner will typically work in only one planning domain, hence will have the least coverage. One might think that domain-independent and domain-configurable planners would have roughly the same coverage—but in practice, domain-configurable planners have greater coverage. This is due partly to efficiency and partly to expressive power.

As an example, let us consider the series of semiannual International Planning Competitions, which have been held in 1998, 2000, 2002, 2004, and 2006. All of the competitions included domain-independent planners; and in addition, the 2000 and 2002 competitions included domain-configurable planners. In the 2000 and 2002 competitions, the domain-configurable planners solved the most problems, solved them the fastest, usually found better solutions, and worked in many nonclassical planning domains that were beyond the scope of the domain-independent planners.

A Cultural Bias

If domain-configurable planners perform better and have more coverage than domain-independent ones, then why were there no domain-configurable planners in the 2004 and 2006 International Planning Competitions? One reason is that it is hard to enter them in the competition, because you must write all of the domain knowledge yourself. This is too much trouble except to make a point. The authors of TLPlan, TALplanner, and SHOP2 felt they had already made their point by demonstrating the high performance of their planners in the 2002 competition, hence they didn't feel motivated to enter their planners again.

So why not revise the International Planning Competitions to include tracks in which the necessary domain descriptions are provided to the contestants? There are two reasons. The first is that unlike classical planning, in which PDDL[11] is the standard language for representing planning domains, there is no standard domain-description representation for domain-configurable planners. The second is that there is a cultural bias against the idea. For example, when Drew McDermott, in an invited talk at the 2005 International Conference on Planning and Scheduling (ICAPS-05), proposed adding an HTN-planning track to the International Planning Competition, several audience members objected that the use of domain-specific knowledge in a planner somehow constitutes "cheating."

Whenever I've discussed this bias with researchers in fields such as operations research, control theory, and engineering, they generally have found it puzzling. A typical reaction has been, "Why would anyone not want to use the knowledge they have about a problem they're trying to solve?"

Historically, there is a very good reason for the bias: it was necessary and useful for the development of automated planning as a research field. The intended focus of the field is planning *as an aspect of intelligent behavior,* and it would have been quite difficult to develop this focus if the field had been tied too closely to any particular application area or set of application areas.

While the bias has been very useful historically, I would argue that it is not so useful any more: the field has matured, and the bias is too restrictive. Application domains in which humans want to do planning and scheduling typically have the following characteristics: a dynamically changing world; multiple agents (both cooperative and adversarial); imperfect and uncertain information about the world; the need to consult external information sources (sensors, databases, human users) to get information about the world; time durations, time constraints, and overlapping actions; durations, time constraints, asynchronous actions; and numeric computations involving resources, probabilities, geometric and spatial relationships. Classical planning excludes all of these characteristics.

Trends

Fortunately, automated-planning research is moving away from the restrictions of classical planning. As an example, consider the evolution of the international planning competitions: The 1998 competition concentrated exclusively on classical planning. The 2000 competition concentrated primarily on classical planning, but one of its tracks (one of the versions of the Miconic-10 elevator domain [Koehler and Schuster 2000]) included some nonclassical elements. The 2002 competition added some elementary notions of time durations, resources. The 2004 competition added inference rules and derived effects, plus a a new track for planning in probabilistic domains. The 2006 competition added soft goals, trajectory constraints, preferences, plan metrics, and constraints expressed in temporal logic.

Another reason for optimism is the successful use of automated planning and scheduling algorithms in high-profile projects such as the Remote Agent (on Deep Space 1) (Muscettola et al. 1998) and the Mars Rovers (Estlin et al. 2003). Successes such as this create excitement about building planners that work in the real world; and applications such as the Mars Rovers provide opportunities for synergy between theory and applications: a better understanding of real-world planning leads to better theories, and better theories lead to better real-world planners.

Finally, automated-planning research has produced some very powerful techniques for reducing the size of the search space, and these techniques can be generalized to work in nonclassical domains. Examples include partial-order planning, HTN planning, and planning in nondeterministic and probabilistic domains.

Partial-Order Planning. The planning algorithms used in the Remote Agent and the Mars Rovers are based on the plan-space planning technique described in the Classical Planning Algorithms subsection. Some of the primary extensions are ways to reason about time, durations, and resources.

HTN Planning. As I discussed earlier, HTN planners have been used in a variety of application domains. Although most HTN planners have been influenced heavily by concepts from classical planning, they incorporate capabilities that go in various ways beyond the restrictions of classical planning.

Planning in Nondeterministic and Probabilistic Domains. In probabilistic planning domains such as Markov Decision Processes (Boutilier, Dean, and Hanks 1996), the actions have multiple possible outcomes, with probabilities for each outcome (see figure 10a). Nondeterministic planning domains (Cimatti et al. 2003) are similar except that no probabilities are attached to the outcomes (see figure 10b).

A series of recent papers have shown how to extend domain-configurable planning techniques to work in nondeterministic (Kuter et al. 2005) and probabilistic (Kuter and Nau 2005) planning domains. In comparison with previous algorithms for such domains, these new algorithms exhibit substantial performance advantages—advantages analogous to the ones for domain-configurable algorithms in classical planning domains (see the Domain-Configurable Planners section).

Directions for Growth

In my view, some of the more important directions for growth in the near future include planning in multiagent environments, reasoning about time, dynamic external information, acquiring domain knowledge, and cross-pollination with other fields. I'll discuss each of these in greater detail in the following subsections.

Planning in Multiagent Environments

Automated planning research has traditionally assumed that the planner is a monolithic program that solves the problem by itself. But in real-world applications, the planner is generally part of a larger system in which there are other agents, either human or automated or both. When these agents interact with the planner, it is important for the planner to recognize what those agents are trying to accomplish, in order to generate an appropriate response. Examples of such situations include mixed-initiative and embedded planning, assisted cognition, customer service hotlines, and computer games.

Reasoning about Time

Classical planning uses a trivial model of time, consisting of a linear sequence of instantaneous states s_0, s_1, s_2, …; and several temporal logics do the same thing. A more comprehensive model of time would include (1) time durations for actions, overlapping actions, and actions whose durations depend on the conditions under which they are executed; (2) resource assignments, and integrated planning/scheduling; (3) continuous change (for

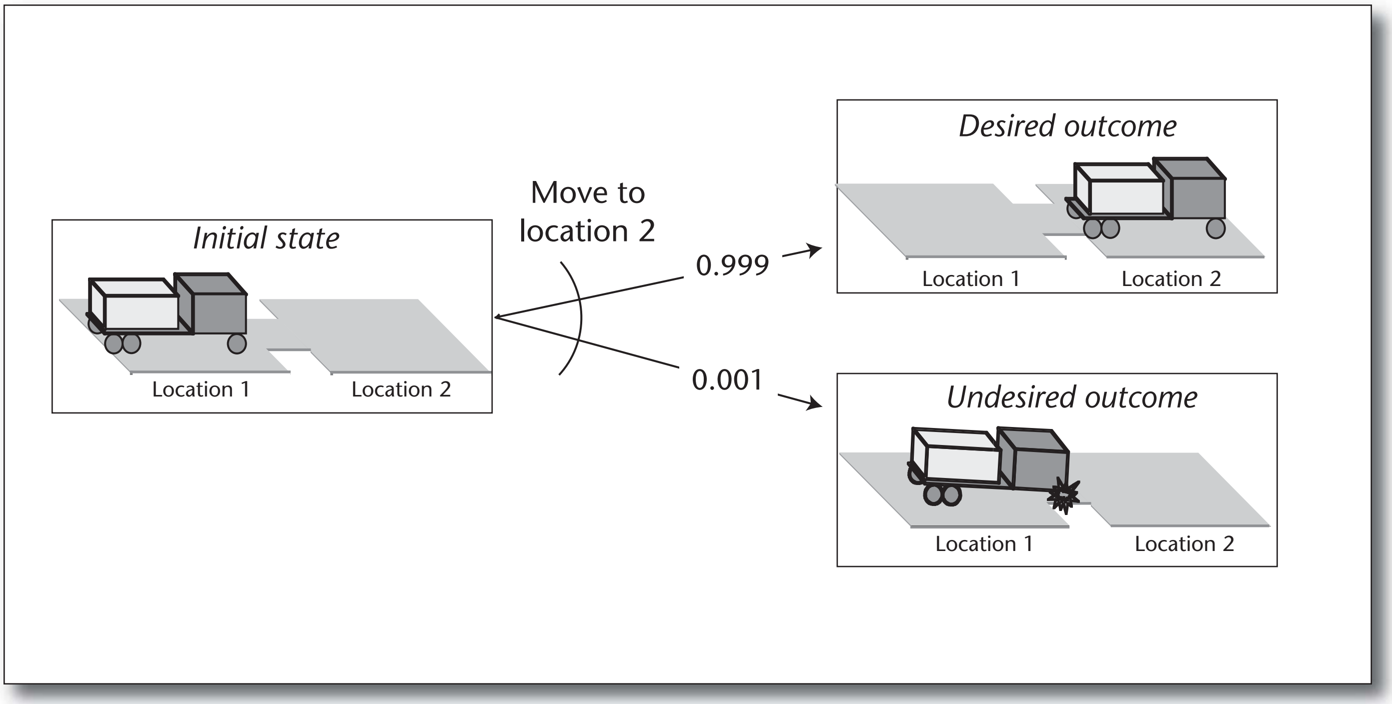

Figure 10. A Simple Example of an Action with a Probabilistic Outcome.

If the robot tries to move to location 2, there is a 99.9 percent chance that it will do so successfully and a 0.1 percent chance it will get a flat tire. An action with a nondeterministic outcome would be similar except that the probabilities would be omitted.

example, vehicle movement); and (4) temporally extended goals (for example, conditions that must be maintained over time).

A healthy amount of growth is already occurring in this direction: for example, various forms of temporal planning were included in the last three International Planning Competitions. But several things remain to be done: for example, the PDDL language used in the competitions does not yet allow actions whose duration depends on the conditions under which they are executed.

Dynamic Environments. In most automated-planning research, the information available is assumed to be static, and the planner starts with all of the information it needs. In real-world planning, planners may need to acquire information from an information source such as a web service, during planning and execution. This raises questions such as What information to look for? Where to get it? How to deal with lag time and information volatility? What if the query for information causes changes in the world? If the planner does not have enough information to infer all of the possible outcomes of the planned actions, or if the plans must be generated in real time, then it may not be feasible to generate the entire plan in advance. Instead, it may be necessary to interleave planning and plan execution.

Some of these questions, I believe, are sufficiently formalizable that a new track could be developed for them in the International Planning Competitions.

Acquiring Domain Knowledge. At many points in this article, I have emphasized the benefits of using domain knowledge during planning, but this leaves the problem of how to acquire this domain knowledge—whether through machine learning, human input, or some combination of the two. This is one of the least appreciated problems for automated-planning research; but in my view it is one of the most important: if we had good ways of acquiring domain knowledge, we could make planners hundreds of times more useful for real-world problems.

Researchers are starting to realize the importance of this problem. For example, at ICAPS-05 there was an informal "Knowledge Engineering Competition" that was more of an exposition than a competition: the participants exhibited GUIs for creating knowledge bases and ways for planners to learn domain knowledge automatically.

A promising source of planning knowledge is the immense collection of data that is now available on the web, which is likely to include information about plans in many different contexts. Can we datamine these plans from the web?

Cross-Pollination with Other Fields. Various kinds of planning are studied in many different fields, including computer games, game theory, operations research, economics, psychology, sociology, political science, industrial engineering, systems science, and control theory. Some of the approaches and techniques developed in these fields have much in common. But it is difficult to tell what the relationships are because the research groups are often nearly disjoint, with different terminology, assumptions, and ideas of what's important. Provided that the appropriate bridges can be made among these fields, there is tremendous potential for cross-pollination. Markov decision processes and computational cultural dynamics are two examples.

Markov decision processes (MDPs) are used in operations research, control theory, and several different subfields of AI including automated planning and reinforcement learning. The MDP models used in automated-planning research typically assume the rewards and probabilities are known, but in reinforcement learning (and often in OR and control theory) they are unknown. MDPs in automated-planning research generally have finitely many states with no good continuous approximations, hence use discrete optimization—but the MDP models used in OR and control theory include features such as infinitely many states, continuous sets of states, actions and costs and rewards that are differentiable functions, hence use linear and nonlinear optimization techniques. Many important problems are hybrids of these differing MDP models, and it would be interesting to combine and extend the techniques from the various fields.

We have instituted a new laboratory at the University of Maryland called the Laboratory for Computational Cultural Dynamics. It brings together faculty in computer science, political science, psychology, criminology, systems engineering, linguistics, and business. The objective is to develop the theory and algorithms needed for tools to support decision making in cultural contexts, to help understand how/why decision makers in various cultures make decisions. The potential benefits of such research include more effective cross-cultural interactions, better governance when different cultures are involved, recovery from conflicts and disasters, and improving quality of life in developing countries.

Conclusion

Automated-planning research has made great strides in recent years. Historically, the field was been limited by its focus on classical planning—but the scope of the field is broadening to include a variety of issues that are important for planning in the real world. As a consequence, automated-planning techniques are finding increased use in practical settings ranging from space exploration to automated manufacturing. Some of the most important areas for future growth of the field include reasoning about other agents, temporal planning, planning in dynamic environments, acquiring domain knowledge, and the potential for cross-pollination with other fields.

Notes

1. Austin Tate (*MIT Encyclopedia of the Cognitive Sciences,* 1999).

2. Here we are using a *Markov game* model of a state-transition function, which assumes that actions and events cannot occur at the same time. To include cases where actions and events could occur simultaneously, we would need $\gamma : S \times A \times E \rightarrow 2^S$.

3. Policies have traditionally been defined to be total functions rather than partial functions. But it is not always necessary to know what to do in *every* state of *S,* because the policy may prevent the system from ever reaching some of the states.

4. Whether to use a plan, a policy, or a more general structure, such as a conditional plan or an execution structure, depends on what kind of planning problem we are trying to solve. In the above example, a plan and a policy work equally well—but more generally, there are some policies that cannot be expressed as plans (for example, in environments where some of the actions have nondeterministic outcomes) and some plans that cannot be expressed as policies (for example, if we come to the same state twice and want to do something different the second time).

5. Note that for an HTN planner or control-rule planner to be domain-configurable, the planning engine must be domain-independent. For example, the version of Bridge Baron mentioned earlier was not domain-configurable because its HTN planning engine was specifically tailored for the game of bridge.

6. Technically, 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.

7. This has also been called STRIPS-style representation, after an early planning system that used a similar representation scheme.

8. Anyone who wants to write a successor of GraphPlan may want to do so soon, before the supply of three-letter acronyms runs out!

9. In some HTN planners, goals can be specified as

AAAI-08 Author Deadlines

December 1, 2007–	
January 25, 2008:	Authors register on the AAAI web site
January 25, 2008:	Electronic abstracts due
January 30, 2008:	Electronic papers due
March 18-20, 2008:	Author feedback about initial reviews
March 31, 2008:	Notification of acceptance or rejection
April 15, 2008:	Camera-ready copy due at AAAI office

tasks such as "achieve(*g*)" where *g* is the goal. But as shown in figure 8, tasks can also represent activities that do not correspond to goals in the classical sense.

10. By analogy, in writing a computer system, one can get the highest level of performance by writing assembly code—but one can get nearly the same level of performance with much less effort by writing in a high-level language.

11. See Drew McDermott's web page, cs-www.cs.yale.edu/homes/dvm.

References

Bacchus, F., and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence* 116(1–2): 123–191.

Bonet, B., and Geffner, H. 1999. Planning as Heuristic Search: New Results. In *Proceedings of the European Conference on Planning (ECP)*. Berlin: Springer-Verlag.

Boutilier, C.; Dean, T. L.; and Hanks, S. 1996. Planning under Uncertainty: Structural Assumptions and Computational Leverage. In *New Directions in AI Planning*, ed. M. Ghallab and A. Milani, 157–171. Amsterdam: IOS Press.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning Via Symbolic Model Checking. *Artificial Intelligence*, 147(1–2): 35–84.

Estlin, T.; Castaño, R.; Anderson, B.; Gaines, D.; Fisher, F.; and Judd, M. 2003. Learning and Planning for Mars Rover Science. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*. San Francisco: Morgan Kaufmann Publishers.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. San Francisco: Morgan Kaufmann Publishers.

Gupta, S. K.; Bourne, D. A; Kim, K.; and Krishanan, S. S. 1998. Automated Process Planning for Sheet Metal Bending Operations. *Journal of Manufacturing Systems* 17(5): 338–360.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI)*, 359–363. Chichester, UK: John Wiley and Sons.

Koehler, J., and Schuster, K. 2000. Elevator Control as a Planning Problem. In *Proceedings of the Fifth International Conference on AI Planning Systems (AIPS)*, 331–338. Menlo Park, CA: AAAI Press.

Kuter, U., and Nau, D. 2005. Using Domain-Configurable Search Control for Probabilistic Planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*. Menlo Park, CA: AAAI Press.

Kuter, U.; Nau, D.; Pistore, M.; and Traverso, P. 2005. A Hierarchical Task-Network Planner Based on Symbolic Model Checking. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 300–309. Menlo Park, CA: AAAI Press.

Kvarnström, J., and Doherty, P. 2001. TALplanner: A Temporal Logic Based Forward Chaining Planner. *Annals of Mathematics and Artificial Intelligence*, 30(1–4): 119–169.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote Agent: To Boldly Go Where No AI System has Gone Before. *Artificial Intelligence*, 103(1–2): 5–47.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20(December): 379–404.

Penberthy, J. S., and Weld, D. 1992. UCPOP: A Sound, Complete, Partial-Order Planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning (KR-92)*. San Francisco: Morgan Kaufmann Publishers.

Smith, S. J. J.; Nau, D. S.; and Throop, T. 1998. Computer Bridge: A Big Win for AI planning. *AI Magazine*, 19(2): 93–105.

Tate, A.; Drabble, B.; and Kirby, R. 1994. *O-Plan2: An Architecture for Command, Planning, and Control*. San Mateo, CA: Morgan-Kaufmann Publishers.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, CA: Morgan Kaufmann Publishers.

Dana Nau is an AAAI Fellow, a professor of both computer science and systems research at the University of Maryland, and the director of the university's Laboratory for Computational Cultural Dynamics. He received his Ph.D. from Duke University in 1979, where he was an NSF graduate fellow. He coauthored the automated-planning algorithms that enabled Bridge Baron to win the 1997 world computer bridge championship. His SHOP2 planning system has been used in hundreds of projects worldwide and won an award in the 2002 International Planning Competition. He has more than 300 publications and is coauthor of *Automated Planning: Theory and Practice*, the first comprehensive textbook on automated planning.