Inference and Learning in Planning (Extended Abstract)

Hector Geffner

ICREA & Universitat Pompeu Fabra
C/Roc Boronat 138, E-08018 Barcelona, Spain
hector.geffner@upf.edu
http://www.tecn.upf.es/~hgeffner

Abstract. Planning is concerned with the development of solvers for a wide range of models where actions must be selected for achieving goals. In these models, actions may be deterministic or not, and full or partial sensing may be available. In the last few years, significant progress has been made, resulting in algorithms that can produce plans effectively in a variety of settings. These developments have to do with the formulation and use of general inference techniques and transformations. In this invited talk, I'll review the inference techniques used for solving individual planning instances from scratch, and discuss the use of learning methods and transformations for obtaining more general solutions.

1 Introduction

The problem of creating agents that can decide what to do on their own has been at the center of AI research since its beginnings. One of the first AI programs to tackle this problem, back in the 50's, was the General Problem Solver (GPS) that selects actions for reducing a difference between the current state and a desired target state [1]. Ever since then, this problem has been tackled in a number of ways in many areas of AI, and in particular in the area of Planning.

The problem of selecting actions for achieving goals, however, even in its most basic version – deterministic actions and complete information – is computationally intractable [2]. Under these assumptions, the problem of finding a plan becomes the well-known problem of finding a path in a directed graph whose nodes, that represent the possible states of the system, are exponential in the number of problem variables.

Until the middle 90's in fact, no planner or program of any sort could synthesize plans for large problems in an effective manner from a description of the actions and goals. In recent years, however, the situation has changed: in the presence of deterministic actions and full knowledge about the initial situation, classical planning algorithms can find plans quickly even in large problems with hundred of variables and actions [3,4]. This is the result of new ideas, like the automatic derivation of heuristic functions [5,6], and a established empirical methodology featuring benchmarks, comparisons, and competitions. Moreover, many of these planners are *action selection mechanisms* that can commit to the next action to do in real-time without having to construct a full plan first [7].

These developments, however, while crucial, do not suffice for producing autonomous agents that can decide by themselves what to do in environments where the two assumptions above (deterministic actions, complete information) do not apply. The more general problem of selecting actions in uncertain, dynamic and/or partially known environments arises in a number of contexts (a rover in Mars, a character in a video-game, a robot in a health-care facility, a softbot in the web, etc.), and has been tackled through a number of different methodologies:

1. *programming-based:* where the *desired behavior is encoded explicitly* by a human programmer in a suitable high-level language,
2. *learning-based:* where the *desired behavior is learned automatically* from trial-and-error experience or information provided by a teacher, or
3. *model-based:* where *the desidered behavior is inferred automatically* from a suitable description of the actions, sensors, and goals.

None of these approaches, however, or a combination of them, has resulted yet in a solid methodology for building agents that can display a robust and flexible behavior in real time in partially known environments. Programming agents by hand puts all the burden in the programmer that cannot anticipate all possible contingencies, leading to systems that are brittle. Learning methods such as reinforcement learning [8], are restricted in scope and do not deal with the problem of incomplete state information. Finally, traditional model-based methods, when applied to models that are more realistic than the ones underlying classical planning, have difficulties scaling up.

Planning in Artificial Intelligence represents the model-based approach to autonomous behavior: a planner is a solver that accepts a model of the actions, sensors, and goals, and produces a controller that determines the actions to do given the observations gathered (Fig. 1). Planners come in a great variety, depending on the types of models they target. Classical planners address deterministic state models with full information about the initial situation [9]; conformant planners address state models with non-deterministic actions and incomplete information about the initial state [10,11], POMDP planners address stochastic state model with partial observability [12], and so on.

In all cases, the models of the environment considered in planning are intractable in the worst case, meaning that brute force methods do not scale up. Domain-independent planning approaches aimed at solving these planning models effectively must thus *recognize* and *exploit* the structure of the individual

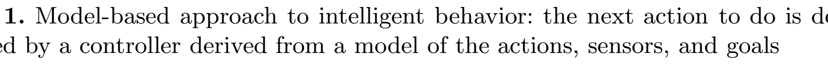

Fig. 1. Model-based approach to intelligent behavior: the next action to do is determined by a controller derived from a model of the actions, sensors, and goals

problems that are given. The key to exploiting this structure is *inference*, as in other AI models such as Constraint Satisfaction Problems and Bayesian Networks [13,14]. In the paper, we will go over the inference techniques that have been found computationally useful in planning research and identify areas where they could benefit from learning techniques as well. In this sense, planners solve problems from scratch by combining search and inference, and do not get any better as more instances from a given domain are solved. Learning should thus help planners to automatically extract domain knowledge that could be used to solve other domain instances more effectively, and in principle, without any search at all.

The paper is organized as follows. We consider the model, language, and inference techniques developed for classical planning, conformant planning, and planning with sensing, in that order. We focus on inference techniques of two types: heuristic functions and transformations. We then consider the use and role of inductive learning methods in planning, in particular, when plan strategies for a whole domain, and not for a single domain instance, are required.

2 Classical Planning

Classical planning is concerned with the selection of actions in environments that are *deterministic* and whose initial state is *fully known.* The model underlying classical planning can be described as a state space containing

- a finite and discrete set of states S,
- a *known initial state* $s_0 \in S$,
- a set $S_G \subseteq S$ of goal states,
- actions $A(s) \subseteq A$ applicable in each $s \in S$,
- a *deterministic transition function* $s' = f(a, s)$ for $a \in A(s)$, and
- *uniform action costs* $c(a, s)$ equal to 1.

A solution or *plan* in this model is a sequence of actions $a_0, \ldots, a_n$ that generates a state sequence $s_0, s_1, \ldots, s_{n+1}$ such that a_i is applicable in the state s_i and results in the state $s_{i+1} = f(a_i, s_i)$, the last of which is a goal state.

The cost of a plan is the sum of the action costs, which in this setting, corresponds to plan length. A plan is optimal it is has minimum cost, and the cost of a problem is the cost of an optimal plan.

Domain-independent classical planners accept a compact description of the above models, and automatically produce a plan (an optimal plan if the planner is optimal). This problem is intractable in the worst case, yet currently large classical problems can be solved using heuristic functions derived from the problem encodings.

A simple but still common language for encoding classical planning problems is Strips [9]. A problem in Strips is a tuple $P = \langle F, O, I, G \rangle$ where

- F stands for set of all *atoms* (boolean vars),
- O stands for set of all *operators* (actions),

- $I \subseteq F$ stands for the *initial situation*, and
- $G \subseteq F$ stands for the *goal situation.*

The actions $o \in O$ are represented by three sets of atoms from F called the Add, Delete, and Precondition lists, denoted as $Add(o)$, $Del(o)$, $Pre(o)$. The first, describes the atoms that the action o makes true, the second, the atoms that o makes false, and the third, the atoms that must be true for the action o to be applicable.

A Strips problem $P = \langle F, O, I, G \rangle$ encodes the state model $\mathcal{S}(P)$ where

- the states $s \in S$ are *collections of atoms* from F,
- the initial state s_0 is I,
- the goal states s are those for which $G \subseteq s$,
- the actions a in $A(s)$ are the ones in O such that $Prec(a) \subseteq s$, and
- the next state is $s' = f(a, s) = (s \setminus Del(a)) \cup Add(a)$.

All areas in Planning, and in particular Classical Planning, have become quite empirical in recent years, with competitions held every two years [15], and hundreds of benchmark problems available in PDDL, a standard syntax for planning that extends Strips [16].

The classical planners that scale up best can solve large problems with hundreds of fluents and actions [17,18]. These planners do not compute optimal solutions and cast the planning problem P as an *heuristic search problem* over the state space $\mathcal{S}(P)$ that defines a directed graph whose nodes are the states, whose initial node is the initial state, and whose target nodes are the states where the goals are true [19]. This graph is never made explicit as it contains a number of states that is exponential in the number of fluents of P, but can be searched quite efficiently with current heuristics.

Heuristic functions $h(s)$ provide an estimate of the cost to reach the goal from any state s, and are derived automatically from a relaxation (simplification) of the problem P [20]. The relaxation most commonly used in planning, called the delete-relaxation and denoted as P^+, is obtained by removing the delete lists from the actions in P. While finding the *optimal* solution to the relaxation P^+ is still NP-hard, finding just one *solution* is easy and can be done in low polynomial time.

The *additive heuristic*, for example, estimates the cost $h(p; s)$ of achieving the atoms p from s through the equations [19]:

$$h(p; s) = \begin{cases} 0 & \text{if } p \in s \\ h(a_p; s) & \text{otherwise} \end{cases}$$

where a_p is a *best support* for p in s defined as

$$a_p \;=\; \text{argmin}_{a \in O(p)} h(a; s)$$

$O(p)$ is the set of actions that add p in P, and $h(a; s)$ is

$$h(a; s) \;=\; cost(a) + \sum_{q \in Pre(a)} h(q; s) \;.$$

The cost of achieving the goal G from s is then defined as

$$h_{add}(s) = \sum_{p \in G} h(p; s) \ .$$

The heuristic h_{add} is not admissible (it's not a lower bound) but is informative and its computation involves the solution of a shortest-path problem in *atom space* as opposed to *state space*. A plan $\pi^+(s)$ for the relaxation P^+ can be obtained from the heuristic $h_{add}(s)$ by simply collecting the *best supports* recursively backwards from the goal [21]. This is actually the technique used in the state-of-the-art planner LAMA [18], winner of the 2008 International Planning Competition [15], that defines the heuristic $h(s)$ as the cost of this 'relaxed plan', and uses it in problems where action costs are not uniform. The search algorithm in LAMA is (greedy) best first search with the evaluation function $f(s) = h(s)$ and two open lists rather one, for giving precedence to the actions applicable in the state s that are most relevant to the goal according to $\pi^+(s)$; the so-called helpful actions [7].

3 Incomplete Information

The good news about classical planning is that it works: large problems can be solved quite fast, and the sheer size of a problem is not an obstacle to its solution. The bad news is that the assumptions underlying classical planning are too restrictive. We address now the problems that arise from the presence of *uncertainty* in the initial situation. The resulting problems are called *conformant* as they have the same form as classical plans, namely plain action sequences, but they must work for each of the initial states that are possible.

An example that illustrates the difficulties that arise from the presence of incomplete information in the initial situation is shown in Fig. 2. It displays a robot that must move from an uncertain initial location I, shown in gray, to the target cell G that must be reached with certainty. The robot can move one cell at a time, without leaving the grid: moves that would leave the agent out of the grid have no effects. The problem is very much like a classical planning

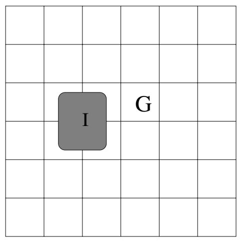

Fig. 2. A problem involving incomplete information: a robot must move from an uncertain initial location I shown in gray, to the target cell G with certainy. For this, it must locate itself into a corner and then head to G.

problem except for the uncertain initial situation I. The solutions to the problem, however, are quite different. Indeed, the best *conformant plan* for the problem must move the robot to a corner first, and then head with certainty to the target G. For example, for being certain that the robot is at the left lower corner of the grid, the robot can move *left* three times, and *down* three times. Notice that this is the opposite of reasoning by cases; indeed, the best action to do from each of the possible initial locations is not to move left or right, but up or right. Yet such moves would not help the robot reach the goal with certainty.

The model for the conformant planning problem is the model for classical planning but with the initial state s_0 replaced by a non-empty *set* S_0 of possible initial states. The Strips syntax for the problem $P = \langle F, O, I, G \rangle$ is also extended to let I stand for a *set of clauses* and not just a set of atoms, and O to include actions with effects L, positive or negative, that are *conditional* on a set of literals $L_1, \ldots, L_n$, written as $L_1, \ldots, L_n \rightarrow L$, where each L_i and L are positive or negative literals.

Conformant planning problems are no longer path-finding problems over a directed graph whose nodes are the *states* of the problem, but rather path-finding problems over a directed graph whose nodes are *sets of states*, also called *belief states* [22]. Belief states express the states of the world that are deemed possible to the agent. Thus, while in classical planning, the size of the (state) space to search is exponential in the number of variables in the problem; in conformant planning, the size of the (belief) space to search is exponential in the number of states. Indeed, conformant planning is harder than classical planning, as even the verification of conformant plans is NP-hard [23].

Conformant planners such as Contingent-FF, MBP, and POND [24,25,26], address the search in belief space using suitable belief representations such as OBDDs, that do not necessarily blow up with the number of states deemed possible, and heuristics that can guide the search for the target beliefs. Another approach that has been pursued recently, that turned out to be the most competitive in the 2006 Int. Planning Competition, is to automatically transform the conformant problems P into classical problems $K(P)$ that are solved by off-the-shelf classical planners.

The translation $K(P) = K_{T,M}(P)$ of a conformant problem P involves two parameters: a set of *tags* T and a set of *merges* M [27]. A tag t is a set (conjunction) of literals in P whose status in the initial situation I is not known, and a merge $m \in M$ is a collection of tags $t_1, \ldots, t_n$ that stands for the DNF formula $t_1 \vee \cdots \vee t_n$. Tags are assumed to represent consistent assumptions about I, i.e. $I \not\models \neg t$, and merges represent disjunctions of assumptions that follow from I; i.e. $I \models t_1 \vee \cdots \vee t_n$.

The fluents in $K_{T,M}(P)$, for the conformant problem $P = \langle F, O, I, G \rangle$ are of the form KL/t for each $L \in F$ and $t \in T$, meaning that "it is known that if t is true in the initial situation, L is true". In addition, $K_{T,M}(P)$ includes extra actions, called *merge actions*, that allow the derivation of a literal KL (i.e. KL/t with the "empty tag", expressing that L is known unconditionally) when KL/t' has been obtained for each tag t' in a merge $m \in M$ for L.

Formally, for a conformant problem $P = \langle F,O,I,G\rangle$, the translation defines the *classical problem* $K_{T,M}(P) = \langle F',O',I',G'\rangle$ where

$$\begin{aligned}
F' =& \{KL/t, K\neg L/t \mid L \in F\}\\
I' =& \{KL/t \mid \text{if } I \models t \supset L\}\\
G' =& \{KL \mid L \in G\}\\
O' =& \{a : KC/t \rightarrow KL/t,\ a : \neg K\neg C/t \rightarrow \neg K\neg L/t\\
&\mid a : C \rightarrow L \text{ in } P\} \cup \{\bigwedge_{t\in m} KL/t \rightarrow KL \mid m \in M_L\}
\end{aligned}$$

with t ranging over T and with the preconditions of the actions a in $K_{T,M}(P)$ including the literal KL if the preconditions of a in P include the literal L.

When $C = L_1,\ldots,L_n$, the expressions KC/t and $\neg K\neg C/t$ are abbreviations for $KL_1/t,\ldots,KL_n/t$ and $\neg K\neg L_1/t,\ldots,\neg K\neg L_n/t$ respectively. A rule $a : C \rightarrow L$ in P gets mapped into "support rules" $a : KC/t \rightarrow KL/t$ and "cancellation rules" $a : \neg K\neg C/t \rightarrow \neg K\neg L/t$; the former "adds" KL/t when the condition C is known in t, the latter undercut the persistence of $K\neg L/t$ except when (a literal in) C is known to be false in t.

The translation $K_{T,M}(P)$ is *sound*, meaning that the classical plans that solve $K_{T,M}(P)$ yield valid conformant plans for P that can be obtained by just dropping the merge actions. On the other hand, the *complexity* and *completeness* of the translation depend on the choice of tags T and merges M. The $K_i(P)$ translation, where i is a non-negative integer, is a special case of the $K_{T,M}(P)$ translation where the tags t are restricted to contain at most i literals. $K_i(P)$ is exponential in i and complete for problems with *conformant width* less than or equal to i. The planner T_0 feeds the $K_1(P)$ translation into the classical FF planner [7] and was the winning entry in the Conformant Track of the 2006 IPC [28].

4 Sensing and Finite-State Controllers

Most often problems that involve *uncertainty* in the initial state of the environment or in the action effects, also involve some type of *feedback* or *sensors* that provide partial state information. As an illustration of a problem of this type, consider the simple grid shown on the left of Fig. 3, where an agent starting in some cell between A and B, mut move to B, and then to A. In this problem, while the exact initial location of the agent is not known, it is assumed that the marks A and B are *observable.*

The solutions to problems involving observations can be expressed in many forms: as contingent plans [24], as policies mapping beliefs into actions [12], and as *finite-state controllers.* A finite-state controller that solves the problem above is shown on the right of Fig. 3. An arrow $q_i \rightarrow q_j$ between one controller state q_i and another (or the same) controller state q_i labeled with a pair O/a means to do action a and switch to state q_j, when o is observed in the state q_i. Starting in the controller state q_0, the controller shown tells the agent to move right until

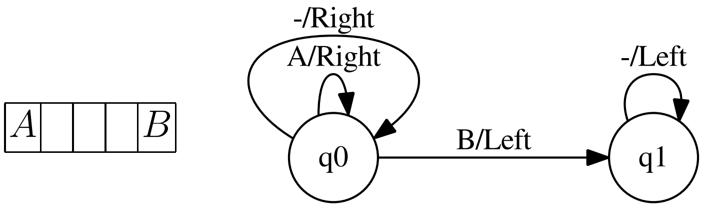

Fig. 3. *Left:* A problem where an agent, initially between A and B, must move to B and then back to A. *Right:* A finite-state controller that solves the problem.

observing B, and then to move left until observing A or B (the observation '-' means no observation).

Finite-state controllers such as the one displayed above have two features that make them more appealing than contingent plans and POMDP policies: they are often very *compact*, and they often quite *general* too. Indeed, the problem above can be changed in a number of ways and the controller shown would still work. For example, the *size of the grid* can be changed from 1×5 to $1 \times n$, the agent can be placed *initially* anywhere in the grid (except at B), and the actions can be made *non-deterministic* by the addition of 'noise'. This generality is well beyond the power of contingent plans or exact POMDP policies that are tied to a particular state space. For these reasons, finite-state controllers are widely used in practice, from controlling non-playing characters in video-games [29] to mobile robots [30,31]. Memoryless controllers or policies [32] are widely used as well, and they are nothing but finite-state controllers with a single state. The additional states provide finite-state controllers with memory that allows different actions to be taken given the same observation.

The benefits of finite-state controllers, however, come at a price: unlike contingent trees and POMDP policies, they are usually not derived automatically from a model but are written by hand; a task that is not trivial even in the simplest cases. There have been attempts for deriving finite-state controllers for POMDPs with a given number of states [33,34,35], but the problem can be solved approximately only, with no correctness guarantees.

Recently, we have extended the translation-based approach to conformant planning presented above [27], to derive finite-state controllers [36]. For this, the *control problem* P is defined in terms of a *conformant problem* with no preconditions, extended with a set O of *observable fluents*. The solution to the problem P is defined in terms of *finite state controllers* $\mathcal{C}_N$ with a given number N of *controller states*. This rules out sequential plans as possible solutions, as they would involve a number of controller states equal to the number of time steps in the plan.

The controller $\mathcal{C}_N$ is a set of tuples $t = \langle i, o, a, k \rangle$ that tell the agent to do a and switch to state q_k when the observation is o and the controller state is q_i. The key result is that a finite-state controller $\mathcal{C}_N$ that solves P can be obtained from the *classical plans* of a *classical problem* P_N obtained by a suitable translation from P, O, and N. The key idea in the translation is to replace each action a in P by an action $a(t)$, for each $t = \langle i, o, a, k \rangle$, so that the effects $C \rightarrow C'$ of a in

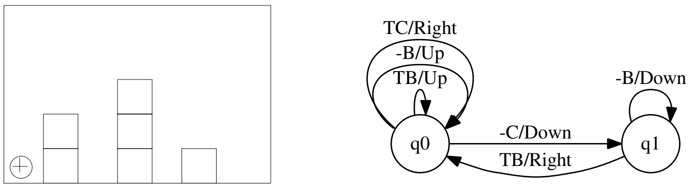

Fig. 4. *Left:* Problem where visual-marker (circle on the lower left) must be moved on top of a green block. The observations are whether the cell currently marked contains a green block (G), a non-green block (B), or neither (C); and whether this cell is at the level of the table (T) or not (–). *Right:* Finite-state controller that solves the problem for any number and arrangement of blocks.

P become effects $q_i, o, C \rightarrow \neg q_i, q_k, C'$ of $a(t)$ in P_N. That is, the effects of the action a are made conditional the observation o and state q_i in the actions $a(t)$ where $t = \langle i, o, a, k \rangle$.

Fig. 4 shows a more challenging problem solved in this way, resulting in a very compact and general controller. In the problem, shown on the left, a visual-marker (a circle on the lower left) must be moved on top of a green block . The observations are whether the cell currently marked contains a green block (G), a non-green block (B), or neither (C); and whether this cell is at the level of the table (T) or not ('-'). The visual marker can be moved one cell at a time in the four directions. This is a problem à la Chapman or Ballard, that have advocated the use of deictic representations of this sort [37,38]. The finite-state controller that results for this problem is shown on the right. Interestingly, it is a very compact and general controller: it involves two states only and can be used to solve the same problem for any number and arrangement of blocks. See [36] for details.

5 Learning and Generalized Policies

We illustrated above that it is possible to obtain from a concrete problem P, a finite-state controller that not only solves P but many variations too, including changes in the initial situation and action effects, and changes in the number of objects and size of the state space. This generalization does not follow from an *inductive* approach over many problem instances, but from a *deductive* approach over a single instance upon which the solution is guaranteed to be correct. The generalization is achieved from a *change in the representation of the solution:* while solutions to P that take the form of contingent plans or POMDP policies would not generalize to problems that involve a different state space, solutions that are expressed as compact finite-state controllers, often do. In principle, these techniques can be used to derive finite-state controllers for solving any instance of a given domain such as Blocks. Such general strategies exist; indeed, one such strategy for Blocks is to put all blocks on the table, and then build the desired towers in order, from the bottom up. Of course, this strategy is not optimal, and indeed no compact optimal strategy for Blocks exists. Yet, the approach presented above does not handle problems of this type. For this, first, observations must be defined on fluents that are not primitive in the problem, and which thus, must be conveniently discovered, like the fluent *above*, the transitive closure of the *on* predicate, that comes very handy in Blocks. Second, the resulting pool of observable fluents becomes then too large, so that the resulting translation P_N into a classical planning problem cannot even be constructed; the translation is indeed exponential in the number of observables. Interestingly, *inductive approaches* have been shown to be able to generate general strategies for domains like Blocks [39], and moreover, some of these inductive approaches do not require any background knowledge and work just with the definition of the planning domain and a small set of solved instances [40,41]. A interesting challenge for the future is the combination of *inductive* and *deductive* approaches for the derivation of general policies able to solve any instance of a given planning domain without search. From the discussion above, it seems that inductive methods are good for selecting informative features, while deductive methods are good for assembling these features into correct general policies. In this sense, planning appears to be an ideally rich application domain where learning adequate representations appears to be possible and critical for achieving both efficiency and generality.

Acknowledgments. This is joint work with a number of students and colleagues, in particular Blai Bonet and Hector Palacios. I thank Joao Gamma and the organizers of DS-09 for the invitation. The author is partially supported by grant TIN2006-15387-C03-03 from MEC, Spain.

References

1. Newell, A., Simon, H.: GPS: a program that simulates human thought. In: Feigenbaum, E., Feldman, J. (eds.) Computers and Thought, pp. 279–293. McGraw-Hill, New York (1963)
2. Bylander, T.: The computational complexity of STRIPS planning. Artificial Intelligence 69, 165–204 (1994)
3. Blum, A., Furst, M.: Fast planning through planning graph analysis. In: Proceedings of IJCAI 1995, pp. 1636–1642. Morgan Kaufmann, San Francisco (1995)
4. Kautz, H., Selman, B.: Pushing the envelope: Planning, propositional logic, and stochastic search. In: Proc. AAAI, pp. 1194–1201 (1996)
5. McDermott, D.: Using regression-match graphs to control search in planning. Artificial Intelligence 109(1-2), 111–159 (1999)
6. Bonet, B., Loerincs, G., Geffner, H.: A robust and fast action selection mechanism for planning. In: Proceedings of AAAI 1997, pp. 714–719. MIT Press, Cambridge (1997)
7. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302 (2001)
8. Sutton, R., Barto, A.: Introduction to Reinforcement Learning. MIT Press, Cambridge (1998)
9. Fikes, R., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 1, 27–120 (1971)

10. Goldman, R.P., Boddy, M.S.: Expressive planning and explicit knowledge. In: Proc. AIPS 1996 (1996)
11. Smith, D., Weld, D.: Conformant graphplan. In: Proceedings AAAI 1998, pp. 889–896. AAAI Press, Menlo Park (1998)
12. Cassandra, A., Kaelbling, L., Littman, M.L.: Acting optimally in partially observable stochastic domains. In: Proc. AAAI, pp. 1023–1028 (1994)
13. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
14. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, San Francisco (1988)
15. International Planning Competitions (2009), `http://icaps-conference.org/index.php/main/competitions`
16. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT (1998)
17. Helmert, M.: The Fast Downward planning system. Journal of Artificial Intelligence Research 26, 191–246 (2006)
18. Richter, S., Helmert, M., Westphal, M.: Landmarks revisited. In: Proc. AAAI, pp. 975–982 (2008)
19. Bonet, B., Geffner, H.: Planning as heuristic search. Artificial Intelligence 129(1–2), 5–33 (2001)
20. Pearl, J.: Heuristics. Addison-Wesley, Reading (1983)
21. Keyder, E., Geffner, H.: Heuristics for planning with action costs revisited. In: Proc. ECAI 2008 (2008)
22. Bonet, B., Geffner, H.: Planning with incomplete information as heuristic search in belief space. In: Proc. of AIPS 2000, pp. 52–61. AAAI Press, Menlo Park (2000)
23. Haslum, P., Jonsson, P.: Some results on the complexity of planning with incomplete information. In: Biundo, S., Fox, M. (eds.) ECP 1999. LNCS (LNAI), vol. 1809, pp. 308–318. Springer, Heidelberg (2000)
24. Hoffmann, J., Brafman, R.: Contingent planning via heuristic forward search with implicit belief states. In: Proc. ICAPS, pp. 71–80 (2005)
25. Bertoli, P., Cimatti, A., Roveri, M., Traverso, P.: Planning in nondeterministic domains under partial observability via symbolic model checking. In: Proc. IJCAI 2001 (2001)
26. Bryce, D., Kambhampati, S., Smith, D.E.: Planning graph heuristics for belief space search. Journal of AI Research 26, 35–99 (2006)
27. Palacios, H., Geffner, H.: From conformant into classical planning: Efficient translations that may be complete too. In: Proc. 17th Int. Conf. on Planning and Scheduling, ICAPS 2007 (2007)
28. Bonet, B., Givan, B.: Results of the conformant track of the 5th int. planning competition (2006), `http://www.ldc.usb.ve/~bonet/ipc5/docs/results-conformant.pdf`
29. Buckland, M.: Programming Game AI by Example. Wordware Publishing, Inc. (2004)
30. Murphy, R.R.: An Introduction to AI Robotics. MIT Press, Cambridge (2000)
31. Mataric, M.J.: The Robotics Primer. MIT Press, Cambridge (2007)
32. Littman, M.L.: Memoryless policies: Theoretical limitations and practical results. In: Cliff, D. (ed.) From Animals to Animats 3. MIT Press, Cambridge (1994)
33. Meuleau, N., Peshkin, L., Kim, K., Kaelbling, L.P.: Learning finite-state controllers for partially observable environments. In: Proc. UAI, pp. 427–436 (1999)

34. Poupart, P., Boutilier, C.: Bounded finite state controllers. In: Proc. NIPS, pp. 823–830 (2003)
35. Amato, C., Bernstein, D., Zilberstein, S.: Optimizing memory-bounded controllers for decentralized pomdps. In: Proc. UAI (2007)
36. Bonet, B., Palacios, H., Geffner, H.: Automatic derivation of memoryless policies and finite-state controllers using classical planners. In: Proc. ICAPS 2009 (2009)
37. Chapman, D.: Penguins can make cake. AI Magazine 10(4), 45–50 (1989)
38. Ballard, D., Hayhoe, M., Pook, P., Rao, R.: Deictic codes for the embodiment of cognition. Behavioral and Brain Sciences 20 (1997)
39. Khardon, R.: Learning action strategies for planning domains. Artificial Intelligence 113, 125–148 (1999)
40. Martin, M., Geffner, H.: Learning generalized policies from planning examples using concept languages. Appl. Intelligence 20(1), 9–19 (2004)
41. Yoon, S., Fern, A., Givan, R.: Inductive policy selection for first-order MDPs. In: Proc. UAI 2002 (2002)