Heuristics, Planning and Cognition

Hector Geffner

1 Introduction

In the book *Heuristics,* Pearl studies the strategies for the control of problem solving processes in human beings and machines, pondering how people manage to solve an extremely broad range of problems with so little effort, and how machines could do the same [Pearl 1983, pp. vii]. The central concept in the book, as captured in the title, are the *heuristics*: the "criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal" [Pearl 1983, pp. 3]. Pearl places special emphasis on heuristics that take the form of *evaluation functions* and which provide quick but approximate estimates of the distance or cost-to-go from a given state to the goal. These heuristic evaluation functions provide the search with a sense of direction with actions resulting in states that are closer to the goal being preferred. An informative heuristic $h(s)$ in the 15-puzzle, for example, is the well known 'sum of Manhattan distances', that adds up the Manhattan distance of each tile, from its location in the state s to its goal location.

The book *Heuristics* laid the foundations for the work in automated problem solving in Artificial Intelligence (AI) and is still a basic reference in the field. On the other hand, as an account of human problem solving, the book has not been as influential. A reason for this is that while the book devotes one chapter to discuss the derivation of *heuristics,* most of the book is devoted to the formulation and analysis of heuristic search *algorithms.* Most of these algorithms, such as A* and AO*, are complete and optimal, meaning that they will find a solution if there is one, and that the solution found will have minimal cost (provided that the heuristic does not overestimate the true costs). Yet, while people excel at solving a wide variety of problems almost effortlessly, it's only in puzzle-like problems where they need to restore to search, and then, they are not particularly good at it and are even worse when solutions must be optimal.

Thus, the account of problem solving in the book exhibits a gap that has been characteristic of AI systems, that result in programs that rival the best human experts in specialized domains but are no match to children in their general problem solving abilities.

In this article, I aim to present recent work in AI Planning, a form of *domain-independent problem solving,* that builds on Pearl's work and bears on this gap.

Planners are general problem solvers aimed at solving an infinite collection of problems automatically. The problems are instances of various classes of models all of which are intractable in the worst case. In order to solve these problems effectively thus, a planner must automatically recognize and exploit their structure. This is the key challenge in planning and, more generally, in domain-independent problem solving. In planning, this challenge has been addressed *by deriving the heuristic evaluations functions automatically* from the problems, an idea explored by Pearl and developed more fully in recent planning research. The resulting domain-independent planners are not as efficient as specialized solvers but are more general, and thus, behave in a way that is closer to people. Moreover, the resulting evaluation functions often enable the solution of problems with almost no search, and appear to play the role of the 'intuitions' and 'feelings' that guide human problem solving and have been difficult to capture explicitly by means of rules. We will see indeed how such heuristic evaluation functions are defined and computed in a domain-independent fashion, and why they can be regarded as relevant from a cognitive point of view.

The organization of the article is the following. We consider in order, planning models, languages, and algorithms (Section 2), the automatic extraction of heuristic evaluation functions and other developments in planning (Sections 3 and 4), the cognitive interpretation of these heuristics (Section 5), and then, more generally, the relation between AI and Cognitive Science (Section 6).

2 Planning

Planning is an area of AI concerned with the selection of actions for achieving goals. The first AI planner and one of the first AI programs was the General Problem Solver (GPS) developed by Newell, Shaw, and Simon in the late 50's [Newell, Shaw, and Simon 1958; Newell and Simon 1963]. Since then, planning has remained a central topic in AI while changing in significant ways: on the one hand, it has become *more mathematical,* with a variety of planning problems defined and studied; on the other, it has become *more empirical,* with planning algorithms evaluated experimentally and planning competitions held periodically.

Planning can be understood as representing one of the three main approaches for *selecting the action to do next*; a problem that is central in the design of autonomous systems, called often the *control problem* in AI.

In the *programming-based approach*, the programmer solves the control problem in its head and makes the solution explicit in the program. For example, for a robot moving in an office environment, the program may say to back up when too close to a wall, to search for a door if the robot has to move to another room, etc. [Brooks 1987; Mataric 2007].

In the *learning-based approach*, the control knowledge is not provided explicitly by a programmer but is learned by trial and error, as in reinforcement learning [Sutton and Barto 1998], or by generalization from examples, as in supervised learning [Mitchell 1997].

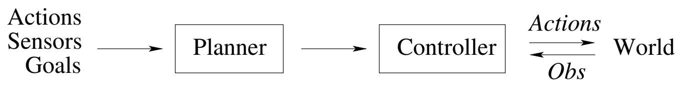

Figure 1. Planning is the model-based approach to autonomous behavior: a planner is a solver that accepts a compact model of the actions, sensors, and goals, and outputs a plan or controller that determines the action to do next given the observations.

Finally, in the *model-based approach*, the control knowledge is derived automatically from a model of the actions, sensors, and goals.

Planning is the model-based approach to autonomous behavior. A planner is a solver that accepts a model of the actions, sensors, and goals, and outputs a plan or controller that determines the action to do next given the observations gathered (Fig. 1). Planners come in a wide variety, depending on the type of model that they target [Ghallab, Nau, and Traverso 2004]. *Classical planners* address deterministic state models with full information about the initial situation, while *conformant planners* address state models with non-deterministic actions and incomplete information about the initial state. In both cases, the resulting plans are open-loop controllers that do not take observations into account. On the other hand, contingent and POMDP planners address scenarios with both uncertainty and feedback, and output genuine closed-loop controllers where the selection of actions depends on the observations gathered.

In all cases, the models are intractable in the worst case, meaning that brute force methods do not scale up to problems involving many actions and variables. Domain-independent approaches aimed at solving these models effectively must thus automatically *recognize* and *exploit* the structure of the individual problems that are given. Like in other AI models such as Constraint Satisfaction Problems and Bayesian Networks [Dechter 2003; Pearl 1988], the key to exploiting the structure of problems in planning models, is *inference*. The most common form of inference in planning is the automatic derivation of heuristic evaluation functions to guide the search. Before considering such domain-independent heuristics, however, we will make precise some of the models used in planning and the languages used for representing them.

2.1 Planning Models

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 thus be described as a state space featuring:

- 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 state $s \in S$,
- a *deterministic state transition function* $f(a,s)$ for $a \in A(s)$ and $s \in S$, and
- positive *action costs* $c(a,s)$ that may depend on the action and the state.

A solution or *plan* 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, and a plan is optimal if it has minimum cost. The cost of a problem is the cost of its optimal solutions. When action costs are all 1, a situation that is common in classical planning, plan cost reduces to plan length, and the optimal plans are simply the shortest ones.

The computation of a classical plan can be cast as a *path-finding* problem in a directed graph whose nodes are the states, and whose source and target nodes are the initial state s_0 and the goal states S_G. Algorithms for solving such problems are polynomial in the number of nodes (states), which is exponential in the number of problem variables (see below). The use of heuristics for guiding the search for plans in large graphs is aimed at improving such worst case behavior.

The model underlying classical planning does not account for either uncertainty or sensing and thus gives rise to plans that represent open-loop controllers where observations play no role. Other planning models in AI take these aspects into account and give rise to different types of controllers.

Conformant planning is planning in the presence of uncertainty in the initial situation and action effects. In the resulting model, the initial state s_0 is replaced by a *set* S_0 of possible initial states, and the deterministic transition function $f(a,s)$ that maps the state s into the unique successor state $s' = f(a,s)$, is replaced by a non-deterministic transition function $F(a,s)$ that maps s into a *set* of possible successor states $s' \in F(a,s)$. A solution to such model, called a conformant plan, is an action sequence that achieves the goal with certainty for *any* possible initial state and *any* possible state transition [Goldman and Boddy 1996]. The search for conformant plans can also be cast as a path-finding problem but over a different, exponentially larger graph whose nodes represent *belief states.* In this formulation, a belief state b stands for the set of states deemed possible, the initial belief state is $b_0 = S_0$, and actions a, whether deterministic or not, map a belief state b into a unique successor belief state b_a, where $s' \in b_a$ if there is a state s in b such that $s' \in F(a,s)$ [Bonet and Geffner 2000].

Planning with sensing, often called contingent planning in AI, refers to planning in the face of both uncertainty and feedback. The model extends the one for conformant planning with a characterization of sensing. A *sensor model* expresses the relation between the observations and the true but possibly hidden states, and can be codified through a set $o \in O$ of observation tokens and a function $o(s)$ that maps states s into observation tokens. An environment is fully observable if different states give rise to different observations, i.e., $o(s) \neq o(s')$ if $s \neq s'$, and partially observable otherwise. While the model for planning with sensing is a slight variation of the model for conformant planning, the resulting solution or plan forms are quite different as observations can and must be taken into account in the selection of actions. Indeed, solution to planning with sensing problems can be expressed equivalently as either *trees* [Weld, Anderson, and Smith 1998], *policies* mapping beliefs into actions [Bonet and Geffner 2000], or *finite-state controllers* [Bonet, Palacios, and Geffner 2009]. A finite-state controller is an automata defined by a collection of tuples of the form $\langle q, o, a, q' \rangle$ that prescribe to do action a and move to the controller state q' after getting the observation o in the controller state q.

The probabilistic versions of these models are also used in planning. The models that result when the actions have stochastic effects and the states are fully observable are the familiar Markov Decision Processes (MDPs) used in Operations Research and Control Theory [Bertsekas 1995], while the models that result when action and sensors are stochastic, are the Partial Observable MDPs (POMDPs) [Kaelbling, Littman, and Cassandra 1998].

2.2 Planning Languages

A domain-independent planner is a general solver over a class of models: classical planners are solvers over the class of basic state models where actions are deterministic and the initial state is fully known, conformant planners are solvers over the class of models where actions are non-deterministic and the initial state is partially known, and so on. In all cases, the corresponding state model that characterizes a given planning problem is not given explicitly but in a compact form, with the states associated with the values of a given set of variables.

One of the most common languages for representing classical problems is Strips, a planning language that can be traced back to the early 70's [Fikes and Nilsson 1971]. A planning problem in Strips is a tuple $P = \langle F, O, I, G \rangle$ where

- F stands for the set of relevant *variables* or *fluents*,
- O stands for the set of relevant *operators* or *actions*,
- $I \subseteq F$ stands for the *initial situation*, and
- $G \subseteq F$ stands for the *goal situation*.

In Strips, 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 in order for the action to be applicable. A Strips problem $P = \langle F, O, I, G \rangle$ encodes in compact form the state model $\mathcal{S}(P)$ where

- the states $s \in S$ are the possible *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 with $Prec(a) \subseteq s$,
- the state transition function is $f(a, s) = (s \setminus Del(a)) \cup Add(a)$, and
- the action costs $c(a)$ are equal to 1 by default.

The states in $S(P)$ represent the possible valuations over the boolean variables in F. Thus, if the set of variables F has cardinality $|F| = n$, the number of states in $S(P)$ is 2^n. A state s represents the valuation where the variables appearing in s are taken to be true, while the variables not appearing in s are false.

As an example, a planning domain that involves three locations l_1, l_2, and l_3, and three tasks t_1, t_2, and t_3, where t_i can be performed only at l_i, can be modeled with a set F of fluents $at(l_i)$ and $done(t_i)$, and a set O of actions $go(l_i, l_j)$ and $do(t_i)$, $i, j = 1, \ldots, 3$, with precondition, add, and delete lists

$$Pre(a) = \{at(l_i)\}\ ,\ Add(a) = \{at(l_j)\}\ ,\ Del(a) = \{at(l_i)\}$$

for $a = go(l_i, l_j)$, and

$$Pre(a) = \{at(l_i)\}\ ,\ Add(a) = \{done(t_i)\}\ ,\ Del(a) = \{\}$$

for $a = do(t_i)$. The problem of doing tasks t_1 and t_2 starting at location l_3 can then be modeled by the tuple $P = \langle F, I, O, G \rangle$ where

$$I = \{at(l_3)\} \ \text{ and } \ G = \{done(t_1), done(t_2)\}\ .$$

A solution to P is an applicable action sequence that maps the state $s_0 = I$ into a state where the goals in G are all true. In this case one such plan is the sequence

$$\pi = \{go(l_3, l_1), do(t_1), go(l_1, l_2), do(t_2)\}\ .$$

The number of states in the problem is 2^6 as there are 6 boolean variables. Still, it can be shown that many of these states are not reachable from the initial state. Indeed, the atoms $at(l_i)$ for $i = 1, 2, 3$ are mutually exclusive and exhaustive, meaning that every state reachable from s_0 makes one and only one of these atoms true. These boolean variables encode indeed the possible values of the multi-valued variable that represents the agent location.

Strips is a planning language based on variables that are boolean, yet planning languages featuring primitive multi-valued variables and richer syntactic constructs are commonly used for describing both classical and non-classical planning models [McDermott 1998; Younes, Littman, Weissman, and Asmuth 2005].

2.3 Planning Algorithms

We have presented some of the models used in domain-independent planning, and one of the languages used for describing them in compact form. We focus now on the algorithms developed for solving them.

GPS, the first AI planner introduced by Newell, Shaw, and Simon, used a technique called means-ends analysis where differences between the current state and the goal situation were identified and mapped into operators that could decrease those differences [Newell and Simon 1963]. Since then, the idea of means-ends analysis has been refined and extended in many ways, seeking planning algorithms that are *sound* (only produce plans), *complete* (produce a plan if one exists), and *effective* (scale up to large problems). By the early 90's, the state-of-the-art method was UCPOP [Penberthy and Weld 1992], an elegant algorithm based on partial-order causal link planning [Sacerdoti 1975; Tate 1977; McAllester and Rosenblitt 1991], a planning method that is sound and complete, but which doesn't scale up too well.

The situation in planning changed drastically in the middle 90's with the introduction of Graphplan [Blum and Furst 1995], a planning algorithm based on the Strips representation but which otherwise had little in common with previous approaches, and scaled up better. Graphplan works iteratively in two phases. In the first phase, Graphplan builds a *plan graph* in polynomial time, made up of a sequence of layers $F_0, A_0, \ldots, F_{n-1}, A_{n-1}, F_n$ where F_i and A_i denote sets of fluents and actions respectively. F_0 is the set of fluents true in the initial situation and n is a planning horizon, initially the index of the first layer F_i where all the goals appear. In this construction, certain pairs of actions and certain pairs of fluents are marked as mutually exclusive or mutex. The meaning of these layers and mutexes is roughly the following: if a fluent p is not in layer F_i, then no plan can achieve p in i steps or less, while if the pair p and q is in F_i but marked as mutex, then no plan can achieve p and q *jointly* in i steps or less. Graphplan makes then an attempt to extract a plan from the graph, a computation that is exponential in the worst case. If the plan extraction fails, the planning horizon n is increased by 1, the plan graph is extended one level, and the plan extraction procedure is tried again. Blum and Furst showed that the planning algorithm is sound, complete, and *optimal*, meaning that the plan obtained minimizes the number of time steps provided that certain sets of actions can be done in parallel. More importantly, they showed *experimentally* that this planning approach scaled up much better than previous approaches.

Due to the new ideas and the emphasis on the empirical evaluation of planning algorithms, Graphplan had a great influence in planning research that has seen two new approaches in recent years that scale up better than Graphplan using methods that are not specific to planning.

In the SAT approach to planning [Kautz and Selman 1996], Strips problems are converted into *satisfiability* problems expressed as a set of clauses (a formula in Conjunctive Normal Form) that are fed into state-of-the-art SAT solvers. If for some horizon n, the clauses are satisfiable, a parallel plan that solves the problem can be read from the model returned by the solver. If not, like in Graphplan, the plan horizon is increased by 1 and the process is repeated until a plan is found. The approach works well when the required horizon is not large and optimal parallel plans are sought.

In the *heuristic search* approach [McDermott 1996; Bonet, Loerincs, and Geffner 1997], the planning problem is solved by heuristic search algorithms with heuristic evaluation functions extracted automatically from the problem encoding. In forward or progression-based planning, the state space $S(P)$ for a problem P is searched for a path connecting the initial state with a goal state. In backward or regression-based planning, plans are searched backwards from the goal. Heuristic search planners have been shown to scale up to very large problems when solutions are not required to be optimal.

The heuristic search approach has actually not only delivered performance but also an explanation for why Graphplan scaled up better than its predecessors. While not described in this form, Graphplan is a heuristic search planner using a heuristic evaluation function encoded implicitly in the planning graph, and a well known admissible search algorithm [Bonet and Geffner 2001]. The difference in performance between recent and older planning algorithms is thus the result of *inference:* while planners searched for plans blindly until Graphplan, they all search with automatically derived heuristics now, or with unit resolution and clause learning when based on the SAT formulation. Domain-independent solvers whose search is not informed by inference of some sort, do not scale up, as there are too many alternatives to choose from, with a few of them leading to the goal.

3 Domain-Independent Planning Heuristics

The main novelty in state-of-the-art planners is the use of automatically derived heuristics to guide the search for plans. In *Heuristics*, Pearl showed how heuristics such as the sum of Manhattan distances for the 15-puzzle, the Euclidian distance for Road Map finding, and the Minimum Spanning Tree for the Travelling Saleman Problem, can all be understood as optimal cost functions of suitable problem *relaxations.* Moreover, for the 15-puzzle, Pearl explicitly considered relaxations obtained mechanically from a Strips representation, showing that both the number of misplaced tiles and the sum of Manhattan distances heuristics are optimal cost functions of relaxations where some *preconditions* of the actions for moving tiles are dropped.

Pearl focused then on the conditions under which a problem relaxation is 'simple enough' so that its optimal cost can be computed in polynomial time. This research problem attracted his attention at the time, and explains his interest on the *graphical structures* underlying various types of problems, including problems of combinatorial optimization, constraint satisfaction, and probabilistic inference. One kind of structure that appeared to result in 'easy' problems in all these contexts was *trees.* Pearl and his students showed indeed that inference on probabilistic Bayesian Trees and Constraint Satisfaction Trees was *polynomial* [Pearl 1982; Dechter and Pearl 1985], even if the general problems are NP-hard (see also [Mackworth and Freuder 1985]). The notion of graphical structures underlying inference problems and the conditions under which they render inference polynomial have been generalized since then in the notion of *treewidth,* a parameter that measures how tree-like is a graph structure [Pearl 1988; Dechter 2003].

Research on the automatic derivation of heuristics in planning builds on Pearl's intuition but takes a different path. The relaxation P^+ that underlies most current heuristics in domain-independent planning is obtained from a Strips problem P by dropping, not the preconditions, but the *delete lists.* This relaxation is quite informative but is not 'easy'; indeed finding an optimal solution to a delete-free problem P^+ is not easier from a complexity point of view than finding an optimal solution to the original problem P. On the other hand, finding *one solution* to P^+, not necessarily optimal, can be done easily, in low polynomial time. The result is that heuristics obtained from P^+ are informative but not *admissible* (they may overestimate the true cost), and hence, they can be used effectively for finding plans but not for finding *optimal* plans.

If $P(s)$ refers to a planning problem that is like $P = \langle F, I, O, G\rangle$ but with $I = s$, and $\pi(s)$ is the solution found for the delete-relaxation $P^+(s)$, the heuristic $h(s)$ that estimates the cost of the problem $P(s)$ is defined as

$$h(s) = Cost(\pi(s)) = \sum_{a \in \pi(s)} cost(a) \ .$$

The plans $\pi(s)$ for the relaxation $P^+(s)$ are called *relaxed plans*, and there have been many proposals for defining and computing them. We explain below one such method that corresponds to running Graphplan on the delete-relaxation $P^+(s)$ [Hoffmann and Nebel 2001]. In delete-free problems, Graphplan runs in polynomial time and its plan graph construction is simplified as there are no mutex relations to keep track of.

The layers F_0, A_0, F_1, $\ldots$, F_{n-1}, A_{n-1}, F_n in the plan graph for $P^+(s)$ are computed starting with $F_0 = s$, by placing in A_i, $i = 1, \ldots, n-1$, all the actions a in P whose preconditions $Pre(a)$ are in F_i, and placing in F_{i+1}, the add effects of those actions along with the fluents in F_i. This construction is terminated when the goals G are all in F_n, or when $F_n = F_{n+1}$. Then if $G \not\subseteq F_n$, $h(s) = \infty$, as it can be shown then that the relaxed problem $P^+(s)$ and the original problem $P(s)$ have no solution. Otherwise, a (relaxed) parallel plan $\pi(s)$ for $P^+(s)$ can be obtained backwards from the layer F_n by collecting the actions that add the goals, and recursively, the actions that add the preconditions of those actions that are not true in the state s.

More precisely, for $G_n = G$ and i from $n-1$ to 0, B_i is set to a minimal collection of actions in A_i that add all the atoms in $G_{i+1} \setminus F_i$, and G_i is set to $Pre(B_i) \cup (G_{i+1} \cap F_i)$ where $Pre(B_i)$ is the collection of fluents that are preconditions of actions in B_i. It can be shown then that $\pi(s) = B_0, \ldots, B_{n-1}$ is a parallel plan for the relaxation $P^+(s)$; the plan being parallel because the actions in each set B_i are assumed to be done in parallel. The heuristic $h(s)$ is then just $Cost(\pi(s))$. This

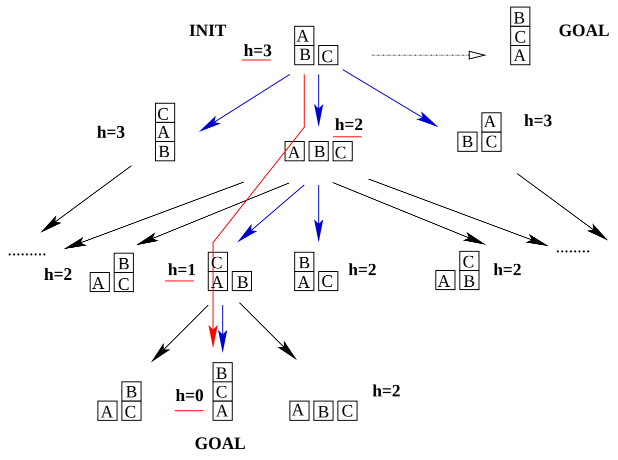

Figure 2. A simple planning problem involving three blocks with initial and goal situations I and G as shown. The actions allow to move a clear block on top of another clear block or to the table. A plan for the problem is a path that connects I with G in the directed graph partially shown. In this example, the plan can be found greedily by taking in each state s, starting with $s = I$, the action that results in a state s' that is closer to the goal according to the heuristic. The heuristic values (shown) are derived automatically from the problem as described in the text.

is indeed the heuristic introduced in the FF planner [Hoffmann and Nebel 2001], which is suitable when action costs are uniform. For non-uniform action costs, other heuristics are more convenient [Keyder and Geffner 2008].

4 Meaning of Domain-Independent Heuristics

In order to illustrate the meaning and derivation of domain-independent heuristics, let us consider the example shown in Fig. 2, where blocks a, b, and c initially arranged so that a is on b, and b and c are on the table, must be rearranged so that b is on c, and c is on a. The actions allow to move a clear block (a block with no block on top) on top of another clear block or to the table. The problem can be expressed as a Strips problem $P = \langle F, I, O, G \rangle$ with a set of atoms F given by $on(x, y)$, $ontable(x)$, and $clear(x)$, where x and y range over the block labels a, b, and c. In the heuristic search approach to planning, the solution to P becomes a path-finding problem in the directed graph associated with the state model $S(P)$, where the nodes stand for the states in $S(P)$, and the actions $a \in O$ are mapped into edges connecting a state s with a state s' when a is applicable in s and maps s into s'.

The Blocks World is simple for people, but until recently, not so simple for *domain-independent* planners. Indeed, the size of the graph to search is exponential in the number of blocks n, with $n!$ possible towers of n blocks, and additional combinations of shorter towers.

Figure 2 shows the search that results from a planner using the heuristic described above, whose value $h(s)$ for each of the states in the graph is shown. All action costs are assumed to be 1. With the heuristic shown, the solution to the problem can be found with no search at all by just selecting in each state s the action that leads to the state s' that is closest to the goal (lowest heuristic value). In the initial state, this action is the one that places block a on the table, in the following state, the action that places c on a, and so on.

In order to understand the numbers shown in the figure, let us see how the value $h(s) = 3$ for the initial state s is derived. The heuristic $h(s)$ is $|\pi(s)|$ where $\pi(s)$ is the plan found for the relaxation $P^+(s)$. The relaxed plan $\pi(s)$ is obtained by constructing first the layered graph F_0, A_0, $\ldots$, F_{n-1}, A_{n-1}, F_n, where $n > 0$ as none of the goals $on(b, c)$ and $on(c, a)$ are in $F_0 = s$. The actions in A_0 are the actions applicable given the atoms in F_0, i.e., the actions a with $Pre(a) \subseteq F_0$. This set includes the actions of moving c to a, a to c, and a to the table, but does not include actions that move b as the precondition $clear(b)$ is not part of F_0. The set F_1 extends F_0 with all the atoms added by the actions in A_0, and includes $on(c, a)$, $on(a, c)$, $ontable(a)$, and $clear(b)$, but not the goal $on(b, c)$. Yet with $clear(b)$ and $clear(c)$ in F_1, the action for moving b to c appears in layer A_1, and therefore, the other goal atom $on(b, c)$ appears in F_2. By collecting the actions that first add the goal atoms $on(c, a)$ and $on(b, c)$, and recursively, the preconditions of those actions that are not in s, a relaxed plan $\pi(s)$ with 3 actions is obtained so that $h(s) = 3$. There are several choices for the actions in $\pi(s)$ that result from the way ties in the plan extraction procedure are broken. One possible relaxed plan involves moving a to the table and c to a in the first step, and b to c in the second step. Another involves moving a to c and c to a first, and then b to c.

It is important to notice that fluent layers such as F_1 in the plan graph do not represent any 'real' states in the original problem P as they include atoms pairs like $on(a, c)$ and $on(c, a)$ that cannot be achieved jointly in any state s' reachable from the initial state. The layer F_1 is instead an *abstraction* that *approximates* the set of all states reachable in one step from the initial state by taking their union. This approximation implies that finding an atom p in a layer F_n with $n > 1$ is no guarantee that there is a real plan for p in $P(s)$ that achieves p in n time steps, rather than one such parallel plan exists in the relaxation. Similarly, the relaxed plans $\pi(s)$ obtained above are quite 'meaningless'; they move a to the table or to c at the same time that they move c to a. Yet, these 'meaningless' relaxed plans $\pi(s)$ yield the heuristic values $h(s)$ that provide the search with a very meaningful and effective sense of direction.

Let us finally point out that the computation of the domain-independent heuristic $h(s)$ results in valuable information that goes beyond the *numbers* $h(s)$. Indeed, from the computation of the heuristic value $h(s)$, it is possible to determine the actions applicable in the state s that are most relevant to the goal, and then focus on the evaluation of the states that result from those actions only. This type of *action pruning* has been shown to be quite effective [Hoffmann and Nebel 2001], and in slightly different form is part of state-of-the-art planners [Richter, Helmert, and Westphal 2008].

5 Other Developments in Planning

Domain-independent planning is concerned with non-classical models also where information about the initial situation is incomplete, actions may have non-deterministic effects, and states may be fully or partially observable. A number of *native solvers* for such models, that include Markov Decision Processes (MDPs) and Partially Observable MDPs have been developed, and progress in the area has been considerable too. Moreover, many of these solvers are also based on *heuristic search* methods (see the article by Bonet and Hansen in this volume). I will not review this literature here but focus instead on two ways in which the results obtained for *classical planning* are relevant to such richer settings too.

First, it's often possible to plan under *uncertainty* without having to model the uncertainty explicitly. This is well known by control engineers that normally design closed-loop controllers for stochastic systems ignoring the 'noise'. Indeed, the error in the model is compensated by the feedback loop. In planning, where non-linear models are considered, the same simplification works too. For instance, in a Blocks World where the action of moving a block may fail, an effective closed-loop policy can be obtained by replanning from the current state when things didn't progress as predicted by the simplified model. Indeed, the planner that did best in the first probabilistic planning competition [Younes, Littman, Weissman, and Asmuth 2005], was not an MDP planner, but a classical replanner of this type. Of course, this approach is not suitable when it may be hard or impossible to recover from failures, or when the system state is not fully observable. In everyday planning, however, such cases may be the exception.

Second, it has been recently shown that it's often possible to efficiently transform problems featuring uncertainty and sensing into classical planning problems that do not. For example, problems P involving uncertainty in the initial situation and no sensing, namely conformant planning problems, can be compiled into classical problems $K(P)$ by adding new actions and fluents that express conditionals [Palacios and Geffner 2007]. The translation from the conformant problem P into the classical problem $K(P)$ is sound and complete, and provided that a *width* parameter defined over P is bounded, it is polynomial too. The result is that the conformant plans for P can be read from the plans for $K(P)$ that can be computed using a classical planner. Moreover, this technique has been recently used for deriving *finite-state controllers* that solve problems featuring both incomplete information and sensing [Bonet, Palacios, and Geffner 2009]. A finite-state controller is an automata that given the current (controller) state and the current observation selects an action and updates the controller state, and so on, until reaching the goal. Figure 3 shows one such problem (left) and the resulting controller (right). The problem, motivated by the work on deictic representations in the selection of actions [Chapman 1989; Ballard, Hayhoe, Pook, and Rao 1997], is about placing a visual marker on top of a green block in a blocks-world scene where the location of the green blocks is not known. The visual marker, initially at the lower left corner of the scene (shown as an eye), can be moved in the four directions, one cell at a time. The observations are whether the cell beneath the marker is empty ('C'), a non-green block ('B'), or a green block ('G'), and whether it is on the table ('T') or not ('-'). The controller shown on the right has been derived by running a classical planner over a classical problem obtained by an automatic translation from the original problem that involves both uncertainty and sensing. In the figure, the controller states q_i are shown in circles while the label o/a on an edge connecting two states q to q' means to do action a when observing o in q and then switching to q'. In the classical planning problem obtained from the translation, the actions are tuples $(f_q, f_o, a, f_{q'})$ whose effects are those of the action a but conditional on the fluents f_q and f_o representing the controller state q and observation o being true. In such a case, the fluent $f_{q'}$ representing the controller state q' is made true and f_q is made false. The two appealing features of this formulation is that the resulting classical plans encode very succint closed-loop controllers, and that these controllers are quite general. Indeed, the controller shown in the figure not only solves the problem for the configuration of blocks shown, but for *any configuration* involving *any number of blocks.* The controller prescribes to move the 'eye' up until there are no blocks, then to move it down until reaching the table and right, and to repeat this process until a green block is found ('G'). Likewise, the 'eye' must move right when there are no blocks in a given spot (both 'T' and 'C' observed). See [Bonet, Palacios, and Geffner 2009] for details.

6 Heuristics and Cognition

Heuristic evaluation functions are used also in other settings such as Chess playing programs [Pearl 1983] and reinforcement learning [Sutton and Barto 1998]. The difference between evaluations functions in Chess, reinforcement learning, and domain-independent planning mimic actually quite closely the relation among the three approaches to action selection mentioned in the introduction: programming-based, learning-based, and model-based. Indeed, the evaluation functions are programmed by hand in Chess, are learned by trial-and-error in reinforcement learning, and are derived from a (relaxed) model in domain-independent planning.

Heuristic evaluation functions in reinforcement learning, called simply valuation functions, are computed by stochastic sampling and dynamic programming updates. This is a *model-free method* that has been shown to be effective in low-level tasks that do not involve large state spaces, and which provides an accurate account of learning in the brain [Schultz, Dayan, and Montague 1997].

Heuristic evaluation functions as used in domain-independent planning are com-

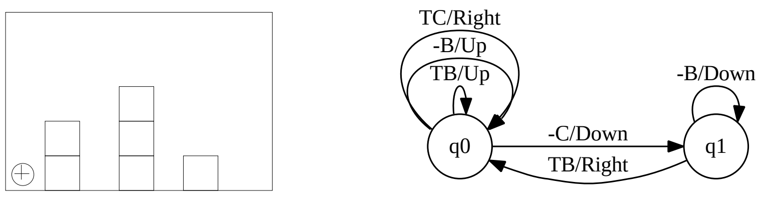

Figure 3. *Left:* The visual marker shown as an 'eye' must be placed on a green block in the blocks-world scene shown, where the locations of the green blocks are not known. The visual marker can be moved in the four directions, one cell at a time. The observations are whether the cell beneath the marker is empty ('C'), a non-green block ('B'), or a green block ('G'), and whether the marker is on the table ('T') or not ('-'). *Right:* The controller derived for this problem using a classical planner over a suitable automatic transformation. The controller states q_i are shown in circles while the label o/a on an edge connecting q to q' means to do a when observing o in q switching then to q'. The controller works not only for the problem instance shown on the left, but for *any* instance resulting from changes in the configuration or in the number of blocks.

puted by *model-based methods* where suitable relaxations are solved from scratch. The technique has been shown to work over large problems involving hundred of actions and fluents. Here I want to argue these methods also have features that make them interesting from a cognitive point of view as a plausible basis for an account of 'feelings', 'emotions', or 'appraisals' in high-level human problem solving. I focus on three of these features.

First, domain-independent heuristics are fast (low-polynomial time) and effective, as the 'fast and frugal' heuristics advocated by Gigerenzer and others [Gigerenzer and Todd 1999; Gigerenzer 2007], and yet, they are general too: they apply indeed to all the problems that fit the (classical planning) model and to problems that can be cast in that form (like the visual-marker problem above).

Second, the derivation of these heuristics sheds light on why appraisals may be opaque from a cognitive point of view, and thus not conscious. This is because the heuristic values are obtained from a relaxed model where the meaning of the symbols is different than the meaning of the symbols in the 'true' model. For example, the action of moving an object from one place to another, deletes the old place in the true model but not in the delete-relaxation where an object can thus appear in multiple places at the same time. Thus, if the agent selecting the actions with the resulting heuristic does not have access to the relaxation, it won't be able to explain how the heuristic evaluations are produced nor what they stand for. The importance of the unconscious in everyday cognition is a topic that has been receiving increased attention in recent years, with conscious, deliberate reasoning, appearing to rely heavily on unconscious processing and representing just the tip of the 'cognitive iceberg' [Wilson 2002; Hassin, Uleman, and Bargh 2005; Evans

2008]. While this is evident in vision and natural language processing, where it is clear that one does not have access to how one 'sees' or 'understands', this is likely to be true in most cognitive tasks, including apparently simple problems such as the Blocks World where our ability to find reasons for the actions selected, does not explain how such actions are selected in the first place. In this sense, the focus of cognitive psychology on puzzles such as the Tower of Hanoi may be misplaced: simple problems, such as the Blocks World, are not simple for *domain-independent solvers,* and there is no question that people are capable of solving domains that they have never seen where the combinatorics would defy a naive, blind solver.

Third, the heuristics provide the agent with a sense of direction or 'gut feelings' that guide the action selection in the presence of many alternatives, while avoiding an infinite regress in the decision process. Indeed, emotions long held to interfere with the decision process and rationality, are now widely perceived as a requisite in contexts where it is not possible to consider all alternatives. Emotions and gut feelings are thus perceived as the 'invisible hand' that successfully guides us out of these mental labyrinths [Ketelaar and Todd 2001; Evans 2002].[1] The 'rationality of the emotions' have been defended on theoretical grounds by philosophers [De Sousa 1990; Elster 1999], and on empirical grounds by neuroscientists that have studied the impairments in the decision process that result from lesions in the frontal lobes [Damasio 1995]. The link between emotions and evaluation functions, point to their computational role as well.

While emotions are currently thought as providing the appraisals that are necessary for navigating in a complex world, there are actually very few accounts of *how such appraisals may be computed.* Reinforcement learning methods provide one such account that works well in low level tasks without requiring a model. Heuristic planning methods provide another account that works well in high-level tasks where the model is known. Moreover, as discussed above, heuristic planning methods do not only provide an account of the appraisals, but also of the actions that are worth evaluating. These are the actions a in the state s that are deemed relevant to the goal in the computation of the heuristic $h(s)$; the so-called helpful actions [Hoffmann and Nebel 2001]. This form of *action pruning* may account for a key difference between programs and humans in games such as Chess: while the former consider all possible moves and responses (up to a certain depth), the latter focus on the analysis and evaluation of a few moves and countermoves. Domain-independent heuristics can account in principle for both the focus and the evaluation, the latter in the *value* of the heuristic function $h(s)$, the former in its *structure.*

[1]Some philosophers and cognitive scientists refer to this combinatorial problem as the 'frame problem' in AI. This terminology, however, is not accurate. The frame problem in AI [McCarthy and Hayes 1969] refers to the problem that arises in logical accounts of actions and change where the description of the action effects does not suffice to capture what does not change. E.g., the number of chairs in the room does not change if the bell rings. The frame problem is the problem of capturing what does not change from a concise logical description of what changes [Ford and Pylyshyn 1996].

7 AI and Cognitive Science: Past and Future

Pearl's ideas on the mechanical discovery of heuristics has received renewed attention in the area of domain-independent planning where heuristic evaluation functions, derived automatically from the problem encoding, are used to guide the search for plans in large spaces. Heuristic search planners are powerful domain-independent solvers that have been *empirically tested* over many large domains involving hundred of actions and variables.

The developments in planning parallel those in other areas of AI and bear on the relevance of Artificial Intelligence to the understanding of the human mind. AI and Cognitive Science were twin disciplines until the 80's, with AI looking to the human mind for inspiration, and Cognitive Science looking to computation as a language for modeling. The relationship between AI and Cognitive Science has changed, however, and the two disciplines do not appear to be that close now. Below, I go over some of the relevant changes that explain this divorce, and explain why, in spite to them, AI remains and will likely remain critically relevant for understanding the human mind, a premise that underlies and motivates the work of Judea Pearl and others AI scientists.

A lot of work in AI until the 80's was about writing programs capable of displaying intelligence over ill-defined problems, either by appealing to introspection or by interviewing an expert. Many good ideas came out from this work, yet few have had a lasting scientific value. The methodological problem with the 'knowledge-based' approach in AI was that the resulting programs were not robust and they always appeared to be missing critical knowledge; either declarative (e.g., that men don't get pregnant), procedural (e.g., which rule or action to apply next), or both. This situation led to an impasse in the 80's, and to many debates and criticisms, like that 'good old fashioned AI' is 'rule application' but human intelligence is not [Haugeland 1993], that representation is not needed for intelligent behavior and gets in the way [Brooks 1991], that subsymbolic neural networks and genetic algorithms are the way to go [Rumelhart and McClelland 1986; Holland 1992], etc.

In part due to the perceived limitations of the knowledge-based approach and the criticisms, and in part due to its own evolution, mainstream AI has changed substantially since the 80's. One of the key methodological changes is that many researchers have moved from the early paradigm of *writing programs for ill-defined problems* to *writing solvers for well-defined mathematical models.* These models include Constraint Satisfaction Problems, Strips Planning, Bayesian Networks and Partially Observable Markov Decision Processes, among others. Solvers are programs that take a compact description of a particular model instance (a planning problem, a CSP instance, and so on) and automatically compute its solution. Unlike the early AI programs, solvers are *general* as they must deal with any problem that fits the model (any instance). Moreover, some of these models, like POMDPs, are extremely expressive. The challenge in this research agenda is mainly *com-*

putational: how to make these domain-independent solvers scale up to large and interesting problems given that all these models are intractable in the worst case. Work in these areas has uncovered techniques that accomplish this by automatically recognizing and exploiting the structure of the problems at hand. In planning, these techniques have to do with the automatic derivation and use of heuristic evaluation functions; in SAT and CSPs, with constraint propagation and learning, while in CSPs and Bayesian Networks, with the use of the underlying graphical structure.

The relevance of the early work in AI to Cognitive Science was based on *intuition:* programs provided a way for specifying intuitions precisely and for trying them out. The more recent work on *domain-independent solvers* is more technical and experimental, and is focused not on reproducing intuitions but on *scalability.* This may give the impression, confirmed by the current literature, that recent work in AI is less relevant to Cognitive Science than work in the past. This impression, however, may prove wrong on at least two grounds. First, intuition is not what it used to be, and it is now regarded as the tip of an iceberg whose bulk is made of massive amounts of shallow, fast, but unconscious inference mechanisms that cannot be rendered explicit in the form of rules [Wilson 2002; Hassin, Uleman, and Bargh 2005; Gigerenzer 2007]. Second, whatever these mechanisms are, they appear to work pretty well and to scale up. This is no small feat, given that most methods, whether intuitive or not, do not. Indeed, if the techniques that really scale up are not that many, a plausible conjecture at this point, it may well be the case that *the twin goals of accounting reliably for the intuitions and of scaling up* have a large overlap. By focusing then on the study of meaningful models and the computational methods for dealing with them *effectively,* AI may prove its relevance to human cognition in ways that may go well beyond the rules, cognitive architectures, and knowledge structures of the 80's. Human Cognition, indeed, still provides the inspiration and motivation for a lot of research in AI. The use of Bayesian Networks in Development Psychology for understanding how children acquire and use causal relations [Gopnik, Glymour, Sobel, Schulz, , Kushnir, and Danks 2004], and the use of Reinforcement Learning algorithms in Neuroscience for interpreting the activity of dopamine cells in the brain [Schultz, Dayan, and Montague 1997], are two examples of general AI techniques that have made it recently into Cognitive Science. As AI focuses on models and solvers able to scale up, more techniques are likely to follow. One such candidate is the automatic derivation of heuristic functions as used in planning, which like the research on Bayesian Networks, owes a lot to the seminal work of Judea Pearl.

References

Ballard, D., M. Hayhoe, P. Pook, and R. Rao (1997). Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences 20*(4), 723–742.

Bertsekas, D. (1995). *Dynamic Programming and Optimal Control, Vols 1 and 2.*

Athena Scientific.

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

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

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

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

Bonet, B., H. Palacios, and H. Geffner (2009). Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-09).*

Brooks, R. (1987). A robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation 2*, 14–27.

Brooks, R. (1991). Intelligence without representation. *Artificial Intelligence 47*(1–2), 139–159.

Chapman, D. (1989). Penguins can make cake. *AI magazine 10*(4), 45–50.

Damasio, A. (1995). *Descartes' Error: Emotion, Reason, and the Human Brain.* Quill.

De Sousa, R. (1990). *The rationality of emotion.* MIT Press.

Dechter, R. (2003). *Constraint Processing.* Morgan Kaufmann.

Dechter, R. and J. Pearl (1985). The anatomy of easy problems: a constraint-satisfaction formulation. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI-85)*, pp. 1066–1072.

Elster, J. (1999). *Alchemies of the Mind: Rationality and the Emotions.* Cambridge University Press.

Evans, D. (2002). The search hypothesis of emotion. *British J. Phil. Science 53*, 497–509.

Evans, J. (2008). Dual-processing accounts of reasoning, judgment, and social cognition. *Annual Review of Pschycology 59*, 255–258.

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

Ford, K. and Z. Pylyshyn (1996). *The robot's dilemma revisited: the frame problem in artificial intelligence.* Ablex Publishing.

Ghallab, M., D. Nau, and P. Traverso (2004). *Automated Planning: theory and practice.* Morgan Kaufmann.

Gigerenzer, G. (2007). *Gut feelings: The intelligence of the unconscious.* Viking Books.

Gigerenzer, G. and P. Todd (1999). *Simple Heuristics that Make Us Smart.* Oxford University Press.

Goldman, R. P. and M. S. Boddy (1996). Expressive planning and explicit knowledge. In *Proc. AIPS-1996.*

Gopnik, A., C. Glymour, D. Sobel, L. Schulz, , T. Kushnir, and D. Danks (2004). A theory of causal learning in children: Causal maps and Bayes nets. *Psychological Review 111*(1), 3–31.

Hassin, R., J. Uleman, and J. Bargh (2005). *The new unconscious.* Oxford University Press, USA.

Haugeland, J. (1993). *Artificial intelligence: The very idea.* MIT press.

Hoffmann, J. and B. Nebel (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research 14*, 253–302.

Holland, J. (1992). *Adaptation in natural and artificial systems.* MIT Press.

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

Kautz, H. and B. Selman (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI*, pp. 1194–1201.

Ketelaar, T. and P. M. Todd (2001). Framing our thoughts: Evolutionary psychology's answer to the computational mind's dilemma. In H. Holcomb (Ed.), *Conceptual Challenges in Evolutionary Psychology.* Kluwer.

Keyder, E. and H. Geffner (2008). Heuristics for planning with action costs revisited. In *Proc. ECAI-08*, pp. 588–592.

Mackworth, A. and E. C. Freuder (1985). The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence 25*(1), 65–74.

Mataric, M. J. (2007). *The Robotics Primer.* MIT Press.

McAllester, D. and D. Rosenblitt (1991). Systematic nonlinear planning. In *Proceedings of AAAI-91*, Anaheim, CA, pp. 634–639. AAAI Press.

McCarthy, J. and P. Hayes (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pp. 463–502. Edinburgh University 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. (1998). PDDL – the planning domain definition language. At `http://ftp.cs.yale.edu/pub/mcdermott`.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Newell, A., J. Shaw, and H. Simon (1958). Elements of a theory of human problem solving. *Psychology Review 23*, 342–343.

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

Palacios, H. and H. Geffner (2007). From conformant into classical planning: Efficient translations that may be complete too. In *Proc. 17th Int. Conf. on Planning and Scheduling (ICAPS-07)*.

Pearl, J. (1982). Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, pp. 133–136.

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

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

Penberthy, J. and D. Weld (1992). UCPOP: A sound, complete, partially order planner for ADL. In *Proceedings KR'92*.

Richter, S., M. Helmert, and M. Westphal (2008). Landmarks revisited. In *Proc. AAAI*, pp. 975–982.

Rumelhart, D. and J. McClelland (Eds.) (1986). *Parallel distributed processing: explorations in the microstructure of cognition. Vol. 1*. MIT Press.

Sacerdoti, E. (1975). The nonlinear nature of plans. In *Proceedings of IJCAI-75*, Tbilisi, Georgia, pp. 206–214.

Schultz, W., P. Dayan, and P. Montague (1997). A neural substrate of prediction and reward. *Science 275*(5306), 1593–1599.

Sutton, R. and A. Barto (1998). *Introduction to Reinforcement Learning*. MIT Press.

Tate, A. (1977). Generating project networks. In *Proc. IJCAI*, pp. 888–893.

Weld, D., C. Anderson, and D. Smith (1998). Extending Graphplan to handle uncertainty and sensing actions. In *Proc. AAAI-98*, pp. 897–904. AAAI Press.

Wilson, T. (2002). *Strangers to ourselves*. Belknap Press.

Younes, H., M. Littman, D. Weissman, and J. Asmuth (2005). The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research 24*, 851–887.