In Wiley Interdisciplinary Reviews: Cognitive Science (4) 4, 2013

Computational Models of Planning (Overview Article)

Hector Geffner
ICREA & Universitat Pompeu Fabra
Roc Boronat 138, 55.213
08018 Barcelona, SPAIN

Abstract

The selection of the action to do next is one of the central problems faced by autonomous agents. Natural and artificial systems address this problem in various ways: action responses can be hardwired, they can be learned, or they can be computed from a model of the situation, the actions, and the goals. Planning is the model-based approach to action selection and a fundamental ingredient of intelligent behavior in both humans and machines. Planning, however, is computationally hard as the consideration of all possible courses of action is not computationally feasible. The problem has been addressed by research in Artificial Intelligence that in recent years has uncovered simple but powerful computational principles that make planning feasible. The principles take the form of domain-independent methods for computing heuristics or appraisals that enable the effective generation of goal-directed behavior even over huge spaces. In this paper, we look at several planning models, at methods that have been shown to scale up to large problems, and at what these methods may suggest about the human mind.

1 Introduction

Planning is one of the key aspects of intelligent behavior, and one of the first to be studied in the field of Artificial Intelligence (AI). The first AI planner and one of the first AI programs was the General Problem Solver (GPS) developed by Newell and Simon in the late 50's [80, 81]. Since then, planning has remained a central topic in AI while changing in significant ways: on the one hand, it has become more rigorous, with a variety of planning models defined and studied; on the other, it has become more empirical, with planning algorithms evaluated experimentally and a strong emphasis placed on scalability: effective goal-oriented behavior that is not limited by the number of actions and variables in the problem.

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. In the *hardwired approach*, action responses are hardwired by nature, or more commonly in AI systems, by a programmer. For a robot moving in an office environment, for example, 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, and so on [25, 71]. The problem with this approach, common to many other AI systems, is that since it is difficult to anticipate all possible situations and the best way to handle them, the resulting systems tend to be brittle. In the *learning-based approach*, on the other hand, action responses are not hardwired but are learned by trial and error as in reinforcement learning [98], or by generalization from examples as in supervised learning [78, 89]. Finally, in the *model-based approach*, action responses are computed from a model of the situation, the actions, the sensors, and goals [89, 39, 37]. The distinction that the philosopher Daniel Dennett makes between 'Darwinian', 'Skinnerian', and 'Popperian' creatures [32], mirrors quite closely the distinction between hardwired agents, agents that learn, and agents that use models respectively. The approaches, however, are not incompatible, and indeed, agents that learn models rather than behavioral responses fit also in the latter class, as they must then use the models learned for selecting the action to do next.

Planning, usually associated with "thinking before acting", is best conceived as the model-based approach to action selection. It is a fundamental ingredient of intelligent behavior in both humans and machines, yet it is computationally hard. Indeed, even in the most simple planning model where the state of the world is fully known and actions have deterministic effects, the number of possible world states is exponential in the number of state variables, and the number of possible plans is exponential in the length of the plans.[1] The situation facing an artificial agent that plans even in the simplest setting, has a key element that is common with humans making plans: neither one can afford to consider all possible courses of action as this is not computationally feasible. This computational challenge has been used as evidence for contesting the possibility of general planning and reasoning abilities in humans or machines [100]. The complexity of planning, however, just implies that no planner can efficiently solve every problem from every domain, not that a planner cannot solve an infinite collection of problems from old and new domains, and hence be useful to an acting agent. This is indeed the way modern AI planners are empirically evaluated and ranked in AI planning competitions, where they are tried on domains that the planners' authors have never seen [29]. Thus, far from representing an insurmountable obstacle, the twin requirements of generality and scalability have been addressed head on in AI planning research, and this has led to simple but powerful computational principles that make domain-independent planning feasible. The principles take the form of heuristics or appraisals that are computed automatically for the problem at hand from suitable relaxations (simplifications) of the problem. The heuristics derived in this way tailor the general solver to the specific problem, and enable the generation of goal-directed behavior even over huge spaces. In this paper, we look at these methods and what they may reveal about the human mind, turning the requirements of generality and scalability from an impossibility into a critical source of insight.

From a cognitive perspective, one lesson to draw is that if one of the primary roles of feelings and emotions is to act as rough guides in the generation of behavior in complex physical and social worlds where the consideration of all possible courses of action is not

[1]More precisely, in the language of computer science, planning is said to be NP-hard, meaning that there cannot be a general sound and complete planning algorithm that runs in polynomial time, unless the fundamental conjecture in computer science that NP-hard problems do not admit polynomial solutions, widely believed to be true, turns out to be false [26].

computationally feasible, much is much to be gained by looking at suitable abstractions of the problem that present the same computational challenges in crisp form. More specifically, we will argue that the domain-independent methods for deriving heuristic functions that have been developed for making planning feasible provide a different angle from which to look at the computation and role of emotional appraisals in humans [36].[2]

The paper is organized as follows. We consider the most basic planning model first, the computational challenges that it presents, and the domain-independent methods that have been developed for addressing this challenge. We then look at what these domain-independent heuristics suggest about the generation, nature, and role of unconscious appraisals, and move on to alternative planning methods and richer planning models. We conclude by discussing some open challenges in planning research and summarizing the main lessons.

2 Basic Model: Classical Planning

The most basic model in planning is concerned with the selection of actions for achieving goals when the initial situation is fully known and actions have deterministic effects. The model underlying this form of planning, called *classical planning*, can be described formally in terms of 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,
- a set of 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 $s \in S$, so that $f(a, s)$ denotes the state that results from doing action a in state s, and
- positive *action costs* $c(a, s)$ that may depend on the action a and the state s.

A solution or *plan* for 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 each action 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; i.e., $s_{i+1} = f(a_i, s_i)$ for $a_i \in A(s_i)$, $i = 0, \ldots, n$, and $s_{n+1} \in S_G$. The cost of a plan is the sum of the action costs $c(a_i, s_i)$, 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 one, plan cost reduces to plan length, and the optimal plans are simply the shortest ones.

As an illustration, the problem of rearranging a set of block towers into a different set of towers can be naturally formulated as a model of this type where the state represents the possible block configurations. Similarly, delivering a set of goods to clients, solving a

[2]The relationship between planning and emotion has been considered by other authors that aim to import elements from the theory of emotions into AI planners [43, 44]. Our focus here goes in the opposite direction, on what modern computational models of planning can tell us about the generation, nature, and role of emotional appraisals.

puzzle such as Rubik's Cube, or playing solitaire can be all expressed as classical planning problems.

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. Planning models that take these aspects into account give rise to different types of controllers and will be considered later.

3 State Variables and Factored Representations

The state models required for planning are not represented explicitly in general because they are often too large. The planning agent is assumed to have instead a compact and implicit representation of the state model given in terms of a set of state variables. The states are the possible combination of values over these variables, and the actions are defined in terms of pre and postconditions expressed over the state variables as well.

One of the most common languages for representing classical planning problems is also one of the oldest, Strips [35], a planning language that can be traced back to the late 60's. A planning problem in Strips is a tuple $P = \langle F, O, I, G \rangle$ where

- F represents a set of *boolean variables*,
- O represents a set of *actions*,
- I represents the *initial situation*, and
- G represents the *goal*.

Both the initial situation I and the goal G are expressed by a set of atoms over F. For a boolean state variable p in F, there are two atoms $p = true$ and $p = false$, abbreviated using logical notation as p and $\neg p$, where the symbol '$\neg$' stands for negation. A state in a Strips problem is a valuation of the state variables represented by the collection of atoms that the state makes true. The initial situation I stands for the state that makes the atoms in I true and all other atoms false, while the goal G stands for the collection of states that make all the atoms in G true.

In Strips, the actions $o \in O$ are represented by three sets of atoms over 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 implicitly, in compact form, the classical state model $S(P)$ where

- the states $s \in S$ are the possible *collections of atoms* over 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 $f(a, s) = (s \setminus Del(a)) \cup Add(a)$, i.e., the state s with the atoms in $Del(a)$ deleted and the atoms in $Add(a)$ added, and
- the action costs $c(a)$ are equal to one by default.

Given that the Strips problem represents the state model $S(P)$, the *plans* for P are defined as the plans for $S(P)$; namely, the action sequences that map the initial state s_0 that corresponds to I into a goal state where the goals G are true. Since the states in $S(P)$ are represented as collections of atoms from F, the number of states in $S(P)$ is $2^{|F|}$ where $|F|$ is the number of boolean variables F in P, usually called *fluents*.

The state representation that follows from a planning language such as Strips is domain-independent. Thus, while a specialized solver for the Blocks World may represent the state by a set of lists, each one representing a tower of blocks, in the state representation that follows from Strips there will be just atoms, and the same will be true of any other domain. As an illustration, a 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 location 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 collectively 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. Planning languages featuring non-boolean variables and richer syntactic constructs than Strips are also common in planning [75, 6, 106].

4 Complexity

The problem of finding a plan for the problem P can be naturally cast as a path-finding problem over an implicit directed graph $G(P)$ whose nodes stand for the states in the model represented by P, and whose edges $s \rightarrow s'$ represent the presence of an action that maps the state s into the state s'. Paths connecting the state s_0, representing the initial situation of P, to a state s where the goals of P are true, capture the plans that solve P. Algorithms for computing such paths are well-known in computer science and include Dijkstra's shortest-path method that iteratively extends the set of nodes whose shortest path distance $g(s)$ from

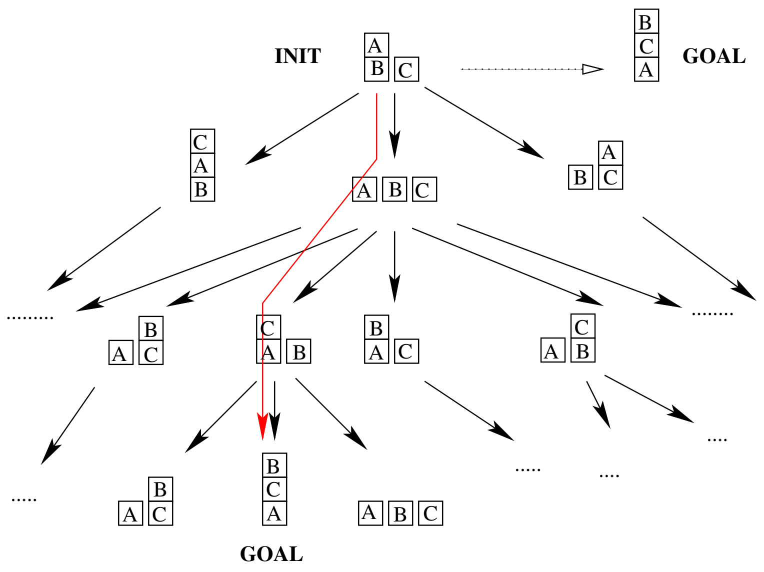

Figure 1: The graph corresponding to a simple planning problem involving three blocks with initial and goal situations I and G as shown. The actions allow moving a clear block on top of another clear block or to the table. The size of the complete graph for this domain is exponential in the number of blocks. A plan for the problem is shown by the path in red.

the source node is labeled as known, starting with the set that comprises only the source node s_0 [33, 30]. Then, in each iteration, Dijkstra's algorithm selects among the nodes s whose shortest path distance from s_0 is not yet known, the one that minimizes the distance $g(s) = \min_{s'} g(s') + c(s', s)$, where s' ranges over the nodes that can be mapped into the state s by an action with cost $c(s', s)$. The algorithm then labels the distance $g(s)$ to the state s as known, and iterates until no more states can be labeled in this manner. The algorithm can be stopped when the node selected is a goal node. The measure $g(s)$ for that node encodes the cost of a cheapest plan that solves the planning problem P.

Algorithms for solving path-finding problems over directed graphs such as Dijkstra's algorithm run in time that is polynomial in the number of nodes in the graph. This, however, is not good enough in planning where the number of nodes is *exponential* in the number of problem variables F. Since in Strips the variables are boolean, the number of nodes in the graph to search can be in the order of $2^{|F|}$. For example, if the problem involves 30 variables, the number of nodes in the graph is $1,073,741,824$, and if it involves 100 variables, the number of nodes is larger than 10^{30}. If it takes 1 second to generate 10^7 nodes (a realistic estimate given current technology), it would take more than 10^{23} seconds to generate all nodes in the latter case. This is however almost one million times the estimated age of the universe.[3] Planning algorithms able to deal with tens of variables or more thus cannot be based on exhaustive methods such as Dijkstra's, and need to search for paths in graphs in a more focused manner.

A more concrete illustration of the complexity inherent in the planning problem can

[3]The age of the universe is estimated at 13.7×10^9 years approximately. Generating 2^{100} nodes at 10^7 nodes a second would take in the order of 10^{15} years, as $2^{100}/(10^7 * 60 * 60 * 24 * 365) \approx 4 * 10^{15}$.

be obtained by considering the familiar Blocks World. Figure 1 shows an instance of the problem 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 moving a clear block (a block with no block on top) on top of another clear block or 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. The figure shows the graph associated to the problem, whose solution is a path connecting the node representing the initial situation with a node representing a goal situation.

The Blocks World is simple for people but until recently, not so easy for *domain-independent* planners that must accept *any* planning problem $P = \langle F, I, O, G \rangle$ in their input, for *any domain*, and solve it automatically *without assuming additional knowledge.* Indeed, the number of states in a Blocks World problem with n blocks is exponential in n, as the states include all the $n!$ possible towers of n blocks plus additional combinations of lower towers.

It may be argued that people solve a problem like Blocks World with *domain-specific control knowledge*; namely, knowledge about *how* problems in the domain are solved, rather than just knowledge about *what* the problem is about. Domain-specific control knowledge for Blocks, for example, may state that 'bad placed' blocks and blocks above them must be all moved, and that blocks should be moved onto other blocks only when the latter are 'well placed'. Many arguments have been raised against the view of humans as 'general problem solvers', some casting the mind as a collection of specialized modules evolved to solve the specific problems of our Pleistocene ancestors [100]. Yet, the arguments against generality are speculative, and it's not clear what the modules are, how specialized they are, nor what are the specific problems that they evolved to solve. Also, if there are many modules, some mechanism would still have to determine which module to use and when, and this mechanism can't be domain-specific. Approaches that deny general problem solving abilities may be actually pushing the complexity of the problem one level up, to the module or modules that must control the rest of the modules.

Actually, these questions involve computational and complexity issues that cannot be answered fully without knowing whether some form of 'generality' is computationally feasible, and what the price for this generality is. Indeed, why deny general problem solving abilities a priori if it is computationally feasible and cheap? Of course, due to the theoretical complexity of planning we cannot expect a general planner to efficiently solve every problem from every possible domain, yet a general planner will be useful enough for an agent if it can solve problems from many domains, provided that the domains themselves are not inherently that hard, and that the size of a problem, as measured by the number of actions and variables, is not by itself an impediment to its solution. This is actually what modern AI planners manage to do. We will see below how this is accomplished and why this is relevant from a cognitive perspective.

5 Heuristics

One way to search for a path linking a source[2] node to a goal node in a huge graph is by providing the search with a sense of direction. Consider for example the problem of looking on a map for a good route between Los Angeles (LA) and New York (NY), regarding the map as a graph whose nodes are cities, and whose edges connect pairs of cities through directed routes at a cost proportional to the route length. Dijkstra's algorithm, sketched above, provides a way for finding routes in such graphs. If our map covers all of North America, however, the algorithm would appear to behave in a strange manner, finding first shortest paths to all cities that are closer to LA than NY (like Mexico City) whether they are on the way to NY or not. The search in such a case is said to be blind, meaning that information about the goal is not used in the exploration of the map. This is definitely not the way that people search for routes in a map. If they want to go to a city that is Northeast, they will look for routes headed in that direction. The search is then focused and the total number of cities in the map is less of a problem. This basic intuition about goal-directed search was incorporated into AI search algorithms in the 60's and 70's in the form of *heuristic functions*: functions $h(s)$ that provide a quick-and-dirty estimate of the distance or cost separating a state s from a goal state. In the route finding problem, for example, a useful heuristic $h(s)$ is the *Euclidean distance* separating a city s from the target city. This is a heuristic that is very easy to compute and which provides a good approximation of the optimal cost $h^*(s)$ of reaching the goal from s, whose exact computation would be much harder (as hard indeed as solving the original problem). Interestingly, if the evaluation function $g(s)$ used to select the next node to label in Dijkstra's algorithm is replaced by the more informed function $f(s) = g(s) + h(s)$ that incorporates both the cost from s_0 to s and an estimate of the cost to go from s to the goal, an *heuristic search algorithm* known as A* is obtained that can look for paths to the goal much more effectively [45].[4] Heuristic search algorithms express a form of goal-driven search, where the goal is no longer passive in the search process, but actively biases the search through the heuristic term $h(s)$. In the extreme case in which the heuristic $h(s)$ is perfect, i.e., the estimates $h(s)$ and the true costs $h^*(s)$ coincide for all states, A* goes straight to the goal with no search at all. At the same time, if the heuristic is completely uninformative, as the null heuristic $h(s) = 0$ for all states, A* reduces to Dijkstra's algorithm. In the middle, when $0 \leq h(s) \leq h^*(s)$ for all s, the heuristic h is said to be *admissible* and A* preserves the optimality properties of Dijkstra but does less work. Indeed, the closer to h^*, the more informed the heuristic, and the less number of states that are visited in the search. Heuristic search methods have been used for finding optimal solutions to problems like Rubik's Cube that involve more than 10^{19} states. The key is the use of suitable admissible heuristic functions $h(s)$ that allow for optimal solutions while considering a tiny fraction of the problem states [65].

6 Domain-independent Generation of Heuristics

Heuristic search algorithms express a form of goal-directed search where heuristic functions are used to guide the search towards the goal. A key question is how such heuristics can be obtained for a given problem. A useful heuristic is one that provides good estimates of the cost to the goal and can be computed reasonably fast. Heuristics are traditionally devised according to the problem at hand: the Euclidean distance is a good heuristic for

[4]This description of A* assumes that the heuristic is monotonic. A more complete description of A* and variations can be found in the standard textbooks [84, 89].

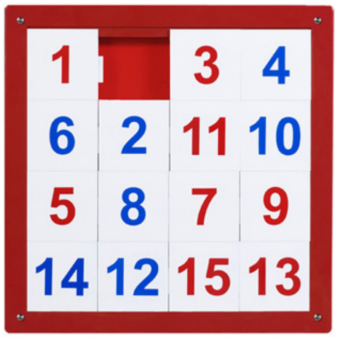

Figure 2: The sliding 15-puzzle where the goal is to get to a configuration where the tiles are ordered by number with the empty square last. The actions allowed are those that slide a tile into the empty square. While the problem is not simple, the heuristic that sums the horizontal and vertical distances of each tile to its target position is simple to compute and provides an informative estimate of the number of steps to the goal. In planning, heuristic functions are obtained automatically from the problem representation.

route finding, the sum of the Manhattan distances of each tile to its destination is a good heuristic for the sliding puzzles, and so on.[5] The general idea that emerges from the various heuristics developed for different problems is that heuristics $h(s)$ can be seen as encoding the cost of reaching the goal from the state s in a problem that is simpler than the original one [95, 77, 84]. For example, the sum of the Manhattan distances in the sliding puzzles (Fig. 2), corresponds to the optimal cost of a simplification of the puzzle where tiles can be moved to adjacent positions, whether such positions are empty or not. Similarly, the Euclidean heuristic for route finding is the cost of a simplification where straight routes are added between any pair of cities. The simplified problems are normally referred to as *relaxations* of the original problem.

The key development in modern planning research in AI was the realization that useful heuristics could be derived automatically from the representation of the problem in a domain-independent language such as Strips [74, 18, 13]. It does not matter what the problem is about, a domain-independent relaxation is obtained directly and effectively from the problem representation from which heuristics that are adapted to the problem can be computed.

The most common and useful domain-independent relaxation in planning is the *delete-relaxation* that maps a problem $P = \langle F, I, O, G \rangle$ in Strips into a problem $P^+ = \langle F, I, O^+, G \rangle$ that is exactly like P but with the actions in O^+ set to the actions in O with empty delete lists. That is, the delete-relaxation is a domain-independent relaxation that takes a planning problem P and produces another problem P^+ where atoms are added exactly like in P but where atoms are never deleted. The relaxation implies, for example, that objects and agents can be in “multiple places” at the same time, as when an object or an agent moves into a new place, the atom encoding the old location is not deleted. Relaxations, however, are not aimed at providing accurate models of the world; quite the opposite, simplified and even

[5]The Manhattan distance of a tile is the sum of the horizon and vertical distances that separate its current position from its destination.

meaningless models of the world that while not accurate yield useful heuristics.

The domain-independent delete-relaxation *heuristic* is obtained as an approximation of the optimal cost of the delete-relaxation P^+, obtained from the cost of a plan that solves P^+ not necessarily optimally.[6] The reason that an approximation is needed is because finding an optimal plan for a delete-free planning problem like P^+ is still a computationally intractable task (also NP-hard). On the other hand, finding just one plan for the relaxation whether optimal or not, can be done quickly and efficiently. The property that allows for this is *decomposability:* a problem without deletes is decomposable in the sense that a plan π for a joint goal G_1 and G_2 can always be obtained from a plan π_1 for the goal G_1 and a plan π_2 for the goal G_2. Indeed, it can be shown that the concatenation of the two plans $\pi = \pi_1, \pi_2$ in either order, is one such plan. This property allows for a simple method for computing plans for the relaxation from which the heuristics are derived.

The main idea behind the procedure for computing the heuristic $h(s)$ for an arbitrary planning problem P can be explained in a few lines. For this, let $P(s)$ refer to the problem that is like P but with the initial situation set to the state s, and let $P^+(s)$ stand for the delete-relaxation of $P(s)$, i.e., the problem that is like $P(s)$ but where the delete-lists are empty. The heuristic $h(s)$ is computed from a plan for the relaxation $P^+(s)$ that is obtained using the decomposition property and a simple iteration. Basically, the plans for achieving the atoms p that are already true in the state s, i.e., $p \in s$, are the empty plans, and if π_1, π_2, $\ldots$, π_m are the plans for achieving each of the preconditions p_1, p_2, $\ldots$, p_m of an action a that has the atom q in the add list, $\pi = \pi_1, \pi_2, \ldots, \pi_m$ followed by the action a is a plan for achieving q. It can be shown that this iteration yields a plan in the relaxation $P^+(s)$ for each atom p that has a plan in the original problem $P(s)$, in a number of steps that is bounded by the number of variables in the problem. A plan for the actual goal G of $P(s)$ in the relaxation $P^+(s)$ can be obtained in a similar manner by just concatenating the plans for each of the atoms q in G in any order. Such a plan for the relaxation $P^+(s)$, denoted as $\pi^+(s)$, is usually called the *relaxed plan.* The heuristic $h(s)$ is set to the cost of such a plan. A better estimate can be obtained if duplicate actions are removed every time plans are concatenated [58]. Other heuristics based also on the delete-relaxation such as the additive heuristic [13] and the FF heuristic [48], are as informative as this heuristic but can be computed faster. Notice that a plan $\pi^+(s)$ for the relaxation yields the heuristic value $h(s)$ that estimates the cost from s to the goal for the state s only. This computation thus must be repeated for every state whose heuristic value is needed.

Figure 3 displays a fragment of the directed graph corresponding to the Blocks World problem shown in Figure 1, with the automatically derived heuristic values next to some of the nodes. The heuristic values shown are computed very fast, in low polynomial time, using an algorithm similar to the one described above, with $h(s)$ representing an approximation of the number of actions needed (cost) to solve the relaxed problem $P^+(s)$. Actually, the instance shown can be solved with *no search at all* by just selecting in each node, starting from the source node, the action that leads to the node with a lower heuristic value (closer to the goal). The resulting plan is shown as a red path in the figure.

In order to get a more vivid idea of where the heuristic values shown in the figure come

[6]Heuristics derived in this way are not admissible; i.e., they may overestimate the true costs and hence are not suitable for computing optimal plans.

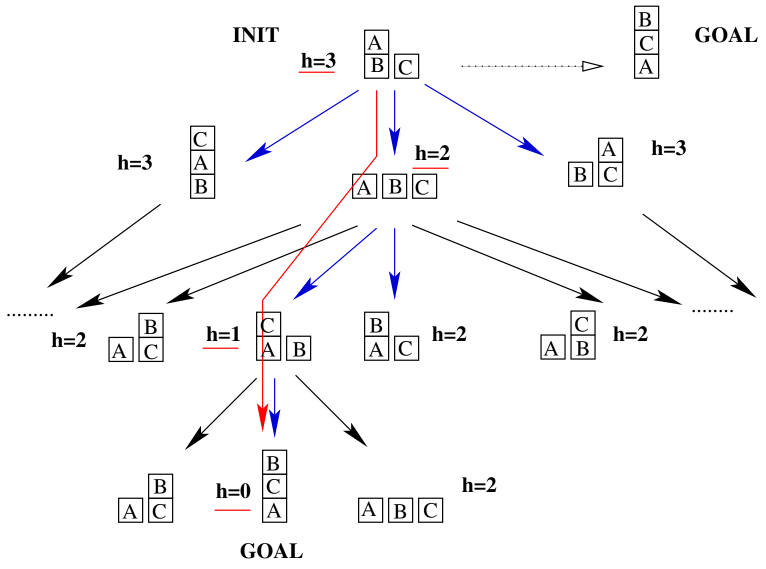

Figure 3: A fragment of the graph corresponding to the blocks problem with the automatically derived heuristic values next to some of the nodes. The heuristic values are computed in low polynomial time and provide the search with a sense of direction. The instance can actually be solved without any search by just performing in each state the action that leads to the node with a lower heuristic value (closer to the goal). The resulting plan is shown in red; helpful actions are shown in blue.

from, consider the heuristic $h(s)$ for the initial state where block A is on B, and both B and C are on the table. In order to get the goal 'B on C' in the relaxation from the state s, two actions are needed: one to get A out of the way to achieve the preconditions for moving B, the second to move B on top of C. On the other hand, in order to achieve the second goal 'C on A' in the relaxation from s, just the action of moving C to A is needed. The result is a heuristic value $h(s) = 3$ as shown, which actually coincides in this case with the cost of the best plan to achieve the joint goal from s in the non-relaxed problem $P(s)$. Nonetheless, this is just a coincidence, and indeed, the best plans in the relaxation $P^+(s)$ can be quite different than the best plans in the original problem $P(s)$. The best plan for $P(s)$ is unique, moving A to the table, then C on A, and finally B on C. On the other hand, a possible optimal plan in the relaxation $P^+(s)$ is to move first C on A, then A on the table, and finally B on C. Of course, this plan does not make any sense in the real problem where A can't be moved when covered by C, yet the relaxation is not aimed at capturing the real problem or the real physics, it is aimed at producing informative but quick estimates of the cost to the goal. The reader can verify that for the leftmost child s' of the initial state s, the costs of the problem $P(s')$ and the relaxation $P^+(s')$ no longer coincide. The former is 4, while the latter is 3, the difference arising from the goal 'C on A' that in the original problem must be undone and then redone. On the other hand, in the relaxation, this is never needed as no atom is ever deleted.

One last point about the domain-independent relaxation and the resulting heuristics: the automatic computation of the heuristic value $h(s)$ from a plan $\pi^+(s)$ for the relaxation $P^+(s)$ yields also relevant *structural* information that is not captured in the estimates themselves.

In particular, among all the actions that are applicable in the state s, the 'relaxed plan' suggests the ones that appear as most relevant. These are the actions that are applicable in the state s and form part of the relaxed plan. In planning, such actions are said to be 'helpful' [48], and this structural information is used too, in one way or another in every state-of-the-art planner. Figure 3 shows the helpful actions in each of the states on the way to the goal by means of arrows depicted in blue. As it can be seen, two of the three applicable actions in the root state are helpful, just two of the six applicable actions are helpful in the second state, and just one action is helpful in the last state before the goal. It is not always the case that the best action in a state is among the ones found to be helpful in that state, or among the ones leading to the children with lowest heuristic values, yet this is often the case and this suffices for making the heuristic and the relaxation so useful. It is also interesting that the notion of helpful actions that emerges from the relaxation suggests a model for identifying the actions that appear as most promising in a state which does not require evaluating them first. This information can be used indeed for deciding which actions to consider in a state, possibly ignoring the other actions. The resulting search is not necessarily complete but it can be quite effective [48]. We will come back to this point below.

7 Heuristic Search

Once a heuristic $h(s)$ is available for estimating the cost to the goal from s, the question is how to use it for finding a path to the goal. The simplest method for using the heuristic is by selecting the action a in s that produces the state s' that appears to be closest to the goal; i.e. the one with minimum $h(s')$ value. This algorithm is known as *hill-climbing*, from its use in maximization problems, as it moves the search along the slope of the heuristic function h where a value $h(s) = 0$ denotes a goal. The problem with this and other local search algorithms is that they can get stuck in states where none of the children improves on (decreases) the heuristic value of the parent.

There are many heuristic search algorithms that avoid the problems of hill-climbing, and they can be classified into two groups: *off-line* and *on-line* algorithms. If planning is thought of as "thinking before acting", off-line search algorithms can be understood as thinking all the way to the solution before acting, while on-line algorithms interleave thinking and acting. The algorithm A* described above is an off-line search algorithm with several guarantees: it is complete; i.e., it will find a solution if there is one, and it is optimal if the heuristic $h(s)$ is admissible (doesn't overestimate true costs). On the other hand, an algorithm like hill-climbing can be used as an on-line algorithm that iteratively applies the action that leads to the best child according to the heuristic. In the on-line setting, the algorithm is known as the *greedy algorithm*; the name greedy implying that actions are selected fast after a shallow and quick lookahead. Of course, more informed versions of this greedy on-line algorithm can be obtained by performing a deeper lookahead, with a lookahead of two levels, resulting in the action that leads to the best grandchild, a lookahead of three levels, resulting in the action that leads to the best great grandchild, and so on. A similar idea is used in chess playing programs that involve a slightly different[2] model with adversarial agents [79, 89]. Two problems with greedy algorithms enhanced with a fixed depth lookahead are that they can be trapped into a loop, returning to states that have been visited earlier in the execution, and that the computational effort grows exponentially with the lookahead depth. The loop problem in the greedy algorithm has a very elegant and powerful solution: the algorithm *Learning Real Time A** or *LRTA** [64] behaves exactly as the greedy algorithm but once it selects the action a in the state s, and before moving to the resulting state s', it changes the heuristic value $h(s)$ to $c(a, s) + h(s')$, where $c(a, s)$ is the cost of the action a in s, and $h(s')$ is the heuristic value of s'. If the problem has no states from which the goal cannot be reached (dead-ends), this simple type of learning guarantees that the goal will eventually be reached, and moreover, that if the process is restarted many times while preserving the heuristic values that have been learned, the goal will eventually be reached optimally, if the initial heuristic is admissible. The algorithm LRTA* is also interesting because it can be easily generalized to other models where actions have probabilistic effects and states are fully or partially observable. The generalized algorithm is known as Real-Time Dynamic Programming or RTDP [8, 14, 15]. One last on-line algorithm for probabilistic models that has become popular in recent years due to its breakthrough performance in the game of Go is UCT [61, 38]. Closely related to UCT are on-line variants of AO* [17], an extension of the A* algorithm for models where actions have non-deterministic effects [68, 84].

8 Heuristics, Values, and Appraisals

Heuristic evaluation functions are used in other settings like chess playing programs and reinforcement learning. In chess, the evaluation functions are programmed by hand [79, 84], while in reinforcement learning, they are learned by trial-and-error [98]. Reinforcement learning is a family of *model-free* methods that have been shown to be effective in low-level tasks, and to provide an accurate account of learning in the brain [92]. Heuristic evaluation functions in domain-independent planning are computed instead using *model-based methods* where suitable relaxations are solved from scratch. These methods have been shown to work over large problems involving hundred of actions and fluents, and interesting lessons can be drawn from these methods as well. Indeed, while feelings and emotions are currently thought of as providing the appraisals that are necessary for navigating in a complex world [31], there are actually very few accounts of how such appraisals may be computed. In planning, the appraisals that manage to integrate information about the current situation, the goal, and the actions, for directing the agent towards the goal, are the heuristics computed from relaxations of the problem. The computational model of appraisals that is based on the solution of relaxations that can be solved in low-polynomial time, suggests potential explanations for a number of observations. For example, the heuristics that result are 'fast and frugal' but unlike the heuristics considered by Gigerenzer and others [41, 40], they are also general: they apply to all the problems that fit the classical planning model or that can be cast in that form. The use of relaxations can also potentially explain why appraisals may be opaque from a cognitive point of view, and thus not be conscious [104, 47, 54]. This is because the appraisals are obtained from a simplified model where, for example, objects can be in different places at the same time, and hence where the meaning of the symbols is different than the meaning of the symbols in the 'true' model. Finally, heuristic appraisals provide the agent with a sense of direction or 'gut feeling' that guide the action selection in the presence of many alternatives, while avoiding an infinite regress in the decision process. Indeed, the computational role of these appraisals is to avoid or at least to reduce the need to search. Explicit evaluation of all possible courses of actions is not feasible computationally, and the heuristics provide the necessary focus. Actually, as discussed above, relaxation-based heuristics in planning do not only provide an account of the value of the different options, but also of the actions that are worth evaluating; the so-called helpful actions [48]. This second aspect, although not necessarily in this form, may potentially help to explain a key difference between programs and humans in games such as chess, for example, where it is well known that humans consider much fewer moves than programs.

9 Alternative Planning Methods

While the heuristic search approach to planning has come to dominate classical planning, many other methods have been proposed, and some of them are widely used and scale up well too. GPS, the first AI planner and one of the first AI programs, was introduced by Newell and Simon in the 50's [80, 81]. It introduced a technique called means-ends analysis where differences between the current state and the goal are identified and mapped into operators that can decrease those differences. Since then, the idea of means-ends analysis has been refined and extended in many ways, in the formulation of 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 planner was UCPOP [85], an implementation of an elegant planning method known as partial-order planning, where plans are not searched either forward from the starting state or backward from the goal, but are constructed from a decomposition scheme in which joint goals are decomposed into subgoals, which create as further subgoals the preconditions of the actions that can establish them [91, 99, 72]. The actions that are incorporated into the plan are partially ordered as needed in order to resolve possible conflicts among them. Partial-order planning algorithms are sound and complete, but do not scale up well, as there are too many choices to make and too little guidance on how to make them.

The situation in planning changed drastically in the middle 90's with the introduction of Graphplan [12], an algorithm that appeared to have little in common with previous approaches but scaled up much better. Graphplan builds a *plan graph* in polynomial time reasoning forward from the initial state, which it then searches backward from the goal to find a plan. It was shown later that the reason Graphplan scaled up well was due to a powerful admissible heuristic implicit the plan graph [46]. The success of Graphplan prompted other approaches. In the SAT approach [55], the planning problem for a fixed planning horizon is converted into a general *satisfiability* problem expressed as a set of clauses (a formula in conjunctive normal form or CNF) that is fed into state-of-the-art SAT solvers that currently manage to solve huge SAT instances even if the SAT problem is NP-complete [11]. A clause is a disjunction of literals, i.e., propositional symbols or their negations as in $x \vee \neg y \vee z$, and a set of clauses is satisfiable if there is a truth assignment to the symbols such that each clause has at least one true literal. In the CNF encoding of the planning problem, atoms and actions are indexed with time indices that range from zero until a planning horizon that is increased one by one until the resulting clauses are satisfiable and a plan can be read from the satisfying assignment. Due to the excellent performance of current SAT solvers, SAT planners manage to solve large problems very fast, making them practically competitive with heuristic search planners [88]. State-of-the-art heuristic search planners use heuristic values derived from the delete-relaxation [74, 18], information about the action that are most helpful [48], and implicit subgoals of the problem, called landmarks, that are also extracted automatically from the problem with methods similar to those used for deriving heuristics [49, 87]. Recently, the success of classical planners has also been explained in term of a structural width parameter that appears to be bounded and small in many domains when goals are restricted to single atoms. Such problems can be solved in time that is exponential in their width, while problems with joint goals can often be decomposed easily into a sequence of low-width problems with single goals [67].

10 Richer Planning Models

Classical planning is planning with deterministic actions from an initial state that is fully known. Many planning problems, however, involve features that are not part of this basic model such as uncertainty, incomplete information, and soft goals. Two types of methods have been pursued for dealing with such features: a *top-down* approach, where *native solvers* have been developed for more expressive planning models, and a *bottom-up* approach, where the power of classical planners is exploited by means of *translations.*

10.1 MDP and POMDP Planning

MDP and POMDP planners are examples of native solvers for more expressive planning models that accommodate stochastic actions, and either full or partial state observability. A Markov Decision Process (MDP) is a state model where the state transition function $f(a, s)$ is replaced by state transition *probabilities* $P_a(s'|s)$, and the next state s', while no longer predictable with certainty, is assumed to be *fully observable* [10, 86, 21]. A solution to an MDP is a function π from states into actions, called a *policy*, that drives the system to a goal state with certainty. A policy induces a probability distribution over the possible state trajectories, and since every state trajectory has a cost, an optimal policy is a policy that drives the system to the goal at a minimum expected cost.

Partially Observable MDPs (POMDPs) extend MDPs by relaxing the assumptions that states are fully observable [3, 10, 53]. In a POMDP, a set of observation tokens $o \in O$ is assumed along with a *sensor model* $P_a(o|s)$ that relates the true but hidden state s of the system with the observable token o. In POMDPs, the initial state of the system is not known but is characterized by a probability distribution, and the task is to drive the system to a final, fully observable target state. Solutions to POMDPs are closed-loop controllers that map *belief states* into actions, with optimal solutions reaching the target state at a minimum expected cost. The belief states are probability distributions over the states.

A simple example of a POMDP is a robot that has to reach a certain location by means of actions whose effects can only be predicted probabilistically. The state of the problem, which is the location of the robot, is not fully observable but rather sonars or other feedback mechanisms are used to provide the robot with[2] partial information about the state. A map showing the locations that are blocked by obstacles may be available, but the map cannot be used in a direct way, as the robot does not know with certainty its location in the map. On the other hand, if the robot can have perfect access to its location, the problem is not a POMDP but an MDP.

While MDPs and POMDPs are commonly described using positive or negative *rewards* rather than positive *costs*, and using *discount factors* rather than *goal states*, simple transformations are known for translating discounted reward MDPs and POMDPs into equivalent *goal MDPs* and *goal POMDPs* as above that are strictly more expressive [10, 15]. From a computational point of view, traditional dynamic programming algorithms have been used to solve MDPs and POMDPs off-line [9, 50, 10], yet the methods that scale up best to larger problems are on-line methods, closely related to the on-line heuristic search algorithms considered above for deterministic problems. These include Real-time Dynamic Programming [8, 14, 15, 62], and UCT [61, 94, 57, 17]. These algorithms are also related to reinforcement learning methods, which can be characterized as methods for solving an MDP by trial-and-error without knowing the value of the cost and probability parameters [98]. Normally, reinforcement learning algorithms learn a representation of the optimal MDP policy without learning the value of these parameters. These are called *model-free* MDP methods [103, 97]. On the other hand, there are approaches that incrementally learn the MDP model and derive the policy from the model. These are called *model-based* reinforcement learning methods, and the most effective of them reduce the learning problem to a planning problem over an 'optimistic' model that is refined incrementally [56, 24, 2]. A last class of POMDP methods map probabilistic planning problems into probabilistic reasoning problems over Bayesian Networks [4, 101, 20], in analogy to the SAT approach to deterministic planning, where atoms and actions are indexed in time up to a given planning horizon. While the reduction of planning to inference, deductive or probabilistic, is appealing, the scalability-quality tradeoff in the probabilistic case, unlike the SAT approach in the deductive case, is not yet clear in comparison with state-of-the-art methods.

10.2 Translations into Classical Planning

Translation-based approaches handle features that are absent from the classical model such as soft-goals, plan-constraints, extended temporal goals, uncertainty, and partial feedback, by compiling them away. We focus only on translations that preserve the semantics of the original problem, yet approximate translations have been found useful as well. For example, FF-Replan is an on-line planner for MDPs whose value has been proved in some of the MDP planning competitions held so far [105]. FF-Replan's approach to MDP planning is simple but effective. It selects the action to do next using a simple relaxation of the MDP: first the probabilities associated with the non-deterministic effects of a probabilistic action are dropped, then the non-deterministic effects of the same action are mapped into different deterministic actions. The relaxation thus maps an MDP where the uncertain effects are controlled by nature into a deterministic problem where the uncertain effects are controlled by the planning agent. Once the plan is obtained for the relaxation, the plan is executed until the agent finds itself in a state that is different from the one predicted by the relaxation. The process resumes from this state and the ex2cution terminates when the goal is reached. The resulting on-line planner is not optimal and can get trapped into dead-ends, yet in most domains, the simplification works rather well. A similar idea, where closed-loop feedback controls for stochastic systems are designed using a simplified model, is common in control engineering [82].

Features that are absent from the classical planning model include rewards, and in particular *soft goals*. Soft goals refer to formulas that if achieved along with the goal entail a positive utility. In the presence of soft goals, the task is to compute action sequences that achieve the goal and maximize overall utility, rather than sequences that achieve the goal and minimize cost. The utility is defined as the sum of the utilities of the soft goals achieved minus the cost of the action sequence. Soft goals express soft preferences as opposed to hard goals that express hard preferences. For example, getting a new T-shirt can be a hard goal for John, while getting a second T-shirt can be regarded as a soft goal. All plans must thus deliver John a new T-shirt, but whether the best plans will deliver him a second T-shirt or not will depend on the extra costs and utilities involved. Interestingly, it turns out that soft-goals can be compiled away easily and efficiently resulting into standard classical planning problems with hard goals only [59]. For soft-goal atoms A one just needs to create new atoms A' and make them into hard goals that can be achieved in one of two ways: by means of the new action *collect*(A) with precondition A and cost 0, or by means of the new action $forego(A)$ with no precondition and cost equal to the utility of A. The best plans for the soft goals become the best plans for the resulting classical problem.

Uncertain information can also be compiled away in conformant problems with deterministic actions [83]. These are problems where the initial state is partially known and a plan to the goal is sought that would work for any possible initial state. The translation maps a conformant problem P into a classical problem, whose solutions, computable with off-the-shelf classical planners, encode the solutions to P. The complexity of the translation is exponential in a width parameter that is often bounded and small. The ideas behind this translation have been used since to define effective action selection mechanisms for on-line planning in the presence of *partial observability* [1, 16, 93] and for computing solutions to planning problems with uncertainty and partial feedback in the form of *finite-state controllers*. One such finite-state controller is shown in Fig. 4 for a problem inspired by the use of deictic representations where objects are not assumed to have unique names but can be referred to by means of suitable indexical expressions and markers [28, 7]. In the figure, a visual marker, that is the circle on the lower left, must be placed on top of a green block by moving it one cell at a time. The location of the green block is not known (it can be anywhere), and the observations are whether the cell currently marked contains a green block (G), a non-green block (B), or neither (C), and also whether the marked cell is at the level of the table (T) or not (–). The finite-state controller shown on the right has been obtained by running a *classical planner* on a suitable translation of the problem [19]. The controller has two states, the initial state q_0 and the state q_1. An edge $q \rightarrow q'$ with label o/a means to do the action a when observing o in the state q, and to move then to the state q'. For example, when the observation TC is received in q_0, the action of moving the marker to the Right is done, and the controller remains in the same state. The reader can check that the controller obtained with the classical planner not only solves the original problem on the left, but also any modification of it resulting from changes in the number or configuration of blocks. In other words, due to its representation, the solu2tion to the original problem generalizes to a much larger class of problems. The problem of devising controllers or strategies that work

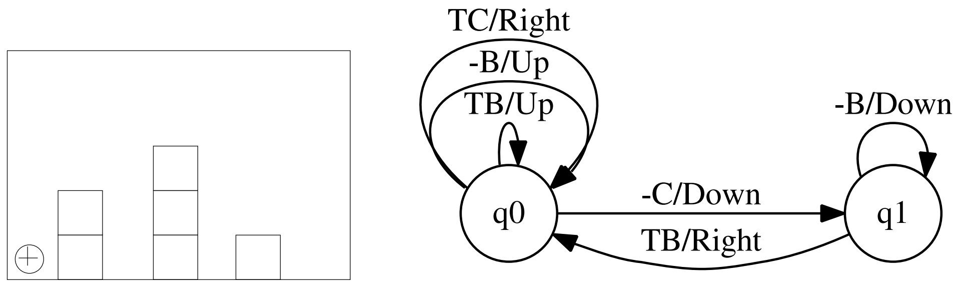

Figure 4: *Left:* Problem where visual-marker (circle on the lower left) must be placed on top of a green block by just observing what's on the marked cell. *Right:* Finite-state controller obtained with a classical planner from suitable translation. The controller has two states, the initial state q_0 and the state q_1. An edge $q \rightarrow q'$ with label o/a means to do the action a when observing o in the state q, and to move then to the state q'. The controller shown solves the problem, and any variation of it resulting from changes in the number or configuration of blocks.

for many or even all domain instances is usually referred to as generalized planning [70, 51].

11 Challenges

There are several open challenges in planning research. One of them is planning in the presence of other agents that plan, often called multiagent planning. People do this naturally all the time: walking on the street, driving, etc. The first question is how plans should be defined. This is a subtle problem and many proposals have been put forward, often building on equilibria notions from game theory [22, 66, 23]. Yet, there are currently no models, algorithms, or implementations of domain-independent planners able to plan meaningfully and efficiently in such settings. This is not entirely surprising, however, given the known limitations of game theory as a descriptive theory of human behavior [27]. Eventually, a working theory of multiagent planning could shed light on the computation, nature, and role of social emotions in multiagent settings, very much as single-agent planning may shed light on the computation, nature, and role of goal appraisals and heuristics in the single-agent setting. A second open problem is the automatic construction and use of hierarchies. Hierarchies form a basic component of Hierarchical Task Networks or HTNs, an alternative model for planning that is concerned with the encoding of *strategies for solving problems* [99, 34, 39], yet hierarchies play no role in state-of-the-art domain independent planners which are completely flat. There is a large body of work on abstract problem solving that is relevant to this question, both old [90, 63, 60, 5] and new [76, 69, 52], but no robust answer yet. A third open problem is learning the planning models by interacting with the environment. We have discussed model-based reinforcement learning algorithms that actively learn model parameters such as probabilities and rewards [56, 24, 2], yet a harder problem is learning the states themselves from partial observations. Several attempts to generalize reinforcement learning algorithms to partially observable settings have been made, some of which learn to identify useful features and feature histories [73, 96, 102], but none so far that can come up with the states and models themselves in a robust and scalable manner.

12 Discussion

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* in AI is more technical and experimental, and is focused not on reproducing intuitions but on *scalability.* This may give the impression 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 [104, 47, 40]. 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. By focusing on the study of meaningful models and the computational methods for dealing with them *effectively,* AI may prove its relevance to the understanding of 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 developmental psychology for understanding how children acquire and use causal relations [42], and the use of reinforcement learning algorithms in neuroscience for interpreting the activity of dopamine cells in the brain [92], 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. In this paper, we have reviewed work in computational models of planning in AI, with an emphasis on deterministic planning models where automatically derived relaxations and heuristics manage to integrate information about the current situation, the goal, and the actions for directing the agent effectively towards the goal. The computational model of goal appraisals that is based on the solution of low-polynomial relaxations may shed light on the computation, nature, and role of other types of appraisals, and on why appraisals are opaque to cognition and cannot be rendered conscious or articulated in words.

Acknowledgments.

The author thanks the reviewers for useful comments. H. Geffner is partially supported by grants TIN2009-10232 and CSD2010-00034 (SimulPast), MICINN, Spain, and EC-7PM-SpaceBook.

References

[1] A. Albore, H. Palacios, and H. Geffner. A translation-based approach to contingent planning. In *Proc. IJCAI-09*, pages 1623–1628, 2009.

[2] J. Asmuth and M. Littman. Learning is planning: near bayes-optimal reinforcement learning via Monte-Carlo tree search. In *Proc. UAI*, pages 19–26, 2011.

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

[4] H. Attias. Planning by probabilistic inference. In *Proc. of the 9th Int. Workshop on Artificial Intelligence and Statistics*, 2003.

[5] F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71:43–100, 1994.

[6] C. Backstrom and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

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

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

[9] R. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

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

[11] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications.* IOS Press, 2009.

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

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

[14] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. 13th Int. Conf. on Automated Planning and Scheduling (ICAPS-2003)*, pages 12–31. AAAI Press, 2003.

[15] B. Bonet and H. Geffner. Solving POMDPs: RTDP-Bel vs. point-based algorithms. In *Proc. IJCAI*, pages 1641–1646, 2009.

[16] B. Bonet and H. Geffner. Planning under partial observability by classical replanning: Theory and experiments. In *Proc. IJCAI-11*, pages 1936–1941, 2011.

[17] B. Bonet and H. Geffner. Action selection for MDPs: Anytime AO* vs. UCT. In *Proc. AAAI-2012*, 2012.

[18] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proc. AAAI-97*, pages 714–719, 1997.

[19] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. ICAPS-09*, pages 34–41, 2009.

[20] M. Botvinick and J. An. Goal-directed decision making in the prefrontal cortex: a computational framework. *Advances in Neural Information Processing Systems (NIPS)*, pages 169–176, 2008.

[21] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *J. Artif. Intell. Res. (JAIR)*, 11:1–94, 1999.

[22] M. Bowling, R. Jensen, and M. Veloso. A formalization of equilibria for multiagent planning. In *Proc. IJCAI-03*, pages 1460–1462, 2003.

[23] R. Brafman, C. Domshlak, Y. Engel, and M. Tennenholtz. Planning Games. In *Proc. IJCAI-09*, pages 73–78, 2009.

[24] R. Brafman and M. Tennenholtz. R-Max: A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003.

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

[26] T. Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[27] C. Camerer. *Behavioral game theory: Experiments in strategic interaction.* Princeton University Press, 2003.

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

[29] A. J. Coles, A. Coles, A. G. Olaya, S. Jiménez, C. L. López, S. Sanner, and S. Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88, 2012.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2009.

[31] A. Damasio. *Descartes' Error: Emotion, Reason, and the Human Brain.* Quill, 1995.

[32] D. Dennett. *Kinds of minds.* Basic Books New York, 1996.

[33] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

[34] K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proc. AAAI-94*, pages 1123–1123, 1994.

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

[36] H. Geffner. Heuristics, planning, cognition. In R. Dechter, H. Geffner, and J. Halpern, editors, *Heuristics, Probability and Causality. A Tribute to Judea Pearl.* College Publications, 2010.

[37] H. Geffner and B. Bonet *Advanced Introduction to Planning: Models and Methods.* Morgan & Claypool. May 2013.

[38] S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *Proc. ICML*, pages 273–280, 2007.

[39] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice.* Morgan Kaufmann, 2004.

[40] G. Gigerenzer. *Gut feelings: The intelligence of the unconscious.* Viking Books, 2007.

[41] G. Gigerenzer and P. Todd. *Simple Heuristics that Make Us Smart.* Oxford, 1999.

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

[43] J. Gratch. Why you should buy an emotional planner. In *Proc. of Agents'99 Workshop on Emotion-based Agent Architectures (EBAA'99)*, 1999. At http://emotions.usc.edu/∼gratch/gratch-ebaa99.pdf.

[44] J. Gratch and S. Marsella. A domain-independent framework for modeling emotion. *Cognitive Systems Research*, 5(4):269–306, 2004.

[45] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4:100–107, 1968.

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

[47] R. Hassin, J. Uleman, and J. Bargh. *The new unconscious.* Oxford University Press, 2005.

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

[49] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

[50] R. Howard. *Dynamic Probabilistic Systems–Volume I: Markov Models.* Wiley, New York, 1971.

[51] Y. Hu and G. De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *Proc. IJCAI*, pages 918–923, 2011.

[52] A. Jonsson. The role of macros in tractable planning over causal graphs. In *Proc. IJCAI-07*, pages 1936–1941, 2007.

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

[54] D. Kahneman. *Thinking, fast and slow.* Farrar, Straus and Giroux, 2011.

[55] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proc. AAAI*, pages 1194–1201, 1996.

[56] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2):209–232, 2002.

[57] T. Keller and P. Eyerich. PROST: Probabilistic planning based on UCT. In *Proc. ICAPS*, 2012.

[58] E. Keyder and H. Geffner. Heuristics for planning with action costs revisited. In *Proc. ECAI-08*, pages 588–592, 2008.

[59] E. Keyder and H. Geffner. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36:547–556, 2009.

[60] C. Knoblock. Learning abstraction hierarchies for problem solving. In *Proc. AAAI-90*, pages 923–928, 1990.

[61] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proc. ECML-2006*, pages 282–293, 2006.

[62] A. Kolobov, Mausam, and D. Weld. LRTDP vs. UCT for online probabilistic planning. In *Proc. AAAI*, 2012.

[63] R. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

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

[65] R. Korf. Finding optimal solutions to Rubik's cube using pattern databases. In *Proc. AAAI-98*, pages 1202–1207, 1998.

[66] R. Larbi, S. Konieczny, and P. Marquis. Extending classical planning to the multi-agent case: A game-theoretic approach. In *Proc. 9th European Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2007)*, volume 4724 of *Lecture Notes in Computer Science*, pages 731–742. Springer, 2007.

[67] N. Lipovetzky and H. Geffner. Width and serialization of classical planning problems. In *Proc. ECAI*, pages 540–545. IOS Press, 2012.

[68] A. Martelli and U. Montanari. Additive AND/OR graphs. In *Proc. IJCAI-73*, pages 1–11, 1973.

[69] B. Marthi, S. Russell, and J. Wolfe. Ang2lic semantics for high-level actions. In *Proc. ICAPS-07*, pages 232–239, 2007.

[70] M. Martin and H. Geffner. Learning generalized policies from planning examples using concept languages. *Appl. Intelligence*, 20(1):9–19, 2004.

[71] M. J. Mataric. *The Robotics Primer.* MIT Press, 2007.

[72] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of AAAI-91*, pages 634–639, 1991.

[73] A. McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings Tenth Int. Conf. on Machine Learning*, pages 190–196, 1993.

[74] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. AIPS-96*, pages 142–149, 1996.

[75] D. McDermott. PDDL – the planning domain definition language. At `http://ftp.cs.yale.edu/pub/mcdermott`, 1998.

[76] S. McIlraith and R. Fadel. Planning with complex actions. In *Proc. NMR-02*, pages 356–364, 2002.

[77] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[78] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[79] A. Newell, J. C. Shaw, and H. Simon. Chess-playing programs and the problem of complexity. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 109–133. McGraw Hill, 1963.

[80] A. Newell and H. Simon. Elements of a theory of human problem solving. *Psychology Review*, 1958.

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

[82] K. Ogata. *Modern control engineering.* Prentice Hall, 2001.

[83] H. Palacios and H. Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.

[84] J. Pearl. *Heuristics.* Addison Wesley, 1983.

[85] J. Penberthy and D. Weld. UCPOP: A sound, complete, partiall order planner for ADL. In *Proc. KR-92*, 1992.

[86] M. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming.* John Wiley and Sons, Inc., 1994. 2

[87] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

[88] J. Rintanen. Heuristics for planning with SAT. In *Proc. on Principles and Practice of Constraint Programming (CP 2010)*, pages 414–428. Springer, 2010.

[89] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2009. 3rd Edition.

[90] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2):115–135, 1974.

[91] E. Sacerdoti. The nonlinear nature of plans. In *Proc. IJCAI-75*, pages 206–214, 1975.

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

[93] G. Shani and R. Brafman. Replanning in domains with partial information and sensing actions. In *Proc. IJCAI-2011*, pages 2021–2026, 2011.

[94] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2164–2172, 2010.

[95] H. Simon. A behavioral model of rational choice. *The quarterly journal of economics*, 69(1):99–118, 1955.

[96] K. Stanley, B. Bryant, and R. Miikkulainen. Real-time neuroevolution in the nero video game. *Evolutionary Computation, IEEE Transactions on*, 9(6):653–668, 2005.

[97] R. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[98] R. Sutton and A. Barto. *Introduction to Reinforcement Learning.* MIT Press, 1998.

[99] A. Tate. Generating project networks. In *Proc. IJCAI*, pages 888–893, 1977.

[100] J. Tooby and L. Cosmides. The psychological foundations of culture. In J. Barkow, L. Cosmides, and J. Tooby, editors, *The Adapted Mind.* Oxford, 1992.

[101] M. Toussaint and A. Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *Proc. 23rd Int. Conf. on Machine Learning*, pages 945–952, 2006.

[102] J. Veness, K. Ng, M. Hutter, W. Uther, and D. Silver. A Monte-Carlo AIXI approximation. *Journal of Artificial Intelligence Research*, 40(1):95–142, 2011.

[103] C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[104] T. Wilson. *Strangers to ourselves.* Belknap Press, 2002.

2

[105] S. Yoon, A. Fern, and R. Givan. FF-replan: A baseline for probabilistic planning. In *Proc. ICAPS-07*, pages 352–359, 2007.

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