C-MAXPLAN: Contingent Planning in the MAXPLAN Framework

Stephen M. Majercik

Department of Computer Science
Duke University
Durham, NC 27708-0129
majercik@cs.duke.edu

Abstract

C-MAXPLAN extends MAXPLAN to include contingent planning in probabilistic propositional domains. Like MAXPLAN, C-MAXPLAN converts the planning problem into an E-MAJSAT instance, a type of probabilistic satisfiability problem in which a set of Boolean variables encodes the plan and a second set of Boolean variables encodes the probabilistic outcome of the plan—the satisfiability problem is to find the setting of the plan variables that maximizes the probability of satisfaction with respect to the outcome variables. We describe two different encodings for contingent plans in this framework: the first one explicitly encodes contingent actions and is similar to the approach taken by C-BURIDAN; the second encodes a *policy* rather than a plan. Although the efficiency with which the resulting satisfiability problem is solved depends critically on the contingent-plan representation, C-MAXPLAN is competitive with state-of-the-art contingent planners on some problems.

Introduction

A *contingent plan* is a plan in which action choices are made contingent on information about the state of the world. In our work, we are concerned with probabilistic propositional planning problems: states are represented as an assignment to a set of Boolean state variables (fluents) and actions map states to states probabilistically. Problems are expressed using a dynamic-belief-network representation. A subset of the state variables are declared *observable*, meaning that a plan can be made contingent on any of these variables. This scheme is sufficiently expressive to allow a domain designer to make a domain fully observable, unobservable, or to have observations depend on actions and states in probabilistic ways.

In this paper, we present C-MAXPLAN, a MAXPLAN-based approach to contingent planning. In C-MAXPLAN, a set of Boolean variables (the choice variables) encodes the contingent plan and a second set (the chance variables) the probabilistic outcome of the plan—the satisfiability problem is to find the setting of the choice variables that maximizes the probability of satisfaction with respect to the chance variables. This is equivalent to the E-MAJSAT problem (Littman, Goldsmith, & Mundhenk 1998).

In the remainder of this section, we describe our domain representation and some related work in the area of contingent planning. In the next section, we describe the operation of C-MAXPLAN. We describe two ways of mapping a contingent planning problem to an E-MAJSAT instance, and present some results. Finally, we conclude with some comments on further work, including a brief description of a promising, completely different approach to contingent planning in the probabilistic satisfiability framework.

Probabilistic Planning Representation

C-MAXPLAN works on partially observable probabilistic propositional planning domains. Such a domain consists of a finite set P of n distinct *propositions*, any of which may be `True` or `False` at any (discrete) time t. A *state* is an assignment of truth values to P. A probabilistic initial state is specified by a set of decision trees, one for each proposition. A proposition p whose initial assignment is independent of all other propositions has a tree consisting of a single node labeled by the probability with which p will be `True` at time 0. A proposition q whose initial assignment is not independent has a decision tree whose nodes are labeled by the propositions which q depends on and whose leaves specify the probability with which q will be `True` at time 0. *Goal states* are specified by a partial assignment G to the set of propositions; any state that extends G is considered to be a goal state.

Each of a set A of *actions* probabilistically transforms a state at time t into a state at time $t+1$ and so induces a probability distribution over the set of all states. In this work, the effect of each action on each proposition is represented as a separate decision tree (Boutilier & Poole 1996). For a given action a, each of the decision trees for the different propositions are ordered, so the decision tree for one proposition can refer to both the new and old values of previous propositions. The leaves of a decision tree describe how the associated proposition changes as a function of the state and action.

A subset of the set of propositions is the set of *observable propositions*. Each observable proposition has,

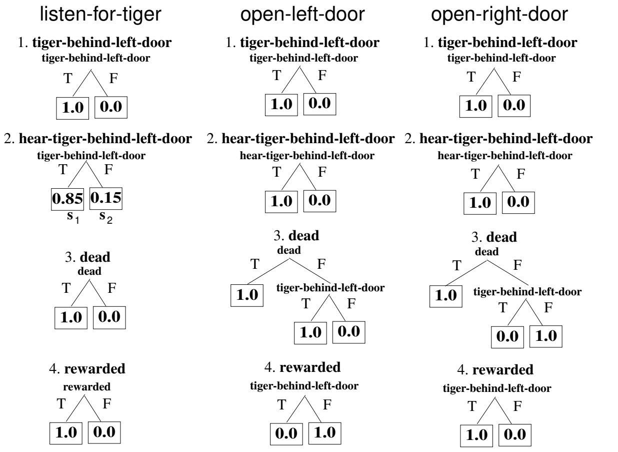

Figure 1: The effects of the actions in the TIGER problem are represented by a set of decision trees.

as its basis, a proposition which represents the actual status of the the thing being observed. (Note that although values are assigned to observable propositions in the initial state, no action at time 1 makes use of these propositions in its decision trees, since there are no valid observations at time 0.)

The planning task is to find a plan that selects an action for each step t as a function of the value of observable propositions for steps before t. We want to find a plan that maximizes (or exceeds a user-specified threshold for) the probability of reaching a goal state.

For example, consider a simple domain based on the TIGER problem of Kaelbling, Littman, & Cassandra (1998). The domain consists of four propositions: tiger-behind-left-door, dead, rewarded and hear-tiger-behind-left-door, the last of which is observable. In the initial state, tiger-behind-left-door is `True` with probability 0.5, dead is `False`, rewarded is `False`, and hear-tiger-behind-left-door is `False` (although irrelevant). The goal states are specified by the partial assignment (rewarded, (not dead)). The three actions are **listen-for-tiger**, **open-left-door**, and **open-right-door** (Figure 1). Actions **open-left-door** and **open-right-door** make reward `True`, as long as the tiger is not behind that door (we assume the tiger is behind the right door if tiger-behind-left-door is `False`). Since tiger-behind-left-door is not observable, the **listen** action becomes important; it causes the observable hear-tiger-behind-left-door proposition to become equal to tiger-behind-left-door with probability 0.85 (and its negation otherwise). By **listen**ing multiple times, it becomes possible to reliably determine the location of the tiger.

Related Work

The type of partially observable planning problem we address, featuring actions with probabilistic effects and noisy observations, is a form of partially observable Markov decision process (POMDP). Algorithms for POMDPs have been around for many years, and use a *flat* representation of domains; that is, every possible combination of state variables is considered independently. In this section, we focus on more recent algorithms that exploit propositional state representations. Of course, any algorithm that can solve a planning problem in a flat representation can also be used to solve a problem in the propositional representation by enumerating states; in fact, this approach is often the fastest for domains with up to five or six fluents.

One of the most well-known contingent planners for probabilistic domains is C-BURIDAN (Draper, Hanks, & Weld 1994), which uses tree-based, probabilistic STRIPS operators to extend partial-order planning to stochastic domains. C-BURIDAN searches for a type of contingent plan whose probability of success meets or exceeds some prespecified threshold. Onder & Pollack (1997) point out that there are some problems with C-BURIDAN, and these could prevent it from solving arbitrary partially observable planning problems. Specifically, C-BURIDAN:

- automatically constructs branches to resolve otherwise unresolvable conflicts without reasoning about whether the branch is *worth* constructing;
- creates a branch to resolve the conflict by inserting an observation step *before* the two conflicting actions, even if inserting the observation step *between* the two actions would be better;
- produces *observation labels* not related to actual propositions, so it needs to try all possible combinations of observation labels and action contexts in order to determine which actions could be triggered.

MAHINUR (Onder & Pollack 1997) is a probabilistic partial-order planner that combines C-BURIDAN's probabilistic action representation and system for managing these actions with a CNLP-style approach to handling contingencies. The novel feature of MAHINUR is that it identifies those contingencies whose failure would have the greatest negative impact on the plan's success and focuses its planning efforts on generating plan branches to deal with those contingencies. Onder & Pollack (1997) identify several domain assumptions (including a type of subgoal decomposability) that underlie the design of MAHINUR. There are no guarantees on the correctness of MAHINUR for domains in which these assumptions are violated.

Several researchers have extended GRAPHPLAN to handle uncertainty. CONFORMANT GRAPHPLAN (Smith & Weld 1998) deals with uncertainty in the initial conditions and in the outcome of actions by attempting to construct a nonsensing, noncontingent plan that will succeed in all cases. PGRAPHPLAN (Blum & Langford 1998) employs forward search through the planning graph to find the contingent plan with the highest expected utility in a completely observable stochastic environment. SENSORY GRAPHPLAN (SGP) (Weld, Anderson, & Smith 1998), unlike CONFORMANT GRAPHPLAN, constructs plans with sensing actions that gather information to be used later in distinguishing between different plan branches. Thus, SGP is an approach to constructing contingent plans. However, SGP has not been extended to handle actions with uncertain effects (except in the conformant case) and imperfect observations, so it is applicable to a subset of partially observable planning problems.

Boutilier & Poole (1996) describe an algorithm for solving partially observable planning problems based on an earlier algorithm for completely observable problems. While promising, little computational experience with this algorithm is available.

C-MAXPLAN: E-MAJSAT

MAXPLAN (Majercik & Littman 1998a; 1998b) was initially developed to solve probabilistic planning problems in completely unobservable domains. The MAXPLAN approach solves a planning problem by first converting the planning problem to an E-MAJSAT problem (Littman, Goldsmith, & Mundhenk 1998).

MAXPLAN solves the E-MAJSAT problem using a modified version of the Davis-Putnam-Logemann-Loveland (DPLL) procedure for determining satisfiability. Essentially, it uses DPLL to determine all possible satisfying assignments, sums the probabilities of the satisfying assignments for each possible choice-variable assignment, and then returns the choice-variable assignment (plan) with the highest probability of producing a satisfying assignment (goal satisfaction). The algorithm uses an efficient splitting heuristic (*time-ordered splitting*), *memoization*, and *caching* (Majercik & Littman 1998a; 1998b) in order to accelerate this procedure.

Although MAXPLAN was initialy developed to solve probabilistic planning problems in completely unobservable domains, it can solve contingent planning problems given an appropriate problem encoding. This highlights the benefit of separating the planning problem into two modules—problem conversion and problem solution. Supporting contingent planning requires only that the E-MAJSAT conversion unit allow the encoding of contingent actions.

More specifically, plans in C-MAXPLAN are encoded in a manner similar to that of C-BURIDAN (Draper, Hanks, & Weld 1994). A contingent action is expressed as a sequence of actions, each one of which executes only if its set of conditions matches the current set of observation labels generated by previous actions. Since the condition sets of the actions are mutually exclusive, the net result is that at most one contingent action will execute, depending on current conditions.

Encoding contingent plans in C-MAXPLAN thus requires a change of perspective. C-MAXPLAN still considers actions sequentially but, since a series of actions can now be interpreted as a single contingent action for a single time step, there is no longer a one-to-one correspondence between actions and time steps. It is now more appropriate to speak of an *n-action* plan rather than an *n-time-step* plan. In order to encode contingent plans, we need additional variables and clauses, and we need to alter the decision trees of the actions. In the following two sections, we describe the new variables and clauses required for a single *action* step of the plan, with the understanding that the variables and clauses required for the always-noncontingent initial action are somewhat different. Following that, we will describe the necessary alterations to the action decision trees.

Variables

A subset of the propositions in our planning domain is designated as the set of observable propositions; these are the propositions on which we wish to condition actions. Each one of these propositions requires:

- two variables indicating the status of our knowledge of the observable proposition; a knowledge variable indicating whether we have observed the status of the observable proposition and a perception variable indicating the agent's perception of the observable proposition,
- an action (possibly an existing one) that sets the knowledge and perception variables,
- three variables indicating the status of our conditioning on the perception variable; either we want the perception to be `True` or we want it to be `False` or we `Don't-Care`, and
- a single variable indicating whether the condition attached to that perception variable has been satisfied.

Finally, for each action step there is a single variable that indicates whether all the conditions specified for that action step were fulfilled.

Clauses

There are two necessary groups of clauses:

- We need a group of clauses that enforces mutual exclusivity of the possible desired conditions for a perception variable; for each perception variable, we want exactly one of the following to be true: the condition requires that the agent perceive the proposition to be `True`, the condition requires that the agent perceive the proposition to be `False`, or the condition specifies that the agent's perception of the proposition is irrelevant (`Don't-Care`).
- For each possible status (`True`/`False`/`Don't-Care`) of each perception upon which we are conditioning, we need a clause that describes what constitutes satisfaction of that condition; e.g. if condition-1 requires that proposition-2 be `False` and the knowledge variable indicating whether the agent has observed proposition-2 is `True` and the perception variable for proposition-2 is `False`, *then* condition-1 is satisfied.

The required alterations of the decision trees are straightforward. Of course, we need new decision trees in each action describing that action's impact on the new variables. More importantly, we need to alter the existing decision trees such that the action has no impact at all if the all-conditions-satisfied variable is not `True` for that action step.

Although C-MAXPLAN's encoding technique is similar to that of C-BURIDAN, the C-MAXPLAN solution technique has some advantages.

- C-MAXPLAN does not automatically insert a conditional branch under certain circumstances as C-BURIDAN does; solving the E-MAJSAT formula requires C-MAXPLAN to reason about where an observation and conditional branch would be worthwhile.
- C-MAXPLAN can insert a conditional branch anywhere in the plan; it is not restricted to inserting observations and conditional branches immediately before conflicting actions, as C-BURIDAN is.
- In C-MAXPLAN the result of an observation *is* a proposition, so the algorithm only has to check the status of the propositions generated by observation actions to see if an action's conditions are satisfied, unlike C-BURIDAN, which has to try matching an action's conditions against all observation labels that have been generated so far.

Solutions using this encoding are a series of grouped action steps with attached conditions. A group can describe either a noncontingent action or a contingent action. A noncontingent action group contains a single action step, all of whose conditions are `Don't-Care`. A contingent action group contains a set of action steps and attached conditions. These condition sets are mutually exclusive; hence, only one action in the group will be effective when the plan is executed. For example, the optimal plan in the 2-step TIGER problem is to **listen-for-tiger**, then **open-right-door** if hear-tiger-behind-left-door is `True`; otherwise **open-left-door**. The noncontingent first action, **listen-for-tiger**, would be in a group by itself with `Don't-Care`-about-hear-tiger-behind-left-door as its condition. The actions **open-right-door** and **open-left-door** would be in a second action group, with conditions hear-tiger-behind-left-door-is-`True` and hear-tiger-behind-left-door-is-`False` respectively.

Initial tests of this contingent planning technique were promising. Even the most basic version of C-MAXPLAN (without memoization) could solve the 2-step TIGER problem in 0.20 cpu seconds, a contingent version of the SAND-CASTLE-67 problem in 3.5 cpu seconds, the SHIP-REJECT problem described by Draper, Hanks, & Weld (1994) in 5.25 cpu seconds (compared to 0.12 cpu seconds for MAHINUR (Onder & Pollack 1997)), and the MEDICAL-1ILL problem described by Weld, Anderson, & Smith (1998) in 0.5 cpu seconds (compared to 0.12 cpu seconds for SGP (Weld, Anderson, & Smith 1998)). (All tests were run on a 300 MHz UltraSparcIIi.) Tests on the MEDICAL-4ILL problem described by Weld, Anderson, & Smith (1998), however, were disappointing. Even using memoization C-MAXPLAN had not solved the problem after several *days*.

We tried to improve this performance by using a *thresholding* technique. If, instead of insisting on the plan with optimal probability, we supply a probability threshold, we can sometimes prune partial plans based on the probability calculated so far and the initial threshold. For choice variables, only upper bound thresholding is possible. If the probability of success given an assignment of `True` to this variable is higher than our threshold, we can prune all plans in which this variable would be assigned `False`. For chance variables, we can perform both upper bound pruning and lower bound pruning. The upper bound pruning is similar to that used for choice variables, although the thresholds passed down the calculation tree must be appropriately adjusted. Lower bound pruning makes use of the fact that a sufficiently *low* probability of success on the first truth assignment checked ensures failure. If the probability of success given the first assignment to the chance variable is low enough, we can determine that the probability weighted average of both truth assignments could not possibly meet our adjusted threshold and we can return failure without calculating the probability of success of the other truth assignment.

Thresholding failed to reduce C-MAXPLAN's running time on the MEDICAL-4ILL problem to reasonable levels. A different approach to plan encoding, however, did yield a significant improvement in performance. Instead of searching for the optimal plan of a given length, we search for an optimal small *policy* to be applied for a given number of steps. In this approach, the decision trees from all actions for each proposition p are merged into a single decision tree that describes the impact of *all* the actions on p via a cascade of condition-fulfillment variables. Essentially, the decision tree says: "If the conditions specified by the policy for action a are satisfied, then decide the status of p according to a's decision tree; otherwise, if the conditions for action b are satisfied, then decide the status of p according to b's decision tree; $\ldots$; otherwise, the status of p remains the same."

In this encoding, we have a single action that is always taken—*follow-the-policy*—and the choice variables in the encoding describe the conditions under which an action should be executed. The setting of these condition-specification variables describes a policy to follow, and the algorithm searches for the setting of these variables that maximizes the probability of satisfying the E-MAJSAT formula (achieving the goals).

The primary advantage of this approach appears to be a more compact encoding of the problem. For example, suppose there is a proposition p whose value is changed only by action a, one of n actions. In C-MAXPLAN, there will be $2(n-1)+c$ clauses for each time step describing what happens to p, where c is the number of clauses needed to describe the impact of a on p. In the policy approach, the decision trees from all actions for each proposition are merged. In the above case, this would result in a decision tree that specifies the impact of a on p with a default branch of "no change." This would produce only $2+c$ clauses. Frequently, many propositions in a planning problem are changed by only a small number of actions, so the reduction in cnumber of clauses can be substantial.

Solutions using this encoding specify a policy to follow; that is, a set of conditions for each possible action, specifying when that action should be executed. Note that there may be more than one action that can be executed for a given set of conditions; such actions may be executed in parallel or in an arbitrary order. (Harmful interactions will have been accounted for in the solution process by reducing the probability of success.)

C-MAXPLAN, with this new encoding, no memoization, and thresholding, was able to solve the MEDICAL-4ILL problem in approximately 4900 cpu seconds. We improved this performance further by adapting the splitting heuristic described by Bayardo & Schrag (1997) for use in C-MAXPLAN. This heuristic selects an initial pool of candidates based on a score that rewards variables that appear both negated and not negated in binary clauses. This initial pool is rescored based on the number of unit propagations that occur for each assignment to each variable, rewarding variables for which both truth values induce unit propagations. Essentially, this heuristic tries to find a variable that will induce the highest number of unit propagations.

Preliminary results using policy-style encodings with thresholding and the new splitting heuristic were promising; the solution time for the MEDICAL-4ILL problem was reduced from 4900 cpu seconds to approximately 100 cpu seconds (compared to approximately 45 cpu seconds for SGP (Weld, Anderson, & Smith 1998)).

Further Work and Conclusion

C-MAXPLAN, like MAXPLAN, must consider every possible satisfying assignment in order to find the optimal plan. Except for small problems, this is an intractable search. The techniques we have used so far to make the search feasible have improved C-MAXPLAN's performance significantly, but they are not sufficient to allow C-MAXPLAN to scale up to large problems. To address this problem, we have begun investigating approximation techniques for C-MAXPLAN.

VOTING is an approximation technique we initially devised for MAXPLAN but which could be applied to C-MAXPLAN. On a single round of VOTING, the algorithm assigns values to all the random variables according to the probabilities associated with them. This corresponds to finding a "likely" situation, given the uncertainty in the domain. The standard C-MAXPLAN algorithm is then used to find every setting of the existential variables that would yield a satisfying assignment. This corresponds to finding all the plans that would achieve the stated goals given the likely situation previously generated. Each such plan receives a vote. After a specified number of rounds, the plan with the most votes is deemed to be the best plan. Finally, standard C-MAXPLAN is used to solve the simpler problem of calculating the exact probability of success of that plan. An initial implementation of this technique indicates that it is feasible, but definitive results are not yet available.

Although the results we obtained with the policy-style encoding of the contingent planning problem are encouraging, there are issues that make this approach problematic. For example, if two actions could be triggered by the same set of conditions, only the first one in the decision-tree cascade will be triggered, so the construction of the decision tree introduces unwanted bias.

Problems in which actions need to be conditioned on an entire *history* of observations present another difficulty. The 4-step TIGER problem is an example. The optimal contingent plan is to **listen-for-tiger** twice. If the observations resulting from these actions agree, take the appropriate action; otherwise, **listen-for-tiger** a third time and then take the action dictated by the majority vote of the three observations. While C-MAXPLAN is capable of encoding such a plan, the size of the encoding can grow exponentially with the length of the history one is conditioning on.

Although our results using C-MAXPLAN with policy-style encodings justify further research, there is another, quite different approach that appears more promising. In this approach, called ZANDER, we formulate the problem as an S-SAT instance, a type of probabilistic satisfiability problem in which groups of existentially and randomly quantified variables alternate. The resulting S-SAT problem is solved by carefully controlling the order in which we consider variables in the solution process, essentially considering variables representing observations before variables representing the actions that may be conditioned on these observations. The solution is a tree, each branch of which describes a plan execution history in the optimal contingent plan. Initial experiments suggest that ZANDER significantly outperforms C-MAXPLAN; details are reported elsewhere (Majercik & Littman 1999).

References

Bayardo, Jr., R. J., and Schrag, R. C. 1997. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 203–208. AAAI Press/The MIT Press.

Blum, A. L., and Langford, J. C. 1998. Probabilistic planning in the Graphplan framework. In *Working Notes of the Workshop on Planning as Combinatorial Search, held in conjunction with the Fourth International Conference on Artificial Intelligence Planning.*

Boutilier, C., and Poole, D. 1996. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1168–1175. AAAI Press/The MIT Press.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, 76–82.

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

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic plan existence and evaluation. *Journal of Artificial Intelligence Research* 9:1–36.

Majercik, S. M., and Littman, M. L. 1998a. MAXPLAN: A new approach to probabilistic planning. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning*, 86–93. AAAI Press.

Majercik, S. M., and Littman, M. L. 1998b. Using caching to solve larger probabilistic planning problems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 954–959. The AAAI Press/The MIT Press.

Majercik, S. M., and Littman, M. L. 1999. Contingent planning under uncertainty via probabilistic satisfiability. Submitted.

Onder, N., and Pollack, M. E. 1997. Contingency selection in plan generation. In *Proceedings of the Fourth European Conference on Planning*.

Smith, D. E., and Weld, D. S. 1998. Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 889–896. AAAI Press/The MIT Press.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending Graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 897–904. AAAI Press/The MIT Press.