An overview of recent algorithms for AI planning

Jussi Rintanen and Jörg Hoffmann
Albert-Ludwigs-Universität Freiburg, Institut für Informatik
Georges-Köhler-Allee, 79110 Freiburg im Breisgau
Germany

Abstract

During the past several years, AI planning has made major steps forward in terms of the size and difficulty of problems that can be solved. In this article we give an overview of the techniques that have been part of the recent developments. Instead of concentrating on individual planning systems, we review the underlying principles behind many of the successful planners. Most of the overview is devoted to classical planning, that is fully deterministic planning with one initial state, and we discuss more general forms of planning only briefly in the end of the article.

1 Introduction

Research on AI planning had concentrated on the so-called non-linear or partial-order planning algorithms (see for example [McAllester and Rosenblitt, 1991]) until the introduction of the Graphplan algorithm in 1995 by Blum and Furst [1997]. This algorithm had two characteristics that separated it from earlier ones: it finds plans of a fixed length (that is incrementally increased until a plan is found), and it uses reachability information for pruning the search tree. These differences brought the performance of Graphplan to a level not seen in connection with earlier planners.

The success of Graphplan led the research community to look at techniques outside the traditional AI planning toolbox. Soon after the introduction of Graphplan Kautz and Selman demonstrated that a general purpose satisfiability algorithm can in many cases outperform Graphplan and other algorithms specifically designed for AI planning [Kautz and Selman, 1996]. This motivated researchers to investigate translating planning to other computational problems and using solvers for them in finding plans. More recently, Bonet and Geffner's HSP planners [Bonet and Geffner, 2001], which use plain forward or backward chaining, have showed very good performance on many benchmark problems, and increased interest in heuristic search as a planning technique.

In this article we discuss the basic ideas related to some of the more interesting recent approaches to planning. First we discuss the constraint-based model that includes the Graphplan and the satisfiability planning approaches, among others (Section 2). We identify the explicit representation of fluent values at all time points as one of the properties of the recent work that separates it from pre-Graphplan planners. This representation makes it easy to combine the basic planning algorithm with different kinds of constraints that speed up plan search, for example reachability information and domain-specific control information (Section 3), and to handle parallel operator application efficiently.

The constraint-based model of classical planning allows plan search in several alternative ways. The main approaches have been backward chaining, as in the Graphplan algorithm, and general constraint solving, as exemplified by the satisfiability planning approach. The latter algorithms often need far less search than backward chaining, but there is a cost to pay: the amount of computation per search tree node is often much higher than in a backward chaining planner. Depending on the type of problem instance at hand, the reduction in the search tree size may dramatically outweight the more expensive computation, and in other types of problems, not much affect the search tree size and lead to an inferior performance. Of course, asymptotically as problem instances get bigger, any additional polynomial time computation that leads to a sufficient reduction in the depth or branching factor of the search trees is beneficial, but this might not show up in problems of the size that can be practically solved.

Distance heuristics have recently caught the attention of the research community, and we discuss some of the work in Section 4. The techniques compute approximate distances between states, which helps in operator selection. For many of the benchmark problems used by the research community, the heuristics give a very good estimate of the distances, and plan search even with a plain forward or backward chaining algorithm can be very efficient. On some of the commonly used benchmark problems, the problems that have been solved this way have been far bigger than those solvable for example with Graphplan or satisfiability planning. However, these heuristic planners have been most successful in finding non-optimal solutions to computationally easy (polynomial time) planning problems, and the situation may be different when optimal (shortest) solutions are needed and in connection with computationally more difficult problems (that are NP-hard or PSPACE-hard) that inherently require search.

A strand of research that is based on a different way of exploring state spaces uses ordered binary decision diagrams (OBDDs) and is discussed in Section 5. So far, OBDD-based

algorithms for classical planning have been rather good on certain benchmarks, but they also have had serious problems with memory consumption, which is an inherent problem of OBDDs. There may be planning problems where symbolic breadth-first search as used by OBDD-based planners would outperform other approaches to classical planning discussed in this article, but it seems that the main applications are in the more general forms of planning that are discussed in the last section of the article.

Section 6 concludes the overview by a brief discussion of algorithms for more general planning problems involving uncertainty and incomplete information. This kind of planning is needed when there are more than one initial state and when the operators and the environment are nondeterministic. Many of the recent techniques for classical planning can be generalized to this setting, but the algorithms have a distinctly different flavor as the notion of plans is more general than in classical planning.

2 Planning as a constraint satisfaction problem

Many of the recent planning algorithms and translational approaches to planning look for plans of a given length n that is increased step by step until a plan is found. Doing plan search in this way has several benefits. First, shortest plans (in terms of points of time) are found. Second, the descriptions of both the initial and the goal states can be used for effectively inferring fluent values at different time points, thereby reducing exhaustive search. In this section we formalize planning in this setting, as well as discuss different search techniques and ways of constraining the search problems further.

Plan operators are pairs $\langle p, e\rangle$ where p (the preconditions) and e (the effects) are sets of literals, that is atomic fluents or negations of atomic fluents. Sometimes p is restricted to atomic fluents. Fluents are often called state variables or facts, and they assume different values at different time points. An operator can be applied if its preconditions are true, and as a result the effects become true. More general definitions of plan operators can be given, for example with arbitrary formulae as preconditions, and so-called conditional effects in which the set of fluents that change is dependent on the truth of some formulae. Most planners take schematic plan operators (parameterized by the domain objects) as input, but transform them to sets of operators like described above.

2.1 Parallelism

An operator application in general means that the effects become true and other, unaffected fluents preserve their truth-values. However, it is not necessary to restrict to one operator application at a time: several operators can often be applied in parallel. This may reduce the planning effort substantially because separately considering $n!$ different (but behaviorally equivalent) orderings of n mutually independent operators is avoided.

The parallel application of operators is well-defined when the operators do not interact. This means that the result of applying the operators in any order is possible and has the same effect. A sufficient condition for this is that the operators do not have contradictory effects and none of the operators falsifies the preconditions of any other. Allowing parallel application still more liberally is possible, and this sometimes leads to even more efficient planning [Dimopoulos *et al.*, 1997].

2.2 Constraints on consecutive fluent values

From the above description of (parallel) operator application the following characterization of possible sequences of operator applications can be derived. For a given initial state, the application of a sequence $A_0, A_1, \ldots, A_{n-1}$ of sets of operators determines the fluent values at time points $0, \ldots, n$. This can be viewed as a model M that is a sequence of propositional models, one for each time point. The truth of a propositional formula P or a set of formulae P at t is denoted by $M \models_t P$. The application of operators $A_t = \{\langle p_1, e_1\rangle, \ldots, \langle p_m, e_m\rangle\}$ at t must satisfy the following.

1. $p_i \cup e_j$ is consistent for all $\{i, j\} \subseteq \{1, \ldots, m\}$ such that $i \neq j$.
2. $M \models_t p_i$ for all $i \in \{1, \ldots, m\}$.
3. $M \models_{t+1} e_i$ for all $i \in \{1, \ldots, m\}$.
4. If an atomic fluent f occurs nowhere in $e_1 \cup \cdots \cup e_m$, then $M \models_t f$ iff $M \models_{t+1} f$.

The planning problem can now be described as follows. Find a model M (and sets $A_0, \ldots, A_{n-1}$) that satisfies the conditions 1-4, and satisfies $M \models_0 I$ for the formula I describing the initial state, and satisfies $M \models_n G$ for the formula G describing the goals. There is such M if and only if there is a plan of length n.

Consider the problem of moving object A from room 1 to room 3 through room 2, and object B from room 2 to room 1. We assume that there are three points of time, and hence two sets of parallel operators can be applied respectively at 0 and 1. In Figure 1 on the left the description of the planning problem is shown, and on the right one of the two solutions with 3 time steps. The corresponding plan consists of operator move(A,R1,R2) at time 0 and operators move(A,R2,R3) and move(B,R2,R1) at time 1.

2.3 Search techniques

Planning proceeds by trying to find plans of length $n = 0, 1, 2, \ldots$ until a plan is a found. Several approaches to plan search have been proposed.

1. The Graphplan algorithm which uses backward chaining (regression) starting from the description of the goal states [Blum and Furst, 1997].

   This corresponds to selecting the operators in the order $A_{n-1}, A_{n-2}, \ldots, A_1, A_0$. To reduce the amount of search, Graphplan uses memoization: information from failed subgoals are recorded so that isomorphic subtrees of the search tree will not be traversed several times.

2. Translation to the satisfiability problem of the classical propositional logic [Kautz and Selman, 1996] (satisfiability planning.)

   The descriptions of the goal and the initial states as well as the conditions 1-4 are translated into propositional logic, and plans are found by a satisfiability algorithm.

	time t		
fluent	0	1	2
at(A,R1)	T		
at(A,R2)	F		
at(A,R3)	F		T
at(B,R1)	F		T
at(B,R2)	T		
at(B,R3)	F		
operators	0	1	
move(A,R1,R2)			
move(A,R2,R3)			
move(A,R3,R2)			
move(A,R2,R1)			
move(B,R1,R2)			
move(B,R2,R3)			
move(B,R3,R2)			
move(B,R2,R1)			

	time t		
fluent	0	1	2
at(A,R1)	T	F	F
at(A,R2)	F	T	F
at(A,R3)	F	F	T
at(B,R1)	F	F	T
at(B,R2)	T	T	F
at(B,R3)	F	F	F
operators	0	1	
move(A,R1,R2)	Y	N	
move(A,R2,R3)	N	Y	
move(A,R3,R2)	N	N	
move(A,R2,R1)	N	N	
move(B,R1,R2)	N	N	
move(B,R2,R3)	N	N	
move(B,R3,R2)	N	N	
move(B,R2,R1)	N	Y	

Figure 1: The representation of a planning problem in terms of the fluent values at different time points, and a solution to the problem that corresponds to a plan. Y in column t means that $o \in A_t$ and N that $o \notin A_t$.

3. Translation to other NP-hard computational problems.

   Several alternatives have been tried out, and the efficiency with best solvers is often at the same order of magnitude as with satisfiability algorithms. Some of these include integer programming [Vossen *et al.*, 1999; Kautz and Walser, 1999], mixed integer linear programming [Wolfman and Weld, 1999], nonmonotonic logic programming [Dimopoulos *et al.*, 1997], and CSPs [van Beek and Chen, 1999].

4. Direct solution by a specialized constraint solver [Rintanen, 1998]. In contrast to the translational approaches, planning-specific properties of the problem instances may be taken advantage of and memory consumption is much lower.

Finding plans in all the above approaches can be sped up by constraining the search problems further. Different types of declarative control information are discussed next.

3 Declarative control information

Many planning algorithms work by incrementally making a description of a plan more complete, and they backtrack when it turns out that the current incomplete plan cannot be extended to a full plan. It is critical to prune the search tree by reducing the number of consecutive backtracking points (the depth of the search tree) and the degree of backtracking points (branching factor). Linear reduction in either yields an exponential reduction in search tree size.

There are many types of declarative control information that help reducing search effort. Some control information, most notably invariants and planning graphs (Section 3.1), contribute to detecting the impossibility of extending an incomplete plan to a full plan, but do not affect the set of possible plans. Other forms of information, for example constraints derived from symmetries (Section 3.2) and domain-specific constraints on operator sequences (Section 3.3), reduce the set of possible plans, but do not make a solvable problem instance unsolvable.

3.1 Invariants and planning graphs

An important form of control information that can be derived from the operator definitions and the initial state is *invariants*. Invariants are formulae that are true in the initial state and are preserved by the application of every operator, and consequently they are true in all states that are reachable from the initial state.

Invariants are useful because they characterize the set of reachable states of the planning problem. If a (possibly incomplete) description of a state violates an invariant, it cannot be reachable from the initial state, and for example a backward chaining planner should reject that kind of subgoal. Similarly, an incomplete description M of states can be extended by using invariants: if $M \models_t a$ holds and $\neg a \vee b$ is an invariant, then $M \models_t b$ must hold.

The connection to reachability leads to iterative algorithms for computing invariants [Blum and Furst, 1997; Rintanen, 2000]. These algorithms iteratively compute sets of formulae that describe an upper bound of the sets of states that are reachable with an increasing number of operator applications from the initial state. When the upper bound changes no more, the formulae are invariants. Blum and Furst use the intermediate stages of the invariant computation for constructing planning graphs, which are useful in speeding up backward-chaining planning algorithms, like Graphplan.

3.2 Symmetries

Many planning problems involve interchangeable objects that cause state spaces to be highly symmetric. For example, a transportation problem with two identical vehicles A and B that are initially at the same location is symmetric with respect to A and B: for all plans the roles of A and B can be exchanged without affecting the correctness of the plans.

The existence of symmetries allows the reduction of search effort. If a planner has determined that solving a certain problem with vehicle A is not possible, it is not necessary to try

to solve the same problem with vehicle B, and thereby the amount of work is halved. The recognition of symmetries may exponentially reduce the search effort as the number of interchangeable objects increases.

Joslin and Roy [1997] give an algorithm that detects symmetries in schematic representations of planning problems. The algorithm is based on recognizing automorphisms in graphs. From the detected symmetries one can infer symmetry-breaking constraints that prevent redundant work.

3.3 Domain-dependent control rules

Domain-independent planners are often not capable of solving certain classes of problems efficiently, even when very effective domain-dependent techniques exist. Strategies for solving such problems can often be expressed as domain-dependent control rules that can be interpreted by a domain-independent planner. Early work on such control rules relied on procedural and planner dependent representations. Recently, Bacchus and Kabanza [2000] have advocated a declarative approach to control information that uses temporal logics.

In classical planning an initial state and a plan determine a sequence of truth-assignments to the fluents, which is essentially a semantic model in a linear temporal logic. A temporal logic formula can therefore be seen as expressing properties of the desired plans. If it is required that the formula is true in the model corresponding to a plan, the possible choices of operators can be restricted. This may speed up planning considerably.

Bacchus and Kabanza presented a procedure for interpreting temporal logic formulae within a simple forward-chaining planner. The progression procedure takes as input a state (the current time point) and a formula, and evaluates the formula with respect to the current fluent values (this evaluation is only partial because the truth of the formula in general depends on future fluent values), and derives a new formula that the next time point has to satisfy. This formula is used in selecting the next operator: if the state obtained by applying an operator cannot satisfy the formula, that operator is rejected.

A formula for a transportation problem could state that a vehicle must stay at a gas station until its tank is full.

$$(\text{at}(v, g) \wedge \text{gasstation}(g)) \rightarrow (\text{at}(v, g) \mathcal{U} \text{fulltank}(v))$$

Here $\mathcal{U}$ is the operator *until*. In a state in which the vehicle is at the gas station and its tank is not full, the progression algorithm produces the following formula.

$$\text{at}(v, g) \mathcal{U} \text{fulltank}(v)$$

When the planner considers an operator that takes the vehicle away from the gas station without filling the tank, the progression algorithm detects the violation of the formula, and the operator must be rejected.

4 Domain independent distance heuristics

The basic idea of heuristic search can be described as follows. Derive an estimation function, a heuristic, that assigns to each search state a value indicating how good that state is. Then, during search, favor those states that seem to be best. The difficulty in applying this to domain independent planning lies in the derivation of the heuristic.

The current techniques to derive heuristics for planning are based on the number of operators that are needed for reaching the goals. Solving this problem exactly is computationally very difficult, and the techniques are based on relaxing the problem P at hand into a simpler problem P' that can be solved efficiently. The solution can be used for estimating the difficulty of P.

A straightforward way to relax a planning problem is to ignore the negative effects of all operators. This relaxation has been proposed by Bonet et al. [1997]. Computing the optimal relaxed solution length, which would yield an admissible heuristic, is NP-hard [Bylander, 1994], so Bonet et al. introduced a method for approximating that length. Facing a search state S, initialize the *weights* of all fluents in S to 0, and that of all other fluents to ∞. Then apply all operators. For each operator with preconditions p that adds a fluent f, update the weight of f to

$$weight(f) := min(weight(f), weight(p) + 1).$$

Iterate the updates until the weight values of all fluents have reached their fixpoint. It is assumed here that operators have only positive preconditions. The weight of a set of fluents is defined as the sum of the individual weights. The difficulty of the state can be estimated as

$$h(S) := weight(\mathcal{G}).$$

Here, $\mathcal{G}$ denotes the goal state of the problem at hand. Bonet and Geffner use this estimation process in all versions of the HSP system [Bonet and Geffner, 2001]. A variation of the process, taking into account positive interactions between the fluents, has recently been proposed by Hoffmann [2000].

In difference to the above estimation processes, going forward from a state towards the goal, one can also estimate backwards, from the goal to the current state. Such approaches have been proposed by McDermott [1996] and Refanidis and Vlahavas [2000].

4.1 Planning with heuristics

Heuristic planners based on ignoring negative effects have achieved extremely competitive runtime behavior on a lot of the commonly used benchmark planning domains. At the AIPS-2000 planning systems competition, four out of five awarded fully automatic planners were based on, or at least incorporating, that approach [Bacchus and Nau, 2001]. All of those planners use the same naive search paradigm, state space search. Their success is apparently due to the quality of their heuristics on many of the current planning benchmarks. The open question is how one can develop heuristic search algorithms that work well on planning problems where ignoring negative effects yields estimates of less quality.

5 Planning with ordered binary decision diagrams

Ordered binary decision diagrams [Bryant, 1992] have been extensively used in computer-aided verification, especially in

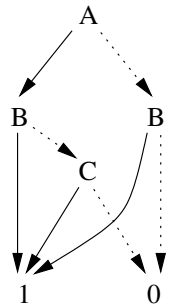

Figure 2: An OBDD

symbolic model-checking, that is formal verification of transition systems (a communication protocol, a circuit, a program) with respect to correctness properties. These properties are typically expressed in a temporal logic. If the transition system does not have the desired property, the model-checker attempts to produce a counterexample, which is a transition sequence that violates the property. Classical planning is equivalent to the model-checking (and counterexample generation) problem restricted to *safety* properties that say that states with a given undesired property should not be reachable. These states correspond to the goal states in planning, and the corresponding counterexamples (transition sequences) are plans. Techniques developed for model-checking are therefore directly applicable to AI planning, and vice versa.

Model-checking (and AI planning) can be performed by explicitly enumerating the state space and finding transition sequences by algorithms that traverse graphs. However, the numbers of states in transition systems are often high, and explicit enumeration of the state space is therefore often not possible. Algorithms that need to handle large state spaces have to represent them implicitly, for example as ordered binary decision diagrams [Bryant, 1992].

Ordered binary decision diagrams provide an efficient canonical representation of Boolean functions. Two OBDDs are logically equivalent if they are the same OBDD. The negation of an OBDD and the conjunction and disjunction of two OBDDs can be computed relatively efficiently. Because of these properties, OBDDs are applied in symbolic model-checking of transition systems and in verifying the equivalence of combinatory circuits. Even though OBDDs are often efficient, in more complex applications their size and memory requirements become prohibitively high, and this is their main drawback.

OBDDs are based on the ternary Boolean operator if-then-else $ite(p, \phi_1, \phi_2)$ defined as $(p \wedge \phi_1) \vee (\neg p \wedge \phi_2)$, where p is a propositional variable. Any Boolean formula can be represented by using this operator together with propositional variables and the constants true and false. Figure 2 depicts an OBDD for the formula $(A \vee B) \wedge (B \vee C)$. The normal arrow coming from a node for P corresponds to the case in which P is true, and the dotted arrow the case in which P is false.

5.1 Representation of transition systems as OBDDs

The usefulness of OBDDs in representing transition systems was discovered by MacMillan et al. at the Carnegie-Mellon University in the end of 1980's. The transition relation of the transition system and the sets of reachable states can be represented as OBDDs, and relevant operations on sets of states can be performed by straightforward OBDD manipulation [Burch *et al.*, 1994].

A transition relation expresses which combinations of truth-values are possible for the fluents in the predecessor and in the successor state. For example, the transition relation for a plan operator o_i with precondition $p_1 \wedge p_2 \wedge \neg p_3$ and effects $p_2, \neg p_3, p_4$ is represented by

$$\tau_i = \overbrace{p_1 \wedge p_2 \wedge \neg p_3}^{\text{precondition}} \wedge (p_1 \leftrightarrow p'_1) \wedge \overbrace{p'_2 \wedge \neg p'_3 \wedge p'_4}^{\text{effects}} \wedge (p_5 \leftrightarrow p'_5),$$

that is, the precondition and the effects are true, and fluents not occurring in the effects preserve their truth-values. Here $p_1, \ldots, p_5$ represent the fluents in the predecessor state and $p'_1, \ldots, p'_5$ represent the fluents in the successor state. Transition relation Θ for all n operators is $\tau_1 \vee \cdots \vee \tau_n$.

Given an initial state I that assigns a truth-value to every fluent or a description G of goal states that assigns truth-values to some of the fluents, we are interested in finding out which states are reachable from I and from which states the states G are reachable. For this the computation of *images* and *preimages* of sets of states under Θ are needed. Here we use preimage computations only. The preimage $\text{preimg}_\Theta(S)$ of states S under Θ is computed by first renaming variables p_i to p'_i in S to obtain S', and then computing $\exists p'_1 p'_2 \cdots p'_n (S' \wedge \Theta)$. Here $\exists p \Phi$ means $\Phi[T/p] \vee \Phi[F/p]$.

5.2 An OBDD-based planning algorithm

An OBDD-based planner can be easily constructed on the basis of image or preimage computation, and the resulting algorithm is a special case of algorithms for model-checking and construction of counterexamples [Burch *et al.*, 1994; Clarke *et al.*, 1994]. The traversal of the state space corresponds to breadth-first search, and there is the choice between forward traversal starting from the initial state I (by image computations) and backward traversal starting from the goal states G (by preimage computations). We discuss the backwards case only. Below, the sets D_i consist of states from which some state in G is reachable by i operator applications or less.

$$\begin{aligned} D_0 &= G \\ D_i &= \text{preimg}_\Theta(D_{i-1}) \cup D_{i-1}, \text{ for } i \geq 1 \end{aligned}$$

The computation is terminated when $I \in D_i$ for some i. Now i is the number of operators needed for reaching G from I, and a plan is a sequence of operators each of which reaches a state that is one step closer to a goal state.

$s := I$
for $j := i - 1$ **to** 0 **do**
  output $o \in O$ such that applying o in s yields $s' \in D_j$
  $s := s'$

end for

Planners based on variations of state-space exploration by OBDDs as described above have been presented by several researchers. The first one was by Cimatti et al. [1997].

6 Conditional and probabilistic planning

The classical planning problem considered in this overview so far is a special case of the problem of planning under uncertainty and incomplete information, often called *conditional planning* or *probabilistic planning*. The former term sometimes refers only to the special case in which the exact probabilities of possible events are ignored.

The more general problem arises because the world in which the plans are executed is not completely known or cannot be completely observed, and events taking place during plan execution may be nondeterministic. Because of these properties, it is not in general possible to determine beforehand one single sequence of operators that would always reach the goals. Instead, the plans have to choose the operators during execution based on the observations concerning the events that actually have taken place. Such plans explicitly assign an operator to every possible state, or take the form of a program in a simple programming language with operators as atomic statements and conditional and loop statements, or, equivalently, as finite automata.

Many of the recent algorithms for conditional and probabilistic planning are generalizations of algorithms for classical planning that we have already described. In this section we briefly discuss some of the main approaches.

6.1 Planning with ordered binary decision diagrams

OBDDs have successfully been used for classical planning, as discussed in Section 5, but their benefits fully show up when a lot of uncertainty and incompleteness is involved.

Cimatti et al. [1998] generalize their earlier work [Cimatti *et al.*, 1997] to universal/reactive/conditional planning with several initial states, nondeterministic operators and environment. This works as follows. Starting from the goal states, the sets of states from which a goal state is reachable with $n \geq 0$ steps or less are computed. When for some n the set includes all initial states, a plan has been found. During the computation, each state is associated with an operation that is along a shortest path to a goal state. Plans are executed by repeatedly observing the current state and executing the operator associated with the state.

In the work by Cimatti et al. discussed above, full observability was assumed. Another extreme is that none of the fluents are observable. This has been called *conformant planning*. Now the plans are sequences of operators, just like in classical planning, because acting differently in different situations is not possible. Cimatti and Roveri [2000] propose an OBDD-based approach to conformant planning. Plan construction is based on representing sets of sets of possible current states (sets of belief states) as OBDDs. Each operator maps belief states to belief states, and plans are found by backward chaining from the goal states.

The Markov decision processes [Puterman, 1994] model of sequential decision making generalizes many types of AI planning. A main difference between most of the work in that area and in AI planning is that AI planners do not represent the transition relations associated with state spaces explicitly. Recently there have been works that combine MDPs with the representation of the underlying transition systems as plan operators. For example, Hoey et al. [1999] perform value iteration on Markov decision processes that are represented as algebraic decision diagrams, which is a generalization of ordered binary decision diagrams.

6.2 Heuristic search

Bonet and Geffner [2000] view conditional and probabilistic planning as heuristic search in the belief space. Each belief state corresponds to a set of states of the underlying planning problem. Plans are found by algorithms like A$*$ and real time dynamic programming. State spaces are represented explicitly.

6.3 Constraint-based planners

The Graphplan algorithm has been generalized to conformant planning by Smith and Weld [1998]. The algorithm constructs a planning graph for every initial state, and during plan search the subgoals are compared to every planning graph. This guarantees the correctness of the plan for every initial state.

There are also generalizations of Kautz and Selman's satisfiability planning to conditional and probabilistic planning. Majercik and Littman [1999] translate probabilistic planning into stochastic satisfiability (SSAT), which is an extension of the propositional satisfiability problem to randomized (probabilistic) quantifiers. Rintanen [1999] translates conditional planning into quantified Boolean formulae (QBF). The planning problems that have been solved by using SSAT/QBF have so far been relatively small.

7 Conclusions

We have discussed a number of techniques that have proved to be useful for AI planning. Many of the developments in the research area are orthogonal to each other, and different successful planners have addressed different aspects in plan search. For example in the context of Graphplan and satisfiability planning, general purpose inference techniques were emphasized. Recent planners that use distance heuristics dispense with the inference aspect almost completely, rely on plain forward or backward chaining, and do not use reachability or other techniques for pruning search trees. These approaches have strengths in different types of problems, and a main challenge in the research field is to discover algorithms that show a strong performance on a wider range of problems than any of the current algorithms.

Research on algorithms for conditional and probabilistic planning has also accelerated during the last decade. This research area has close connections to problems addressed in other areas in computer science, such as program synthesis, as well as to operations research and control theory.

Acknowledgements

We thank Bernhard Nebel for many useful comments on an early version of this overview.

References

[Bacchus and Kabanza, 2000] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1–2):123–191, 2000.

[Bacchus and Nau, 2001] Fahiem Bacchus and Dana Nau. The AIPS-2000 planning systems competition. *AI Magazine*, 2001. to appear.

[Blum and Furst, 1997] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[Bonet and Geffner, 2000] Blai Bonet and Héctor Geffner. Planning with incomplete information as heuristic search in belief space. In Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 52–61. AAAI Press, 2000.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 2001. to appear.

[Bonet *et al.*, 1997] Blai Bonet, Gábor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pages 714–719, Menlo Park, California, July 1997. AAAI Press.

[Bryant, 1992] R. E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[Burch *et al.*, 1994] J. R. Burch, E. M. Clarke, D. E. Long, K. L. MacMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, April 1994.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[Cimatti and Roveri, 2000] Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[Cimatti *et al.*, 1997] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: a decision procedure for $\mathcal{AR}$. In S. Steel and R. Alami, editors, *Recent Advances in AI Planning. Fourth European Conference on Planning (ECP'97)*, number 1348 in Lecture Notes in Computer Science, pages 130–142. Springer-Verlag, 1997.

[Cimatti *et al.*, 1998] Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) and the Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 875–881. AAAI Press, 1998.

[Clarke *et al.*, 1994] Edmund Clarke, Orna Grumberg, Kenneth McMillan, and Xudong Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. Technical Report CS-94-204, Carnegie Mellon University, School of Computer Science, October 1994.

[Dimopoulos *et al.*, 1997] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in nonmonotonic logic programs. In S. Steel and R. Alami, editors, *Recent Advances in AI Planning. Fourth European Conference on Planning (ECP'97)*, number 1348 in Lecture Notes in Computer Science, pages 169–181. Springer-Verlag, 1997.

[Hoey *et al.*, 1999] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In Kathryn B. Laskey and Henri Prade, editors, *Uncertainty in Artificial Intelligence, Proceedings of the Fifteenth Conference*, pages 279–288. Morgan Kaufmann Publishers, 1999.

[Hoffmann, 2000] Jörg Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In Zbigniew W. Raś and Setsuo Ohsuga, editors, *Foundations of Intelligent Systems, 12th International Symposium, ISMIS'00*, number 1932 in Lecture Notes in Artificial Intelligence, pages 216–227. Springer-Verlag, 2000.

[Joslin and Roy, 1997] David Joslin and Amitabha Roy. Exploiting symmetry in lifted CSPs. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pages 197–202, Menlo Park, July 1997. AAAI Press.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park, California, August 1996. AAAI Press.

[Kautz and Walser, 1999] Henry Kautz and Joachim Walser. State-space planning by integer optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99) and the Eleventh Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 526–533. AAAI Press, 1999.

[Majercik and Littman, 1999] Stephen M. Majercik and Michael L. Littman. Contingent planning under uncertainty via probabilistic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*

(AAAI-99) and the Eleventh Conference on Innovative Applications of Artificial Intelligence (IAAI-99), pages 549–556. AAAI Press, 1999.

[McAllester and Rosenblitt, 1991] David A. McAllester and David Rosenblitt. Systematic nonlinear planning. In T. L. Dean and K. McKeown, editors, *Proceedings of the 9th National Conference on Artificial Intelligence*, volume 2, pages 634–639. AAAI Press / The MIT Press, 1991.

[McDermott, 1996] Drew McDermott. A heuristic estimator for means-ends analysis in planning. In Brian Drabble, editor, *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, pages 142–149. AAAI Press, 1996.

[Puterman, 1994] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

[Refanidis and Vlahavas, 2000] Ioannis Refanidis and Ioannis Vlahavas. GRT: a domain independent heuristic for STRIPS worlds based on greedy regression tables. In Susanne Biundo and Maria Fox, editors, *Recent Advances in AI Planning. Fifth European Conference on Planning (ECP'99)*, number 1809 in Lecture Notes in Artificial Intelligence, pages 347–359. Springer-Verlag, 2000.

[Rintanen, 1998] Jussi Rintanen. A planning algorithm not based on directional search. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, pages 617–624. Morgan Kaufmann Publishers, June 1998.

[Rintanen, 1999] Jussi Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[Rintanen, 2000] Jussi Rintanen. An iterative algorithm for synthesizing invariants. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000) and the Twelfth Conference on Innovative Applications of Artificial Intelligence (IAAI-2000)*, pages 806–811. AAAI Press, 2000.

[Smith and Weld, 1998] David E. Smith and Daniel S. Weld. Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) and the Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 889–896. AAAI Press, 1998.

[van Beek and Chen, 1999] Peter van Beek and Xinguang Chen. CPlan: A constraint programming approach to planning. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99) and the Eleventh Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 585–590. AAAI Press, 1999.

[Vossen *et al.*, 1999] Thomas Vossen, Michael Ball, Amnon Lotem, and Dana Nau. On the use of integer programming models in AI planning. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, volume I, pages 304–309. Morgan Kaufmann Publishers, 1999.

[Wolfman and Weld, 1999] Steven A. Wolfman and Daniel S. Weld. The LPSAT engine & its application to resource planning. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, volume I, pages 310–315. Morgan Kaufmann Publishers, 1999.

Jussi Rintanen (born in 1968 in Tampere, Finland) studied computer science at the Helsinki University of Technology from 1987 until 1996, spent the year 1995-96 at the University of Texas at Austin, and got his Ph.D. degree from the Helsinki University of Technology in January 1997. He then worked as a project researcher at the University of Ulm until joining the AI department of Albert-Ludwidgs-Universität Freiburg in October 1999 as a scientific assistant. His research interests include applications of logic in computer science, especially in planning.

Born 1971 in Heidelberg, Jörg Hoffmann received a diploma in computer science from Freiburg University in 1999. From April 1999 to Dezember 2000, he was a member of the graduate school on "Human and Machine Intelligence" in Freiburg. He is now working on a project on planning as heuristic search, financed by DFG.