Automated Conversion and Simplification of Plan Representations

Martin Allen and Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{mwallen, shlomo}@cs.umass.edu

Abstract

As planning agents grow more sophisticated, issues of plan *representation* arise alongside concerns with plan *generation*. Planning methods work over increasingly large and difficult problems and resulting plans are often complex or unwieldy. Further, where planners must interact with human beings—either for purposes of plan verification and analysis, or in *mixed-initiative* plan-generation settings—plans must be represented so that the intended course of action is readily visible. We propose automated techniques for the simplification of plans, and for conversion between distinct plan representations; our proposal is illustrated by examples from our recent research, concerning conversion between large-scale MDP solutions and graph-based contingency plans.

Introduction

With a few exceptions, planning research has concentrated upon finding plans that describe a correct or optimal course of action to some objective. Other aspects of the resulting plan have been somewhat neglected, such as compactness, the cost of storage or transmission, the ability to understand or explain the plan, or the ability to verify the plan or to recover from failure during execution. As planning algorithms improve in their ability to solve large, realistic problems, so grow the complexity and size of the resulting plans. Many existing techniques produce plans that are optimal, but that rank poorly in terms of compactness, clarity, verifiability, and ease of transmission to other decision-makers.

As an example, consider the use of *Markov decision processes* (MDPs) in recent work on planning. Developed originally within operations research and control theory, MDPs have been recently adopted by the AI community as a general framework for planning under uncertainty (Dean *et al.* 1995), and have proven useful in practice for such problems as mobile robot control, supply chain planning, forecasting, and equipment maintenance (Boutilier, Dean, & Hanks 1999). MDPs provide a powerful general model, along with well-understood and powerful algorithms for generating optimal solutions. The solution to an MDP is a *policy*, normally represented as a function from states in the environment to actions. From the perspective of a planning problem, such a mapping can in turn be seen as nothing other than a *complete plan*, dealing with all possible contingencies that might arise during execution. However, MDP techniques have the disadvantage that policies for complex environments can grow quite large, and prove difficult to represent in a compact or straightforward fashion.

For many purposes, however, plan representation is of the essence. NASA's semi-autonomous robotic space exploration program—particularly the Mars rover project—is an example. Existing rover designs have limited memory and processing power, and constrained bandwidth for purposes of communication with remote operators. These features, combined with the difficulty of accurate advance modelling of Martian surface conditions and the need for in-depth verification of plans prior to execution, mean that plans for rover operation must be represented compactly and without undue complexity (Bresina *et al.* 2002). Thus, MDP methods have been considered infeasible, and research has concentrated upon *limited contingency planning*, in which the planner adds progressively more branches to a basic plan, subject to constraints upon overall number of branch-points and through-paths. Such plans are generally sub-optimal, with effort directed towards identifying and planning for those contingencies deemed most likely (Dearden *et al.* 2003), of generating plans that are optimal only *relative to* a fixed degree of branching (Meuleau & Smith 2003). Similarly, work on *mixed-initiative* planning (Burstein & McDermott 1996), in which planners work in concert with human beings, require that interim plans be represented comprehensibly, allowing users to provide guidance.

The apparent gap between optimal probabilistic planning methods and more traditional planning representations and techniques poses a significant challenge for researchers. In order for automated planning to be useful, the plans produced must be of high quality. At the same time, the plans produced serve many purposes, and often must be judged based on more than the course of action they dictate. We propose a line of research into automated techniques for simplifying and converting between plan representations, to proceed alongside more conventional research on generating initial plans. We sketch here some work in this direction, and suggest further avenues for research on the questions raised.

Converting between Plan Representations

Our work so far has focussed on conversion between MDP solutions and graph-based contingency plans. In order to solve large planning problems efficiently, MDPs and their solutions are represented using *decision diagrams*. We have found that simple transformations effectively reduce the size of output policies, and convert them into relatively compact contingency plans. Besides advantages stemming from their small size, such plans allow for more straightforward analysis of expected behavior, better serving the needs of human users or plan-verification systems.

Decision diagrams and MDPs. Binary decision diagrams (BDDs) compactly represent Boolean functions (Bryant 1986). Each BDD is a directed acyclic graph with nodes representing variables and edges representing their values. Each path in the BDD terminates in an output node (0 or 1), giving the function's value for the variable assignment corresponding to that path. Although the worst-case size of a BDD is exponential in the number of variables—it may simply be a complete binary tree over those variables—many useful functions are susceptible to compact representations.

If the BDD is *ordered*, so variables occur in the same order on all paths, then any function has a *canonical representation*, relative to that order. The canonical form can be found by constructing the complete binary tree for the function and then recursively combining identical subtrees, removing nodes whose outgoing edges all lead to the same subtree, replacing them with a single edge to that subtree. Although the result is minimal relative to a fixed variable ordering, there is no guarantee that this ordering is the best possible. Indeed, finding the best variable ordering for representing a function is generally intractable. Still, many useful functions have compact and easily-found BDD forms; further, use of ordered BDDs allows many operations over the represented functions to be performed efficiently by combining basic operations over the diagrams themselves. In practice, ordered BDDs efficiently manipulate functions over very many variables, and prove useful in such areas as symbolic model-checking for circuit and system design.

Algebraic decision diagrams (ADDs) are straightforward extensions of BDDs, with real-valued outputs (Bahar *et al.* 1993). ADDs can be used to represent *factored MDPs*, in which states are assignments of values to variables. Since the cost of computing using ADDs is proportional to their size, MDPs with large state spaces can be solved efficiently if represented using relatively compact ADDs. For instance, the state-transition function for some action a can be calculated by an ADD D_a, in which each path represents two tuples of values V_1 and V_2, and that outputs the probability of moving from states given by V_1 to those given by V_2 under a. The elimination of unneeded variables by canonical minimization thus corresponds to using *state abstraction*. For example, eliminating variable v from a path in D_a is equivalent to treating the transition probability for action a as the same in all states identical in variables except v.

STATE	VARIABLES	ACTION
1	$\langle 0,0,0,0,0,0,0,0,0,0,0,0\rangle$	COLLECT
2	$\langle 0,0,0,0,0,0,0,0,0,0,0,1\rangle$	COLLECT
3	$\langle 0,0,0,0,0,0,0,0,0,0,1,0\rangle$	COLLECT
	⋮	
4094	$\langle 1,1,1,1,1,1,1,1,1,1,0,1\rangle$	LEFT
4095	$\langle 1,1,1,1,1,1,1,1,1,1,1,0\rangle$	COLLECT
4096	$\langle 1,1,1,1,1,1,1,1,1,1,1,1\rangle$	COLLECT

Table 1: A look-up table description of a policy for the rover problem MDP.

Solution techniques. The SPUDD planner first used ADDs to represent and solve MDPs by way of dynamic programming (Hoey *et al.* 1999). Our work uses Symbolic-LAO* (SLAO*), which also employs ADDs and is able to solve very large problem-instances (Feng & Hansen 2002). The algorithm extends LAO*, which combines heuristic search and dynamic programming to solve MDPs (Hansen & Zilberstein 2001). At each search step, LAO* expands states along the leading edge of the current policy are according to an admissible heuristic; dynamic programming is then used to evaluate all currently expanded states, and the policy is updated relative to the new evaluation. Since MDP policies may contain loops, LAO* performs these updates using iterative methods (e.g. policy iteration), and is guaranteed to terminate in an optimal policy. As a heuristic method, furthermore, it guides search away from states that an optimal policy never visits. Thus, in domains where policy exploration can ignore a large fraction of the overall states, LAO* performs significantly better than MDP algorithms that must update the value of every possible state.

SLAO* extends the idea using ADDs, allowing compact representation of large search-spaces. Furthermore, it exploits commonly-used routines for performing reachability analysis in order to restrict its attention to states that may actually be visited by the current policy. Policy expansion is guided by an admissible heuristic, computed quickly using an approximate algorithm. This heuristic element, and the efficiency reachability analysis is implemented efficiently for decision diagrams, makes SLAO* quite powerful. Experiments have shown it to outperform regular LAO* and SPUDD; it handles problems with over 2^{40} states, beyond the practical reach of either of the others.

Our work here is based upon the application of SLAO* to a version of the Mars Rover problem discussed in (Horstmann & Zilberstein 2003). Each instance involves a robotic rover-agent moving over a hill on which there a number of locations at which to collect samples, with varying rewards between them and a fixed time-window in which to operate. Movement uphill and downhill, as well as the collection of samples, varies according to a set of normal distributions. We investigate processes for automatically simplifying the policy generated by SLAO*, and for converting it automatically into a graph-based contingency plan.

Automated conversion of output plans. SLAO* produces an MDP policy in the form of an ADD mapping sets of states to actions. Leaf-nodes of this ADD are labelled with actions $\{a_u, a_0, \ldots, a_n\}$. These actions comprise an optimal policy, with the exception of a_u, a special place-holder; since some values of the variables necessary to compute the policy are never actually visited by the policy, these states are safely assigned an "unknown" action, without concern for sacrific-

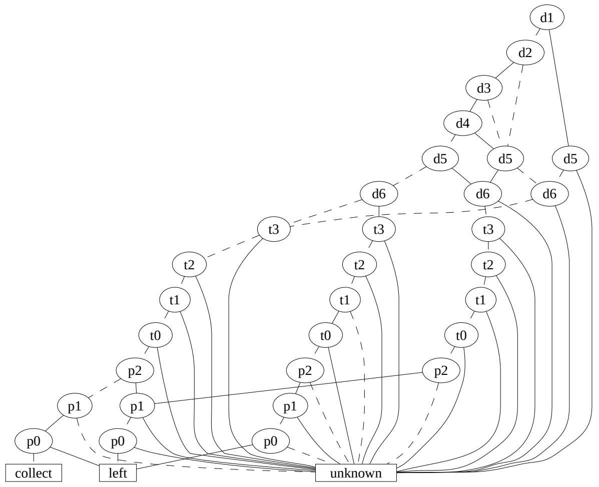

Figure 1: A sample policy output by SLAO*.

ing optimality. Figure 1 shows an example for a relatively simple instance of the Mars Rover problem with 2^{12} states, given by variables for current time, rover position, and action duration. We write ADDs as binary trees with outputs at the leaves, using a solid edge downward from a node where a variable is true, and a dashed edge where false. Here, for instance, we take action "collect" in any state corresponding to the left-most path in the ADD. On the other hand, as can be seen by the right-most path, this optimal policy never visits a state in which variables $d1$ and $d5$ are both true, and all such states are assigned the "unknown" action.

On the one hand, the ADD version of this policy is reasonably compact, relative to a more traditional look-up table representation (see Table 1), which must generate an action for every possible explicit combination of variables. On the other hand, ADD solutions for problem-domains with a larger number of variables and many possible actions will generally be far more complicated, featuring many hundreds of nodes and complicated paths. Indeed, SLAO*, like most MDP solution techniques, is ultimately indifferent to the final form of its policy. Many MDP solvers generate the policy in the form of a simple look-up table, as large as the state-space itself; here too, the final ADD can be very large and very complicated. Furthermore, even the sample policy of Figure 1 tells us nothing about such things as the *order* in which the rover will perform its actions. Policies of this sort are therefore of limited usefulness for straightforward plan analysis and verification; human users are provided with little information that might allow them, for instance, to identify errors in the initial specification of the planning domain.

Pruning an MDP policy. As explained, SLAO* produces an ADD calculating the optimal policy for all reachable states, assigning unreachable states to a default "unknown" action. Although the policy does not actually dictate what to do in such a state, the resulting plan is still optimal, since it simply does not matter which action is assigned a state one is guaranteed never to reach. By the same token, plan optimality is preserved even if we assign one of our *other* actions to an unreachable state (a point originally made by Feng and Hansen (Feng & Hansen 2002)). With this in mind, we present a pruning algorithm, which trims the ADD of those branches leading to the "unknown" action. Effectively, this routine maps any unreachable state s to one of the real actions that occurs in the optimal policy, eliminating variables inessential to computing the optimal action.

```
PRUNE
input: Policy-ADD $P$.
  while (true) do
    if ROOT is OUTPUT-NODE
      return ROOT
    else if TRUE(ROOT) is "unknown"
      set ROOT ← FALSE(ROOT)
    else if FALSE(ROOT) is "unknown"
      set ROOT ← TRUE(ROOT)
    else do
      set TRUE(ROOT) ← PRUNE(TRUE(ROOT))
      set FALSE(ROOT) ← PRUNE(FALSE(ROOT))
      return $P$
  end
```

Table 2: The policy-pruning algorithm.

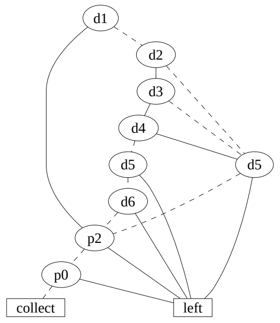

Figure 2: Pruned policy from Figure 1.

Pruning proceeds using a simple top-down algorithm that discards any node with an out-edge (true or false) pointing to the "unknown" action, and returning that node's other child instead; if neither child is "unknown," the algorithm is recursively applied to the children. Table 2 shows the pseudocode for this method; Figure 2 shows the result of applying it to the ADD from Figure 1. Unreachable value-combinations have now dropped out of the policy; it is straightforward to verify that old and new policies dictate exactly the same actions for all legitimately reachable states.

The problem of computing a minimal decision diagram without "unknown" actions is NP-complete (Sauerhoff & Wegener 1996). Thus our pruning method, which visits each node but once, can only be heuristic, and cannot guarantee that the pruned version of the policy is as small as possible, even for its fixed variable ordering. However, experimental results show that this simple method can be quite effective in significantly reducing output policy size. It is also easy enough to see that pruned policies remain optimal. Plan util-

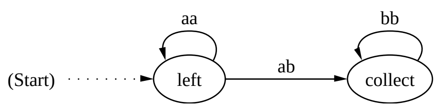

Figure 3: Plan created from policy of Figure 2.

ity is unaffected no matter what expected value is assigned to the "unknown" action a_u in any state; thus, we can assign a_u the same expected value as any other action a_i, and so can in fact simply assign a_i itself to unreachable states, without affecting the value of the policy as a whole.

Converting plan representations. While the pruning method just given reduces the size of the policy output by our MDP solver, it does nothing to answer questions about the relative ordering of actions and outcomes in the plan. For such purposes, MDP policies are not natural candidates; we therefore investigate methods for converting such policies into useful *graph-based contingency plans*.

For our purposes, a contingency plan consists of a directed graph (V, E) with starting vertex v_0. Each vertex $v \in V$ is labelled with an *action*. Edges $e \in E$ are labelled with a *set of states*. To follows a contingency plan, one starts in vertex v_0, takes the associated action a_0, and observes the state s' that results; one then follows the edge labelled with the set containing s', repeating the process for the next vertex.

The complexity of a contingency plan can be measured as some function of its graph-theoretical and other properties, chosen according to the original reason for our interest in plan simplicity. Recent research on contingency plans seeks to limit the number of branch points in a plan, either in total or along any one path, on the grounds that plans with limited branching are more easily verified (Dearden *et al.* 2003; Meuleau & Smith 2003). Prior work of the second author measures plan complexity as a weighted combination of the number of nodes, the branching factor, and the size or complexity of the labels along the edges (Horstmann & Zilberstein 2003). For purposes of plan analysis, and to catch out unexpected behaviors that might indicate errors in the original problem specification, it can be useful to reveal the overall structure of the plan by keeping edge labels relatively small and allowing more branching, if this makes parsing the expected sequence of events easier. For agents with limited storage and transmission capacity, on the other hand, one basic and important measure of complexity is simply the overall size of the plan, measured in terms of the sum total of the space required to store a computationally useful description of the nodes, edges, and labels of the plan. We develop a method that transforms policy ADDs into relatively compact contingency plans using a heuristic technique, based on the reachability analysis performed by SLAO*.

Generating a basic plan. After pruning, policies are converted to contingency plans. For each action a in the pruned policy, we generate the characteristic function of its *action-set*, $S(a)$—the states that the policy maps to a—by converting the policy-ADD into a BDD, mapping the output node for action a to 1, and all others to 0. After canonical minimization, this BDD calculates characteristic function $\chi_{S(a)}$. Graph vertices v_a are generated for each action a, with initial vertex corresponding to the action taken in the MDP's starting state. For each v_a, determine outgoing edges by:

1. Calculating $Reach(S(a), a)$, the set of all states reachable from any state in $S(a)$ by taking action a.
2. For each action-node $v_{a'}$ (including v_a itself), calculating the intersection $L = (Reach(S(a), a) \cap S(a'))$.
3. If L is non-empty, adding an edge $(v_a \rightarrow v_{a'})$, labelled with the ADD for the characteristic function, χ_L.

These operations are all easily implemented using the ADDs available. Reachability analysis uses routines that already exist for SLAO*. Calculating set intersection involves simply multiplying the ADDs for their characteristic functions.

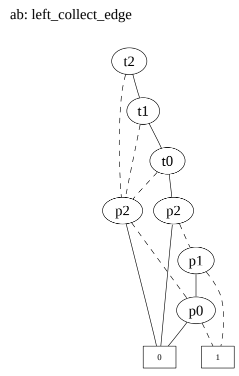

Figure 4: Edge ab from the plan in Figure 3.

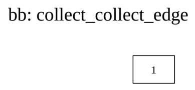

Figure 5: Edge bb from the plan in Figure 3 (this is simply the constant "1" = TRUE node).

Figure 3 shows the result of applying this method to the policy considered in Figure 2. Edges are given unique labels (the ADDs that label them are generated separately for convenience); Figures 4 and 5 show the labels for edges ab and bb, respectively. Figure 4 gives the characteristic function for the set of states in which the rover will *collect* samples, after having just moved *left*. Figure 5 is potentially more interesting. Here, the edge is simply the constant output $1 = $ TRUE; this (and the fact that there is no edge from the *collect* node back to the *left* node) indicates that in all cases, once the rover starts collecting samples, it does not move again. Thus, the plan of action is a sequence of leftward movements, followed by a period of sample collection before time expires. So, even this simple example reveals an important difference between graph-based contingency plans and the original MDP policies: while both dictate an optimal course of action, the contingency plan shows something that the MDP policy does not immediately reveal. Since the original policy in Figure 1 does not carry with it any explicit information about reachability, the actual behavior to be expected from the rover cannot be read off the policy in the way that it is made visible by the corresponding plan. Graph-based contingency plans can thus serve the purposes of plan analysis and verification more directly.

Measuring and minimizing plan complexity. The overall size of a decision diagram, measured in number of nodes, is the most important measure for computational purposes. This follows from the fact that the complexity of operations over decision diagrams is bounded by the basic size of the diagram, rather than the function that is being calculated. While particular classes of functions may on average be representable using smaller diagrams than some others, there is little about individual diagrams, apart from their relative size, that makes them harder or easier to handle. Thus, we can meaningfully measure the size and complexity of a graph-based contingency plan in this context as the simple sum of the number of vertices and edges in the graph, combined with the total number of nodes occurring in the ADDs labelling the various edges. Of course, this is not the only relevant complexity measure applicable to such plans.

Our ongoing research concerns further methods for simplifying graph-based plans. (Horstmann & Zilberstein 2003) suggest ways in which further manipulation of the initial plan generated from an MDP policy can improve plan complexity. In that work, sets of states labelling some edge in the plan are represented using intervals taken from some lines of measure, and it is noted that it can sometimes be simpler to separately describe a pair of sets S_1, S_2 than to describe their union $S = (S_1 \cup S_2)$ as a whole. In such cases, the overall size and complexity of a plan may in fact be minimized by *duplicating* a vertex v_a, and dividing the edges and labels connected to it. Plan simplification routines may then involve searching for ways of reducing plan complexity by choosing vertices and labels that can profitably be split. Not only can such splitting introduce smaller edge-labels, but it can make the overall structure of the plan more transparent.

We are currently investigating the use of such techniques in the ADD context. Bryant (Bryant 1986) gives examples of functions for which the diagram computing the union of any pair is larger than the summed size of the pair alone. The SPUDD planner (Hoey *et al.* 1999) also decomposes ADDs to keep overall diagram-size manageable. It is not yet clear how often this behavior arises for the policy-functions of interest here; we are interested in analyzing the properties of policy-ADDs that give rise to it. In the meantime, randomized and local-search techniques are again useful, in order to examine different possible ways of dividing ADDs into smaller subcomponents. This line of research has applicability in such areas as mixed-initiative planning, since it allows users to further unpack the structure of a contingency plan by isolating the relative order of events, or by focussing on the outcomes of actions for smaller subsets of states.

Conclusions and Further Directions

While the work described here employs the SLAO* algorithm in particular, it points to the general potential usefulness of reachability analysis for planning. SLAO* can solve very large planning problems, since it uses information about reachability in the planning process itself. Other algorithms can minimize plan size by exploiting the same techniques, or by performing reachability analysis after the initial plan is generated. In either case, information about which states are genuinely reachable allows for aggressive pruning of policies, while still guaranteeing optimality. Thus an algorithm like SPUDD—which also uses decision diagrams and can efficiently check reachability—could also be used to minimize and convert plan representations.

Our current work follows up on that described here. In particular, we are investigating methods for rewriting the graphical plans generated from optimal MDP solutions, focussing especially on their branching-structure. For many domains, the graphical plans produced by our method can feature action-nodes of high degree, corresponding to a large number of distinct contingencies in the execution of the plan. In addition, each action in the plan is initially represented by but a single node in the graph. Thus, it may be quite difficult to identify the exact circumstances in which one might take some particular action a; the conditions under which the plan dictates doing a may only be identified by some complicated disjunction of possible states, describing all the possible plan-paths eventuating in taking that action. On the other hand, a plan structured as a tree, in which there is only one path to any given node, can divide the possible courses of action in ways that make expected behavior under the plan more evident. By transforming the plan into a tree, we reduce the number of contingency branches down any given path. We are thus looking at ways of automatically converting our simple graph-based contingency plans into tree (or tree-like) structures; the complexity of such a plan can be identified, for instance, in terms of the ability to restrict the number of possible states one might be in at any one action-node in the plan. Of course, converting to a tree format can drastically increase the size of the plan, due to duplication of nodes and edges; this is but one case of the trade-offs that must be made, and balances struck, in pursuit of more meaningful plan representations.

In general, researchers need to face the new challenges that come with the increased success of planning methods. Plans serve the purposes of not only those who must follow them, but also those who must verify and analyze them. As problems grow increasingly complicated, we need to pay attention not only to how solutions are generated, but to how they are represented. Our work makes some first steps toward this problem, providing automated techniques for reducing the size and complexity of optimal plans, and for converting them to representations that make their structure more evident. We are interested in approaches that address both sides of the planning problem within one unified framework, balancing the requirement that plans be generally successful against the demand that they be simpler and more comprehensible. Such a framework would address at least some of the following fundamental questions:

1. How can planning algorithms exploit structured representations of problem domains in order to reduce both computational and representational complexity?
2. Do the sorts of structured domain-models used to improve

results in plan generation also lend themselves well to reducing representational complexity? What sorts of novel representational structures do we arrive at if we keep both goals in mind from the start?

3. For a given representational scheme, what well-defined measures of representational complexity best reflect potential objectives of a plan's end-user, such as overall size, verifiability, or comprehensibility?
4. What automated methods optimally—or to best approximation—reduce some well-defined measure of representational complexity, and what are the computational costs of doing so?
5. Given added objectives of reducing representational complexity, are some representations better than others? In particular, are some representations less effective for the initial solution of a planning problem, but more effective overall once we factor in the extra cost of simplification?
6. When users of a plan are willing to sacrifice some degree of optimality for simpler representations, how best can we measure and implement the resulting trade-offs?
7. How do we design and evaluate anytime algorithms for optimizing both value and simplicity of plans, and how do we control such algorithms in a real-time domain?

Of these questions, the last two, concerning the exact relationship between the quality of a plan and properties of its representational structure, are among the most interesting. Once the idea of plan simplicity and representation is taken seriously—in the sense that there are worthwhile goals having to do with how a plan is represented—then the actual and potential trade-offs between solution quality and representation become important. If some plan-representation scheme is simply useless for our particular purposes, then it will be necessary to replace the current plan with one based on a better representation, even if solution quality suffers as a result. From the standpoint of research in resource-bounded reasoning, this situation is a familiar one. Where limits on resources such as time or power do not allow for computation of a fully-optimal solution, agents often have to sacrifice plan quality. The same can be necessary where optimality must be sacrificed for the sake of a different representation.

Planning theory and practice would be well-served by research aimed at better establishing the relationship between plan quality and representational simplicity. Once these relationships are properly understood, it becomes possible to present a unified framework for planning that is optimal or near-optimal with respect to some combined measure of solution quality and representational simplicity. Such a unified perspective also makes possible the construction of algorithms that make trade-offs between these features of a plan, and opens the door for anytime methods and real-time control of planning processes. A planning methodology and architecture that pays attention not only to generating but also to representing plans will provide users with the ability to produce good plans of action that are also suitable for such purposes as verification and analysis. In mixed-initiative settings, the ability to automatically convert between computationally useful planning structures and others that are more intuitive to human users would also have obvious benefits. Although much remains to be done, the research we have presented here is one step in that direction.

Acknowledgments

This work was supported in part by the National Science Foundation under grants IIS-9907331 and IIS-0219606, and by NASA under cooperative agreement NCC 2-1311. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the NSF or NASA.

The authors also thank Zhengzhu Feng and Max Horstmann for their ideas and assistance.

References

Bahar, R. I.; Frohm, E.; Gaona, C.; Hachtel, G.; Macii, E.; Pardo, A.; and Somenzi, F. 1993. Algebraic decision diagrams and their applications. In *Proc. Intl. Conf. on Computer-Aided Design*, 188–191.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *J. of AI Research* 11:1–94.

Bresina, J.; Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. 18th Conf. on Uncertainty in AI*.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comp.* C-35:677–691.

Burstein, M., and McDermott, D. 1996. Issues in the development of human-computer mixed-initiative planning. In Gorayska, B., and Mey, J. L., eds., *Cognitive Technology*. Elsevier. 285–303.

Dean, T.; Kaelbling, L.; Kirman, J.; and Nicholson., A. 1995. Planning under time constraints in stochastic domains. *Artifical Intelligence* 76:35–74.

Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2003. Incremental contingency planning. In *Proc. ICAPS-03 Workshop on Planning under Uncertainty*.

Feng, Z., and Hansen, E. A. 2002. Symbolic heuristic search for factored Markov decision processes. In *Proc. 18th Natl. Conf. on Artificial Intelligence*, 455–460.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129:35–62.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proc. 15th Conf. Uncertainty in AI*, 279–288.

Horstmann, M., and Zilberstein, S. 2003. Automated generation of understandable contingency plans. In *Proc. ICAPS-03 Workshop on Planning under Uncertainty*.

Meuleau, N., and Smith, D. E. 2003. Optimal limited contingency planning. In *Proc. ICAPS-03 Workshop on Planning under Uncertainty*.

Sauerhoff, M., and Wegener, I. 1996. On the complexity of minimizing the OBDD size for incompletely specified functions. *IEEE Trans. CAD* 15(11):1435–1437.