# Perspectives on Artificial Intelligence Planning

**Héctor Geffner**
Departamento de Tecnología
Universitat Pompeu Fabra – ICREA
08003 Barcelona, Spain
hector.geffner@tecn.upf.es

### Abstract

Planning is a key area in Artificial Intelligence. In its general form, planning is concerned with the automatic synthesis of action strategies (plans) from a description of actions, sensors, and goals. Planning thus contrasts with two other approaches to intelligent behavior: the programming approach, where action strategies are defined by hand, and the learning approach, where action strategies are inferred from experience. Different assumptions about the nature of actions, sensors, and costs lead to various forms of planning: planning with complete information and deterministic actions (classical planning), planning with non-deterministic actions and sensing, planning with temporal and concurrent actions, etc. Most work so far has been devoted to classical planning, where significant changes have taken place in the last few years. On the methodological side, the area has become more empirical, on the technical side, approaches based on heuristic or constrained-based search have become common.

In this paper, I try to provide a coherent picture of Planning in AI, making emphasis on the mathematical models that underlie various forms of planning and the ideas that have been found most useful computationally.

## Introduction

The development of *general problem solvers* has been one of the main goals in Artificial Intelligence. A general problem solver is a program that accepts high-level descriptions of problems and automatically computes their solution (Newell & Simon 1963). The motivations for such solvers are two. On the cognitive side, humans are general problem solvers, and thus understanding, reconstructing, or simply emulating such behavior poses a key challenge. On the technical side, modeling problems at a high level is most often simpler than procedurally encoding their solutions, thus an effective GPS tool can be very useful in practice.

A general problem solver must provide the user with a suitable *general modeling language* for describing problems, and *general algorithms* for solving them. While the solutions obtained may not be as good or as fast as those obtained by more specialized methods, the approach will still pay off if the performance of the two methods is not too far apart, or if implementing the specialized solution is just too cumbersome.

In order to develop a general problem solver, its *scope* must be clearly defined. Otherwise, it is not possible to device neither the language nor the algorithms. Indeed, different problems have solutions whose *forms* are often different. For example, the solution of a problem like the Rubik's cube is a *sequence of actions,* while the solution of a diagnostic problem may be an *action strategy* determining the tests to be performed as a function of the observations gathered.

The scope of a general problem solver can be defined by means of a suitable class of mathematical models. The mathematical model that underlies the work in *classical planning* for example, is the familiar *state model* where actions deterministically map one state into another and the task is to find an action sequence that maps the initial state into a goal state. In *conformant planning* (Smith & Weld 1998), on the other hand, an action maps a state into a set of possible successor states, and the task is to find a sequence of actions that lead to the goal for any possible transition and initial state. As *uncertainty* in the state transitions or the initial state is introduced, *feedback* also becomes important, affecting drastically the form of plans. In the absence of feedback, as in classical or conformant planning, a plan is a fixed *sequence of actions*, yet in the presence of full-state feedback, the actions depend on the state observed, and thus plans become *functions mapping states into actions.*

AI Planning is general problem solving over a class of models. Models define the scope of a planner, the types of problems it is supposed to handle, the form of the solutions, and the solutions that are best or optimal. Planning, from this perspective, is about the convenient representation and effective solution of a certain class of mathematical models.

In this paper, I try to provide a coherent picture of AI Planning as a combination of three elements:

1. **representation languages** for describing problems conveniently,
2. **mathematical models** for making the different planning tasks precise, and
3. **algorithms** for solving these models effectively (often making use of information available in their representation)

I also make emphasis on the ideas that have been found most useful computationally, in particular in the area of classical planning, where most of the work has been made. In recent years, this area has undergone significant changes in both methodology and techniques. On the methodological side, the area has become more experimental, on the technical side, approaches based on heuristic or constrained-based search have become common. These changes have been brought about by some key publications (in particular (Blum & Furst 1995)), and events like the Planning Competition (McDermott 2000; Bacchus 2001).

The paper is not a survey on planning but a personal appraisal of part of the field. In particular, I won't say much about knowledge-based planning (Wilkins & des Jardins 2001), where the statement of the planning problem is extended with handcrafted domain-dependent knowledge relevant for solving it. While this is probably the preferred approach in practice and a lot of work in AI has been devoted to it, knowledge-based and domain-independent planning are complementary approaches, and progress in domain-independent planning will eventually translate into more robust knowledge-based planners, that rely less critically on user-provided control information (see (Long & Fox 2000) for a different integration).

It should be mentioned that AI Planning is not the only area concerned with the development of general problem solvers. E.g., work in Linear and Integer Programming (Wolsey 1998) and Constraint Programming (Marriot & Stuckey 1999) is also concerned with the development of problem solvers but over a *different class of models:* linear integer programs, in the first case; constraint satisfaction problems in the second. More interestingly, in both of these areas there has also been a trend to distinguish the *models* from the *languages* used to represent them. For example, AMPL is a recent high-level language for describing linear and integer programs (Fourer, Gay, & Kernighan 1993), while OPL is a recent language for describing such programs and CSPs (Van Hentenryck 1999).

The rest of the paper is organized as follows. We discuss planning models, planning languages, and some key computational aspects in planning in that order, and end with a brief discussion. The presentation aims to be coherent but is definitively not exhaustive.

## Models

We consider first the models that make the semantics of various common planning tasks precise.

### Classical Planning

*Classical planning* can be understood in terms of deterministic *state models* characterized by the following elements (Newell & Simon 1972; Nilsson 1980):

S1. A finite and discrete state space $S$,

S2. an initial situation given by a state $s_0 \in S$,

S3. a goal situation given by a non empty set $S_G \subseteq S$,

S4. actions $A(s) \subseteq A$ applicable in each state $s \in S$,

S5. a deterministic state transition function $f(a, s)$ for $a \in A(s)$

S6. positive action costs $c(a, s)$ for doing action $a$ in $s$.

A *solution* to state models of this type is a sequence of actions $a_0$, $a_1$, …, $a_n$ that generates a state trajectory $s_0$, $s_1 = f(s_0)$, …, $s_{n+1} = f(s_i, a_i)$ such that each action $a_i$ is applicable in $s_i$ and $s_{n+1}$ is a goal state, i.e., $a_i \in A(s_i)$ and $s_{n+1} \in S_G$. The solution is *optimal* when the total cost $\sum_{i=0}^{n} c(s_i, a_i)$ is minimal. In classical planning, it is also assumed that all costs $c(a, s)$ are equal and thus that the optimal plans are the ones with minimal *length.*

Classical planning can be formulated as a deterministic state model that can be solved by *searching* the state-space S1–S6. This is the approach taken by heuristic search planners such as HSP (Bonet & Geffner 2001) and FF (Hoffmann & Nebel 2001). Classical planning, however, can be formulated and solved in a number of other ways; e.g., as a SAT problem (Kautz & Selman 1996), as a Constraint Satisfaction Problem (Do & Kambhampati 2000), as an Integer Programming problem (Vossen *et al.* 1999), etc. These other formulations, however, appear to yield a better pay off when other forms of planning are considered such as parallel planning or planning with resources. The success of heuristic search planners in the classical setting, as seen for example in the last AIPS Planning Competition (Bacchus 2001), lies in the use of good *heuristic functions* which are automatically extracted from the problem representation.

### Planning with Uncertainty

Classical planning assumes that the initial state of the system is known and that state transitions are deterministic. When these assumptions are removed, the state of the world is no longer known. In order to define the planning task in the resulting setting it is necessary to to define how *uncertainty* is modeled, and how *sensing* or *feedback* (that reduce uncertainty) are taken into account.

Uncertainty in planning can be modeled in a number of ways. The two most common approaches are *pure non-determinism* and *probabilities.* In the first case, uncertainty about the state of the world is represented by the *set of states* $S' \subseteq S$ that are deemed possible, in the second, by a *probability distribution* over $S$. In both cases, we refer to the state of uncertainty as a *belief state* and distinguish it from the *actual* but potentially *unknown* state of the world.

In a similar way, uncertainty in state transitions is modeled by a function $F$ that maps an action $a$ and a state $s$ into a non-empty set of states $F(a, s) \subseteq S$ in the non-deterministic setting, and by a distribution $P_a(s'|s)$ for all $s' \in S$ in the probabilistic setting. In either case, an action $a$ *deterministically* maps a belief state $b$ into a new belief state $b_a$. The formula for the non-deterministic case is

$$b_a = \{s \mid s \in F(a, s') \text{ and } s' \in b\} \quad (1)$$

while the analogous formula for the probabilistic case is

$$b_a(s) = \sum_{s' \in S} P_a(s|s') \cdot b(s') \quad (2)$$

What is important is that in both cases the problem of *planning under uncertainty without feedback* reduces to a *deterministic search problem* in *belief space,* a space which can be characterized as the space S1–S6 above by the following elements:

C1. A space $B$ of *belief states* over $S$,

C2. an initial situation given by a belief state $b_0 \in B$,

C3. a goal situation given by a set of target beliefs $B_G$

C4. actions $A(b) \subseteq A$ applicable in each belief state $b$

C5. deterministic transitions $b$ to $b_a$ for $a \in A(b)$ given by (1) and (2) above, and

C6. positive action costs $c(a, b)$.

In this space, it is common to define the target beliefs $B_G$ as the ones that make a given goal condition $G$ certain, the set of applicable actions $A(b)$ in $b$, as the actions $a$ whose preconditions are certain in $b$, and the costs $c(a, b)$ to be uniform. In the non-deterministic case, this results in the model that underlies *conformant planning* (Smith & Weld 1998), namely planning that leads to the goal for any possible initial state or transition. In general, some of these choices can be relaxed without affecting the nature of the problem; in particular, the target belief states $B_G$ and the action preconditions $A(b)$ can be extended to include epistemic conditions like the truth value of an atom $p$ being known (see below for the relation between propositions and states). In the probabilistic setting, additional options are available, e.g., the target belief states $b$ may be defined as the ones that set the probability of the goal above a threshold ($B_G = \{b \mid \sum_{s \in S_G} b(s) > Th\}$) as in the Buridan planner (Kushmerick, Hanks, & Weld 1995), action costs $c(a, b)$ can be defined as expected costs ($c(a, b) = \sum_{s \in S} c(a, s)b(s)$), etc.

## Planning with Sensing

Classical and conformant planning are forms of *open-loop* planning. The resulting plans prescribe the sequences of actions that need to be executed assuming that no additional information is available at execution time. With the ability to sense the world, the choice of the actions depends on the observation gathered and thus the *form* of the plans changes. Of course, sensing only makes sense in a state of uncertainty; if there is no uncertainty, sensing provides no useful information and can be ignored.[1]

In the presence of sensing, the choice of the action $a_i$ at time $i$ depends on all observations $o_0, o_1, \ldots, o_{i-1}$ gathered up to that point. In the case known as *full-state observability*, observations are assumed to reveal the true state of the world, and through standard markovian assumptions, the last observation (state) summarizes the information of all previous states and observations. The choice of the next action in that case becomes a *function* mapping states into actions. Such models are known as Markovian Decision Problems or MDPs (Bertsekas 1995; Dean *et al.* 1993; Boutilier, Dean, & Hanks 1995; Russell & Norvig 1994).

[1]Sensing, however, is useful in the classical setting too if the model used is not guaranteed to be completely accurate (model uncertainty). Indeed, it is standard practice in Control Engineering to obtain a closed-loop control assuming a deterministic model of a non-deterministic system, treating model inaccuracies as noise that gets corrected through the control loop.

In the general case, observations reveal *partial* information about the true state of the world and it is necessary to model how the two are related. The solution then takes the form of functions mapping *belief states* into actions, as states are no longer known and belief states summarize all the information from previous belief states and partial observations (Astrom 1965; Bertsekas 1995). In the probabilistic case, such models are known as Partially Observable Markovian Decision Problems or POMDPs (Sondik 1971; Kaebling, Littman, & Cassandra 1998).

In the presence of feedback, the effects of actions on belief states is no longer deterministic. Let $[o]$ stand for the set of states compatible with an observation $o$ and, for the non-deterministic case, let $O(a, s)$ stand for the (noisy) sensor model, i.e. for the set of observations that are possible when an action $a$ is done and the (hidden) state $s$ is reached. Then the belief states $b_a^o$ that can follow an action $a$ in a belief state $b$ are:

$$b_a^o = \{s \mid s \in b_a \text{ and } s \in [o]\} \tag{3}$$

with $b_a$ defined as in (1) and $o \in O(a, s)$ for $s \in b_a$. The analogous equation in the probabilistic case is

$$b_a^o(s) = P_a(o|s)b_a(s)/b_a(o) \tag{4}$$

with $b_a$ defined as in (2), $b_a(o) \neq 0$ (this is the probability of obtaining observation $o$ after doing action $a$ in $b$), and $P_a(o|s)$ representing the probabilistic sensor model (the probability of obtaining $o$ after doing action $a$ and ending up in $s$).

In either case, the problem of planning with uncertainty and *partial feedback,* becomes a search problem in a non-deterministic *belief space*, while the problem of planning with uncertainty and *full state feedback* is a search problem in a non-deterministic *state space.* In both cases the solutions are not action sequences but *policies:* policies mapping belief states into actions, in the first case, and policies mapping world states into actions in the second. Furthermore, the optimal policies are the ones that minimize the worst possible cost to the goal in the purely non-deterministic formulation, and the expected cost to the goal in the probabilistic formulation. The worst possible cost however is often infinite and thus other notions of success in purely non-deterministic planning have been proposed such as the notion of *strong cyclic plans* (Daniele, Traverso, & Vardi 1999).

State and belief-state policies are the mathematical objects that constitute the solutions of problems involving full and partial feedback respectively. These policies, however, can be represented in a number of ways such as conditional plans or situation-action rules. Moreover, in large problems, it is often not necessary to define such policies over all (belief) states, but only over the initial (belief) state and the (belief) states that are reachable through the policy.

## Temporal Planning

Temporal models extend classical planning in a different direction. We consider here a simple but general model where actions have durations and their execution can overlap in time. More precisely, we assume a duration $d(a) > 0$ for each action $a$, and a predicate $comp(A)$ that defines when a set of actions $A$ can be executed concurrently. For example, in the presence of unary resources, $comp(A)$ will be false if $A$ contains a pair of actions using the same resource (Beck & Fox 1998). Similarly, $comp(A)$ can be defined to be false when $A$ includes pairs of actions having interacting effects or preconditions (Blum & Furst 1995), etc.

A state model for *temporal planning* can be obtained from model S1–S6 by two simple transformations. First we need to replace the single actions $a$ in that model by *sets* of legal actions. Temporal plans thus become sequences $A_0$, $A_1$, $A_2$ of legal sets of actions in which all actions in each set $A_i$ start their execution at the same time $t_i$. The end or completion time of an action $a$ in $A_i$ is thus $t_i + d(a)$ where $d(a)$ is the duration of $a$. The start times $t_i$ are defined implicitly in the plan as follows: $t_0 = 0$ and $t_{i+1}$ is given by the end time of the first action in $A_0, \ldots, A_i$ that completes after $t_i$. The states $s_i$ are defined in a similar way. The initial state $s_0$ is given, while $s_{i+1}$ is a function of the state $s_i$ at time $t_i$ and the set of actions $A^i$ in the plan that complete exactly at time $t + 1$; i.e., $s_{i+1} = f_T(A^i, s_i)$. The state transition function $f_T$ is obtained from the representation of the individual actions; e.g., if actions are represented in Strips and actions with conflicting adds and deletes are not legal, then $s_{i+1}$ will be obtained by 'adding' to $s_i$ all the atoms in the add lists of the actions in $A^i$ and deleting all the atoms in the delete lists. Clearly, $s_{i+1} = s_i$ if $A_i$ is empty.

The representation of temporal plans above is not complete, as for example, it cannot represent a plan in which some arbitrary slack is inserted between two actions. Yet the representation is complete 'enough' as if there is a plan that achieves the goal, for most reasonable optimality criteria, there is a plan in this representation that achieves the goal and is as good. A similar temporal model is used in (Bacchus & Ady 2001) while a regressed version of this model is used in (Haslum & Geffner 2001).

The definitions above provide a state model for temporal planning along the lines of model S1–S6 for classical planning with primitive actions replaced by *legal sets of actions.* Namely, a (valid) temporal plan is a sequence of legal sets of actions mapping the initial state into a goal state. For the rest, the models are the same, except for the cost structure which we haven't yet defined. In sequential planning, the overall cost of a plan $P = a_0, a_1, \ldots, a_n$ is defined as the sum $\sum_{i=0,n} c(a_i, s_i)$ where $s_i$ is either the initial state ($i = 0$) or the state that follows action $a_{i-1}$ ($i > 0$). This cost structure is much more flexible than the one used in classical planning where $c(a, s) = 1$, and it's often quite adequate in the sequential setting. In the temporal setting, on the other hand, this cost structure is not adequate for capturing standard optimality criteria such as the minimization of the *makespan* of a plan, the makespan being the max completion time of all actions in the plan. Indeed, the makespan of a plan cannot be obtained by adding up action costs of the form $c(A_i, s_i)$ for the sets of actions $A_i$ in the plan, as the contribution of the actions in $A_i$ to the makespan depends on the actions taken at the previous steps $A_0, \ldots,$ $A_{i-1}$. Nonetheless, it is sufficient to define the action costs as $c(A_i, s_i | A_0, \ldots, A_{i-1})$ and then minimize the total cost $c(A_0, s_0) + \sum_{i=1,n} c(A_i, s_i | A_0, \ldots, A_{i-1})$. Then, if we denote the last completion time of an action in $A_0, \ldots, A_i$ as $E_i$, the makespan minimization criterion results from setting $c(A_i, s_i | A_0, \ldots, A_{i-1})$ to $E_i - E_{i-1}$, for $i > 0$, and $c(A_0, s_0)$ to $E_0$. The makespan minimization criterion ignores the state component in the action costs, yet this can be used to define more complex optimality criteria such as those taking into account the value of some variables or the use of resources.

While from the similarity between the model for sequential planning and the model for temporal planning both tasks appear to be close from a *mathematical point of view*, they are quite different from a *computational point of view*. Indeed, while heuristic search is probably the best current approach for optimal and non-optimal sequential planning, it does *not* represent the best approach for parallel planning (temporal planning with actions of unit durations only) or temporal planning (yet see (Haslum & Geffner 2001; Do & Kambhampati 2001)). The problem is the *branching factor:* indeed, if there are $n$ primitive actions applicable in a state, the branching factor of the sequential space is $n$, while the branching factor of the temporal or parallel space is $2^n$ (namely, there are up to $2^n$ possible legal sets of primitive actions). SAT and CSP approaches branch in a different way, and through suitable constraints and constraint propagation rules may be able to integrate the pruning afforded by heuristic functions in the heuristic search framework with a more convenient exploration of the space of plans (see the discussion on branching below). Similarly, while partial order planners (POP) have lost favor in recent years to heuristic and SAT/CSP approaches, they are likely to be suitable for parallel and temporal planning if extended with suitable heuristic estimators and propagation rules. See (Nguyen & Kambhampati 2001) for a recent heuristic POP planner, (Jonsson *et al.* 2000) for a constraint-based temporal POP planner, and (Geffner 2001) for a general formulation of temporal planning as 'branch and bound' that connects heuristic and POP approaches in classical planning to constraint-based approaches used in scheduling. For more on the relation between planning and scheduling see (Smith, Frank, & Jonsson 2000).

## Languages

The models above provide a mathematical characterization of the basic planning tasks, their solutions, and their optimal solutions. These models however do not provide a convenient *language* for encoding planning problems. This is because the explicit characterization of the state space and state transitions is feasible only in small problems. In large problems, the state space and state transitions need to be represented *implicitly* in a logical *action language*, normally through a set of (state) variables and action rules. A good action language is one that supports compact encodings of the models of interest. In AI Planning, the standard language for many years has been the Strips language introduced in (Fikes & Nilsson 1971). While from a logical point of view, Strips is a very limited language, Strips is well known and will help us to illustrate the relationship between planning *languages* and planning *models*, and to motivate some of the extensions that have been proposed.

The Strips language comprises two parts: a language for describing the world and a language for describing how the world changes. The first is called the *state language* or simply the language, and the second, the *operator language.* We consider the Strips language as used currently in planning (e.g., the Strips subset of PDDL (Fox & Long 2001)), rather than the original version of Strips that is more complex.

The Strips language $\mathcal{L}$ is a simple logical language made up of two types of symbols: relational and constant symbols. In the expression $on(a, b)$, *on* is a relational symbol of arity 2, and $a$ and $b$ are constant symbols. In Strips, there are no functional symbols and the constant symbols are the only *terms.* The *atoms* are defined in a standard way from the combination $p(t_1, \ldots, t_k)$ of a relational symbol $p$ and a tuple of terms $t_i$ of the same arity as $p$. Similarly the Strips *formulas* are obtained by closing the set of atoms under the standard propositional connectives. In Strips, only conjunctions are used and they are identified with sets of atoms.

A main difference between relational and constant symbols in Strips is that the former are used to keep track of aspects of the world that may change as a result of the actions (e.g., the symbol *on* in $on(a, b)$), while the latter are used to refer to objects in the domain (e.g., the symbols $a$ and $b$ in $on(a, b)$). More precisely, actions in Strips affect the denotation of relational symbols but not the denotation of constant symbols. For this reason, the former are said to be *fluent* symbols, and the latter, *fixed* symbols or *constants.*

The *operators* in Strips are defined over the set of atoms in $\mathcal{L}$. Each operator *op* has a precondition, add, and delete lists $Prec(op)$, $Add(op)$, and $Del(op)$ given by sets of atoms. Operators are normally defined by means of *schemas;* here we assume that such schemas have been grounded.

A Strips planning problem $P = \langle A, O, I, G \rangle$ consists of a tuple where $A$ stands for the set of all atoms (boolean variables) in the domain, $O$ is the set of operators, and $I$ and $G$ are sets of atoms defining the *initial* and *goal* situations. The problem $P$ defines a deterministic state model $\mathcal{S}(P)$ like S1–S6 above where

A1. the states $s$ are sets of atoms from $A$

A2. the initial state $s_0$ is $I$

A3. the goal states are the states $s$ such that $G \subseteq s$

A4. $A(s)$ is the set of operators $o \in O$ s.t. $Prec(o) \subseteq s$

A5. the state transition function $f$ is such that $f(a, s) = s + Add(a) - Del(a)$ for $a \in A(s)$

A6. costs $c(a, s)$ are all equal to 1

This mapping defines the semantics of a Strips planning problem $P$, whose solution is given by the solution of the state model $\mathcal{S}(P)$.

Strips allows for compact encodings of state models yet in many cases these encodings could be improved substantially by moving Strips in the direction of a truly first-order logical language with negated literals, conditional effects, quantification, function symbols, etc. (Pednault 1989). Actually, the changes needed to accommodate these extensions are relatively minor once the logical relationship between the state language $\mathcal{L}$ and the states $s$ in the model is made explicit.[2] Basically, *the states are suitable, finite representations of the logical interpretations of the state language* $\mathcal{L}$ (Geffner 2000). A logical interpretation must assign a denotation to every constant, function, and relational symbol in $\mathcal{L}$, and implicitly, through the standard composition rules, a denotation to every term and formula. Furthermore, if the denotation of non-fluent symbols, is fixed, states must encode the denotation of fluent symbols only. In Strips, the only fluent symbols are the relational symbols and a state $s$ encodes their denotation by enumerating all the atoms that are true (it is also implicitly assumed that the constant symbols denote different objects in the domain and that there are no other objects). Actually, any first-order formula involving these constant and relational symbols can be evaluated in a state provided that no other non-logical symbols are used. Function symbols, whether fluent or not, can be added in a similar way provided that their denotation can be represented finitely. This can be achieved by forcing their arguments to take values in a finite domain.

If states are suitable representations of the denotation of fluent symbols, expressions in the *action language* (the operators in Strips) prescribe how these denotations change as a result of the actions. In general terms, an action $a$ is characterized by a precondition and several effect expressions. The precondition is an arbitrary formula which must be true in the state for the action to be applicable. In Strips, preconditions are sets (conjunctions) of atoms, yet in more expressive state languages they can be arbitrary formulas. Effect expressions are more subtle, and each effect prescribes how the denotation of a fluent symbol changes as a result of the action. For a relational fluent symbol $p$, an effect of an action $a$ will have the general form:

$$\text{if } A, \text{ then } p(t) := true/false$$

meaning that in the next state the denotation of $p$ must be updated to make $p(t)$ true (false) when the denotation of $A$ is true in $s$. In Strips, $A$ is empty, the term $t$ only contains non-fluent symbols, and atoms made true go into the add list and atoms made false into the delete list. In the more general case, $t$ itself may contain fluent symbols and it makes sense to evaluate this tuple of terms in the state $s$ where the action is executed. For functional fluent symbols $f$ (including constant fluent symbols), an effect expression will thus have the general form

$$\text{if } A, \text{ then } f(t) := t'$$

meaning that in the next state the denotation of $f$ must updated so that $f$ maps the value of the term $t$ into the value of the term $t'$, both obtained in the state $s$. For example, an effect like

$$\text{if } x < 10, \text{ then } x := x + 1$$

says that the value of $x$ should be increased by one in the next state if $x$ is currently smaller than 10. For fluent sym-

[2]The relationship between action languages and the semantical structures that they describe has been clarified along the years by work in the area of Reasoning about Change; e.g., see (Gelfond & Lifschitz 1993; Sandewall 1994).

bols which are not affected by any action, their denotation is assumed to persist.

Domain-independent planners with expressive state and action languages of this type, include GPT (Bonet & Geffner 2000) and MBP (Bertoli *et al.* 2001), both of which provide additional constructs for expressing non-determinism and sensing. Most knowledge-based planners provide very rich modeling languages, often including facilities for representing time and resources; see (Wilkins 1988; Currie & Tate 1991; Levesque *et al.* 1997; Bacchus & Kabanza 2000; Jonsson *et al.* 2000; Chien *et al.* 2000; Kvarnström, Doherty, & Haslum 2000). For the last version of the standard PDDL language see (Fox & Long 2001), while for complexity issues that arise as the basic Strips language is extended, see (Nebel 1999)

## Computation

While models and languages are key aspects in planning research, the current excitement in the field is to a large extent the result of recent advances in the resolution of Strips planning problems. Most of these advances followed the work on Graphplan, a planner introduced in (Blum & Furst 1995) which was shown to be more powerful than other approaches at the time, namely partial and total ordered planners like (Penberthy & Weld 1992; Fink & Veloso 1996). Graphplan was followed by Satplan (Kautz & Selman 1996; 1999), a SAT-based planner and HSP (Bonet, Loerincs, & Geffner 1997; Bonet & Geffner 2001), an heuristic search planner (see also (McDermott 1996)). In the Strips track of the first 1998 AIPS Planning Competition (McDermott 2000), all the planners were based on either Graphplan, SATPlan, or HSP, and they all were shown to be solve much larger problems than could be solved previously. The same remained true for the Strips track of the second 2000 AIPS Planning Contest (Bacchus 2001), that attracted 13 planners. In this last contest, heuristic search planners did best, with four out of the top five planners doing heuristic search one way or the other, and with FF (Hoffmann & Nebel 2001) doing best of all. The contest, however, is basically concerned with sequential planning, and thus these results do not carry much meaning for parallel or temporal planning, aspects that are expected to be addressed in the next planning contest.

In this section, we will review some of the ideas exploited by some of these recent planners, and then we'll move to temporal and uncertainty planning. We'll focus mostly on *optimal planning.* The reason is methodological; while there is always room for new, smarter non-optimal search algorithms, optimal search, an in particular optimal search in linear space (Korf 1993), is mostly the result of two operations: the branching scheme, defining how the space of solutions is searched, and the pruning scheme, defining how partial solutions are pruned. At the same time, ideas useful in optimal planning can often be used in non-optimal planning, while the opposite is seldom true.

From this perspective, an heuristic search planner like HSPr* (Haslum & Geffner 2000) does branching by regression and prunes a partial solution (a plan tail) when the accumulated cost plus the remaining cost (as estimated by an admissible heuristic) exceeds a bound. Similarly, a planner like Satplan does branching by selecting a variable and trying each of its values, pruning a 'partial plan' when a suitable form of constraint propagation (unit resolution) detects a contradiction. In the middle, partial-order planners offer an alternative branching scheme, their weakness lying in the lack of good pruning criteria.

### Heuristic Search

Heuristic search planners map planning problems into search problems that are solved with an heuristic function derived automatically from the problem representation. Then they use this heuristic to guide the search for plans either forward from the initial state (progression planners) or backward from the goal (regression planners) using algorithms such as A*, IDA*, or others. The derivation and use of these heuristic functions is the key element and the main novelty in these planners. Different types of heuristic functions have been formulated in planning and all of them correspond to optimal cost functions of relaxed problems or suitable approximations (Pearl 1983).

The first type of heuristics are based on a relaxation in which the delete lists of all operators is ignored (i.e., they are assumed empty). This simplification, however, does not make the problem of finding optimal plans tractable, thus additional simplifications are needed. The heuristic in planners like Unpop and HSP further assumes that atoms are independent and thus that the cost of achieving a set of atoms corresponds to the sum of the costs of achieving each atom in the set. The resulting heuristic, called the *additive heuristic,* is quite informative and can be computed reasonably fast for each new node. It can be formally defined by a fixed point equation of the form

$$h_+(p) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ \min_{a \in O(p)} [1 + h_+(Prec(a))] & \text{otherwise} \end{cases} \tag{5}$$

where $O(p)$ stands for the set of operators that add $p$ and $h_+(C)$ is defined for sets of atoms $C$ as

$$h_+(C) \stackrel{\text{def}}{=} \sum_{q \in C} h_+(q) \tag{6}$$

The heuristic $h_+(s)$ of a state is $h_+(G)$ where $G$ is the goal, and is obtained by solving (5) above with single-source shortest path algorithms. The additive heuristic, however, is not *admissible* (i.e., it is not a lower bound) and hence it's not useful for optimal planning. A different (non-admissible) heuristic based also on the delete-list relaxation is the FF heuristic. FF does not assume that atoms are independent but rather solves the relaxation suboptimally, and in the way, extracts useful information for guiding a hill-climbing search. For a detailed comparison between FF and HSP, see (Hoffmann & Nebel 2001).

A different relaxation is used in (Haslum & Geffner 2000) for defining a family of polynomial and admissible heuristics $h^m$ for $m = 1, 2, \ldots$. These heuristics take positive and negative interactions into account, and follow from a relaxation in which the cost of a set of atoms $C$ with size $|C| > m$ is approximated (recursively) by the cost of the mostly costly set $D$ in $C$ of size $m$. This relaxation can be formalized by a set of fixed point equations like (5) which can also be solved by shortest path algorithms. Indeed, for $m = 1$, i.e., when the cost of a set of atoms is assumed to be given by the cost of the most costly atom in the set, the resulting equations for $h^m$ are like (5)–(6) with the summation in (6) replaced by maximization. For $m > 1$, the resulting equations and their computation are more complex. The planner HSPr* uses the heuristic $h^2$, which is computed only once, within an IDA* regression search. (Haslum & Geffner 2000) also develops $h^m$ estimators por *parallel planning*. While in (Bonet & Geffner 1999) the point is made that Graphplan can be understood as an heuristic search planner doing IDA* regression search with an heuristic function $h_G$ implicitly encoded in the plan graph, in (Haslum & Geffner 2000) it is shown that the $h_G$ heuristic corresponds precisely to the $h^m$ heuristic for parallel planning with $m = 2$.

A more recent relaxation model used for deriving heuristics in planning is based on the idea of pattern databases (Culberson & Schaeffer 1998). Pattern databases have been proposed in the context of the 15-puzzle and have also been used for solving the Rubik's cube optimally (Korf 1998). The general idea of pattern databases is to project the state space $S$ of a problem into a smaller state space $\hat{S}$ that can be solved optimally and exhaustively. Then the heuristic $h(s)$ for the original state space is obtained from the solution cost $\hat{h}^*(\hat{s})$ of the projected state $\hat{s}$ in the relaxed space $\hat{S}$. For example, if $S$ comprises the possible values of a given set of multi-valued variables, then the projected space $\hat{S}$ can be obtained by treating several values as the *same* value, thus collapsing a potentially large collection of states into a single state (Holte & Hernadvolgyi 1999). Alternatively, we can perform abstraction in the set of variables rather than in the set of values. In the Strips setting, projected state spaces can be obtained by removing a number of atoms from the model; i.e., removing them from the initial state, the goal, and the operators. Then the projected problem, if sufficiently small, can be solved optimally from all initial states, and the resulting costs can be stored in a table providing lower bounds for the original problem. Heuristics resulting from different projections (pattern dbs) can be combined with the max operator to provide still more informed heuristics. See (Edelkamp 2001) for a more elaborated scheme that uses pattern databases in planning.

Pattern databases are just one of a number of novel and powerful ideas that have been recently developed in the area of problem solving and that are likely to have influence in planning. See (Korf 2000) and (Junghanns & Schaeffer 1999).

## Branching

Optimal planning and search is mostly branching and pruning. While the importance of lower bounds or admissible heuristics for pruning is well understood in AI, the importance of good branching rules is seldom mentioned. Indeed, the index of standard AI textbooks, for example, have entries for 'branching factor' but not for branching itself. Still the way the space of solutions is explored has a strong influence on performance. In AI problem solving, branching has been, for the most part, applying all possible actions. This type of forward branching, as well as its opposite, backward branching, where solutions are constructed backward from the goal, works quite well in problems like the 15-puzzle, Rubik's cube, and most sequential planning problems. Yet, for other domains like the Travelling Salesman Problem (Balas & Toth 1985), the Job Shop Scheduling Problem (Beck & Fox 1998), and others, more convenient branching schemes have been developed, which are likely to be necessary in tasks like temporal planning where the branching factor of progression and regression planners blows up. For example in the TSP, one can perform branching by selecting an edge connecting a city $i$ to a city $j$, then creating two subproblems, one in which city $i$ is followed by city $j$ in the tour, the other in which is not. This branching rule, along with a suitable lower bound estimator for the partial tours, works much better for larger TSPs than the most straightforward approach of starting in one city and then connecting the last visited city with each non-visited city.

In AI Planning, the issue of branching has been considered but in a different form: as the choice of the space in which to perform the search. State-space planners, namely progression and regression planners, search in the space of states, while partial-order planners search in the space of plans (Weld 1994). This is a useful distinction yet it does not show what the two approaches have in common and what they have in common with SAT and CSP approaches. All planners indeed search in the space of plans. It just happens that state-space planners are designed to build plans from the head or the tail only, and as a result, the resulting partial plan heads or tails can be suitably summarized by the information obtained by progressing the initial state through the plan head or regressing the goal through the plan tail. This is useful for computing the estimated cost $f(p)$ of the best complete plans that extend a given partial plan $p$. In state-based or directional planners this estimated cost can be split into two: the accumulated cost of the plan $g(p)$ that depends on $p$ only, and the estimated cost of the remaining plan $h(s)$ that depends only on the state $s$ obtained by progression or regression and which summarizes the partial plan $p$ completely. In partial-order and SAT/CSP planners this split of the cost function $f(p)$ for a 'partial' plan $p$ is not possible. On the other hand, state-based planners, as we have seen, suffer from a high-branching factor in temporal planning, where the set of parallel macro actions is exponential in the number of primitive actions, and in a number of sequential domains like Sokoban (Junghanns & Schaeffer 1999), where the number of applicable actions is just too large.

There is no reason, however, for choosing between directional heuristic planners and non-directional non-heuristic planners. As in other combinatorial problems, it should be possible to combine informative lower bounds and effective branching rules, thus allowing us to prune partial solutions $p$ whose estimated completion cost $f(p)$ exceeds a bound $B$. One option is to devise good admissible estimators $f(p)$ for non-directional plans. Indeed the failure of partial-order planning in relation to Graphplan and other modern approaches is that POP is a smart but blind search (branching) scheme, and hence, cannot compete with informed search algorithms such as Graphplan or HSP. The situation for SAT and CSP approaches is different. First of all, SAT and CSP approaches such as (Kautz & Selman 1999; Rintanen 1998; Do & Kambhampati 2000; M. Baioletti & Milani 2000) are not blind as they are all built on top of the plan graph constructed by Graphplan or a suitable SAT or CSP translation of it. The plan graph, as we have seen, encodes an informative and admissible heuristic function. In addition, these systems explicitly represent the condition $f(p) \leq B$ that the estimated completion cost $f(p)$ of a partial plan $p$ has not to exceed a bound $B$. This is accomplished through suitable clauses or constraints that are checked for consistency in every node. For example, if the goal is $G = \{q, r\}$ and the bound $B$ is 10, then a planner like Blackbox will force the constraint that $q$ and $r$ must be true at time $10$ by adding the clauses $q_{10}$ and $r_{10}$. Thus while SAT and CSP approaches do not perform explicit lower bound computations, they check the pruning condition $f(p) \leq B$ implicitly by constraint propagation and consistency checking. Whether explicit lower bound computations or consistency checking through suitable constraint propagation rules is the best approach for reasoning with non-directional plans depends on which of the two strategies yields more pruning. Actually, the best method will probably emerge from an integration of the two approaches as can be seen from recent work in constraint programming (Caseau & Laburthe 1994; Focacci, Lodi, & Milano 1999).

## Search in Non-Deterministic Spaces

Heuristic and constraint-based approaches, so powerful in the deterministic setting, are not directly applicable to problems involving non-determinism and feedback, as the solution of these problems is not a sequence of actions but a function mapping states into actions. The standard methods for solving such problems are not based on heuristic or constraint-based search but on dynamic programming (Bellman 1957; Bertsekas 1995). Dynamic programming (DP) methods compute a value function over *all* states, and use this function to define the policy. The greedy policy $\pi_V(s)$ relative to a given value function $V$ corresponds to the function that maps states $s$ into actions $a$ that minimize the worst cost or the expected cost of reaching the goal from $s$

$$\pi_V(s) \stackrel{\text{def}}{=} \text{argmin}_{a \in A(s)} \left( c(a,s) + \max_{s' \in F(a,s)} V(s') \right)$$

$$\pi_V(s) \stackrel{\text{def}}{=} \text{argmin}_{a \in A(s)} \left( c(a,s) + \sum_{s' \in S} P_a(s'|s)\, V(s') \right)$$

according to whether state transitions are modeled non-deterministically or probabilistically. While standard heuristic search methods like A* or IDA* cannot be used to search for policies, any heuristic function $h$ determines a greedy policy $\pi_V$ for $V = h$. Moreover, one of the basic results in DP is that this greedy policy is *optimal* when $V$ is the optimal cost function. DP methods like value iteration thus aim to compute the optimal cost function or a suitable approximation of it, and plug that function into the greedy policy. The optimal cost function is the solution of the Bellman equation

$$V(s) = \min_{a \in A(s)} \left( c(a,s) + \max_{s' \in F(a,s)} V(s') \right)$$

$$V(s) = \min_{a \in A(s)} \left( c(a,s) + \sum_{s' \in S} P_a(s'|s)\, V(s') \right)$$

for the non-deterministic and stochastic cases respectively, in both cases with $V(s) = 0$ for all goal states. Value iteration solves the Bellman equation by plugging an estimate $V_i$ function on the right hand side, and obtaining a new value function $V_{i+1}$ on the left hand side. This process is iterated until a fixed point is reached (in the probabilistic case, the convergence is defined in a slightly different way, see (Bertsekas 1995)). In asynchronous DP, a single vector $V$ is used on the left and right, and the iterations are updates on this vector. Moreover, rather than performing a parallel update over all states in $V$, some arbitrary subset of states is selected for update in each iteration. The same convergence guarantees exist, as long as all states are updated sufficiently often (Bertsekas 1995).

DP methods work well for spaces containing hundreds of thousands of states, or even few millions. For larger spaces, the time and space requirements of pure DP methods is prohibitive. This is in contrast to heuristic search methods for deterministic problems that can deal with huge state spaces provided a good heuristic function is used for avoiding consideration of most states.

In the last few years, promising strategies that integrate DP and heuristic search methods have been proposed. Real time dynamic programming (Barto, Bradtke, & Singh 1995) is one such strategy. As an heuristic search method, RTDP performs iterated greedy searches using an heuristic function that is adjusted dynamically. More precisely, in every non-goal state $s$, the best action $a$ according to the heuristic is selected (i.e., $a = \pi_h(s)$) and the heuristic value $h(s)$ of the state $s$ is updated using Bellman equation. Then a random successor state of $s$ and $a$ is selected using the transition function or transition probabilities, and this process is repeated until the goal is reached. Barto et al. show two key results that generalize those proved earlier by Korf in the deterministic setting for the LRTA* algorithm (Korf 1990) (see also (Koenig & Simmons 1995)). First, that this dynamic greedy search cannot be trapped into loops forever, and thus, that it eventually reaches the goal (provided the space is suitably 'connected'). This is what's called a single trial of the algorithm. Second, that consecutive trials of the RTDP algorithm using the heuristic (value) function resulting from the previous trials, eventually results in a greedy policy that is optimal (provided that the heuristic used in the first trial is admissible). The importance of these results is that DP updates in the RTDP algorithm are focused on the states visited in the search, and that this set of states can be very small in comparison with the whole state space if the initial heuristic is sufficiently informed. In particular this may result in some states never being visited, something that makes RTDP different from other (asynchronous) DP algorithms and more similar to heuristic search algorithms in AI.

A more recent heuristic DP algorithm is LAO* (Hansen & Zilberstein 2001). LAO* is an extension of the well known AO* algorithm (Nilsson 1980; Pearl 1983) that replaces the backward induction step in AO*, which is suitable for acyclic graphs, into a full DP step. While different on the surface, LAO* and RTDP are similar and both can be seen as variations of a best-first algorithm in which a state in the best *partial* policy is selected for expansion in each step until the best partial policy becomes 'complete'. In the next few years, we are likely to get a better theoretical and empirical understanding of these and other heuristic DP algorithms, and of the heuristics that are needed to make them work in large non-deterministic spaces.

### Belief Space

Planning problems involving uncertainty but no feedback can be mapped into deterministic search problems in *belief space* (Section 2). These are the so-called conformant planning problems. The two key problems from a computational point of view, are the derivation of good heuristic functions in belief space, and the effective representation and update of belief states in large state spaces. In principle, any admissible heuristic resulting from assuming complete observability can be 'lifted' into an admissible heuristic for conformant planning, yet the resulting heuristics can be very poor. Other ideas are necessary for scaling up; a promising alternative is the use of *state projections* as found in pattern databases and the heuristics $h^m$. Non-admissible heuristics for guiding a greedy search in belief space have been recently proposed in (Bertoli & Cimatti 2002) which also deals with the problem of efficiently representing and updating belief states through the use of OBDDs. For other approaches to conformant planning, see (Rintanen 1999; Ferraris & Giunchiglia 2000).

For Planning with uncertainty *and* partial observability, the same problems need to be faced, and in the addition, the search in belief space becomes non-deterministic. There are very few planners that handle both uncertainty and partial observability. Probably the best current such planners are GPT (Bonet & Geffner 2000), an RTDP planner, and MBP (Bertoli *et al.* 2001), a model checking planner. While planning with uncertainty and feedback is very hard, both systems can model and solve non-trivial problems. Both are available in the web and interested readers are encouraged to give them a try.

## Conclusions

We have approached the problem of planning from a perspective that makes a clear distinction between planning languages, models, and algorithms. AI Planning, from this point of view, is about the representation and resolution of certain classes of models. We have also discussed some of the ideas that we think are most important from a computational point of view: heuristic functions, branching and constraint propagation rules, search in non-deterministic spaces, and search in belief spaces.

It is an exciting time to be doing research in planning. New approaches and empirical standards have opened up the field and progress in the last few years has been very fast. There is also a convergence at many levels with other areas concerned with modeling and problem solving like Constraint Programming and Combinatorial Optimization. More complex problems can now be modeled and solved, and this trend is likely to continue over the next few years. This is likely to have an impact on the use of planning tools outside of academia which is still limited.

## Acknowledgments

Most of my work in planning has been in collaboration with Blai Bonet while I was at the Universidad Simón Bolívar in Caracas, Venezuela. I've also collaborated with Patrik Haslum from Linköping University and have benefited from conversations with many other people, too many to mention here.

## References

Astrom, K. 1965. Optimal control of markov decision processes with incomplete state estimation. *J. Math. Anal. Appl.* 10:174–205.

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: A forward chaining approach. In *Proc. IJCAI-01*, 417–424.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Bacchus, F. 2001. The 2000 AI Planning Systems Competition. *Artificial Intelligence Magazine* 22(3).

Balas, E., and Toth, P. 1985. Branch and bound methods. In *et al.*, E. L. L., ed., *The Traveling Salesman Problem*. Essex: John Wiley and Sons. 361–401.

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Beck, J., and Fox, M. 1998. A generic framework for constraint-directed scheduling. *AI Magazine* 19(4).

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Bertoli, P., and Cimatti, A. 2002. Improving heuristics for planning as search in belief space. In *Proc. AIPS-2002*.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proc. IJCAI-01*.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control, Vols 1 and 2*. Athena Scientific.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, 1636–1642. Morgan Kaufmann.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proceedings of ECP-99*, 359–371. Springer.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. of AIPS-2000*, 52–61. AAAI Press.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, 714–719. MIT Press.

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: structural assumptions and computational leverage. In *Proceedings of EWSP-95*.

Caseau, Y., and Laburthe, F. 1994. Improved CLP scheduling with task intervals. In *Proc. ICLP-94*, 369–383. MIT Press.

Chien *et al.*, S. 2000. Aspen – automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps2000*.

Culberson, J., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):319–333.

Currie, K., and Tate, A. 1991. O-Plan: the open planning architecture. *Artificial Intelligence* 52(1):49–86.

Daniele, M.; Traverso, P.; and Vardi, M. Y. 1999. Strong cyclic planning revisited. In *Proceedings of ECP-99*, 35–48. Springer.

Dean, T.; Kaebling, L.; Kirman, J.; and Nicholson, A. 1993. Planning with deadlines in stochastic domains. In *Proceedings AAAI93*, 574–579. MIT Press.

Do, M. B., and Kambhampati, S. 2000. Solving planning-graph by compiling it into CSP. In *Proc. AIPS-00*, 82–91.

Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In *Proc. ECP 2001*, 82–91.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. ECP 2001*.

Ferraris, P., and Giunchiglia, E. 2000. Planning as satisfiability in nondeterministic domains. In *Proceedings AAAI-2000*, 748–753.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 1:27–120.

Fink, E., and Veloso, M. 1996. Formalizing the PRODIGY planning algorithm. In Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*. IOS Press (Amsterdam). 261–272.

Focacci, F.; Lodi, A.; and Milano, M. 1999. Solving TSPs with time windows with constraints. In *Proc. ICLP-99*. MIT Press.

Fourer, R.; Gay, D.; and Kernighan, B. W. 1993. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. At `www.dur.ac.uk/d.p.long/competition.html`.

Geffner, H. 2000. Functional strips. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer. 187–205.

Geffner, H. 2001. Planning as branch and bound and its relation to constraint-based approaches. Technical report, Universidad Simón Bolívar. Available at `www.ldc.usb.ve/~hector`.

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *J. of Logic Programming* 17:301–322.

Hansen, E., and Zilberstein, S. 2001. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129:35–62.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proc. of the Fifth International Conference on AI Planning Systems (AIPS-2000)*, 70–82.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proc. ECP-01*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 2001:253–302.

Holte, R., and Hernadvolgyi, I. 1999. A space-time tradeoff for memory-based heuristics. In *Proceedings AAAI-99*, 704–709. Mit Press.

Jonsson, A.; Morris, P.; Muscettla, N.; and Rajan, K. 2000. Planning in interplanetary space: Theory and practice. In *Proc. AIPS-2000*.

Junghanns, A., and Schaeffer, J. 1999. Domain-dependent single-agent search enhancements. In *Proc. IJCAI-99*. Morgan Kaufmann.

Kaebling, L.; Littman, M.; and Cassandra, T. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1–2):99–134.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, 1194–1201. AAAI Press / MIT Press.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and Graph-based planning. In Dean, T., ed., *Proceedings IJCAI-99*, 318–327. Morgan Kaufmann.

Koenig, S., and Simmons, R. 1995. Real-time search in non-deterministic domains. In *Proceedings IJCAI-95*, 1660–1667. Morgan Kaufmann.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42:189–211.

Korf, R. 1993. Linear-space best-first search. *Artificial Intelligence* 62:41–78.

Korf, R. 1998. Finding optimal solutions to Rubik's cube using pattern databases. In *Proceedings of AAAI-98*, 1202–1207. AAAI Press / MIT Press.

Korf, R. 2000. Recent progress on the design and analysis of admissible heuristic functions. In *Proc. AAAI-2000*, 1165–1750.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76:239–286.

Kvarnström, J.; Doherty, P.; and Haslum, P. 2000. Extending TALplanner with concurrency and resources. In *Proc. ECAI-2000*, 501–505.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming* 31:59–83.

Long, D., and Fox, M. 2000. Automatic synthesis and use of generic types in planning. In *Proceedings AIPS 2000*, 196–206.

M. Baioletti, S. M., and Milani, A. 2000. DPPlan: An algorithm for fast solution extraction from a planning graph. In *Proc. AIPS-2000*.

Marriot, K., and Stuckey, P. 1999. *Programming with Constraints*. MIT Press.

McDermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int. Conf. on AI Planning Systems (AIPS-96)*.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *Artificial Intelligence Magazine* 21(2):35–56.

Nebel, B. 1999. Compilation schemes: A theoretical tool for assessing the expressive power of planning formalisms. In *KI-99: Advances in Artificial Intelligence*, 183–194. Springer-Verlag.

Newell, A., and Simon, H. 1963. GPS: a program that simulates human thought. In Feigenbaum, E., and Feldman, J., eds., *Computers and Thought*. McGraw Hill. 279–293.

Newell, A., and Simon, H. 1972. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice–Hall.

Nguyen, X. L., and Kambhampati, S. 2001. Reviving partial order planning. In *Proc. IJCAI-01*.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Tioga.

Pearl, J. 1983. *Heuristics*. Addison Wesley.

Pednault, E. 1989. ADL: Exploring the middle ground between Strips and the situation calcules. In Brachman, R.; Levesque, H.; and Reiter, R., eds., *Proc. KR-89*, 324–332. Morgan Kaufmann.

Penberthy, J., and Weld, D. 1992. Ucpop: A sound, complete, partiall order planner for adl. In *Proceedings KR'92*.

Rintanen, J. 1998. A planning algorithm not based on directional search. In *Proceedings KR'98*, 617–624. Morgan Kaufmann.

Rintanen, J. 1999. Constructing conditional plans by a theorem prover. *J. of AI Research* 10:323–352.

Russell, S., and Norvig, P. 1994. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Sandewall, E. 1994. *Features and Fluents. The Representation of Knowledge about Dynamical Systems*. Oxford Univ. Press.

Smith, D., and Weld, D. 1998. Conformant graphplan. In *Proceedings AAAI-98*, 889–896. AAAI Press.

Smith, D.; Frank, J.; and Jonsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1).

Sondik, E. 1971. *The Optimal Control of Partially Observable Markov Processes*. Ph.D. Dissertation, Stanford University.

Van Hentenryck, P. 1999. *The OPL Optimization Programming Language*. MIT Press.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in AI planning. In *Proceedings IJCAI-99*.

Weld, D. S. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.

Wilkins, D. E., and des Jardins, M. 2001. A call for knowledge-based planning. *AI Magazine* 22(1):99–115.

Wilkins, D. 1988. *Practical Planning: Extending the classical AI paradigm*. M. Kaufmann.

Wolsey, L. 1998. *Integer Programming*. Wiley.