Automated Generation of Understandable Contingency Plans

Max Horstmann and Shlomo Zilberstein
Department of Computer Science
University of Massachusetts at Amherst
Amherst, MA 01003
{horstman,shlomo}@cs.umass.edu

Abstract

Markov decision processes (MDPs) and contingency planning (CP) are two widely used approaches to planning under uncertainty. MDPs are attractive because the model is extremely *general* and because many algorithms exist for deriving *optimal* plans. In contrast, CP is normally performed using heuristic techniques that do not guarantee optimality, but the resulting plans are more *compact* and more *understandable*. The inability to present MDP policies in a clear, intuitive way has limited their applicability in some important domains. We examine the relationship between the two paradigms and present an anytime algorithm for deriving optimal contingency plans for an MDP. The resulting algorithm combines effectively the strengths of the two approaches.

Introduction

Two closely related decision-theoretic planning paradigms have emerged in the area of planning under uncertainty, each offering a different set of advantages. One paradigm is based on the Markov decision process (MDP) which has become a general framework for planning (Boutilier et al., 1999) and reinforcement learning (Sutton and Barto, 1998). The other paradigm is Contingency Planning (CP) (Bresina and Washington, 2001; Dearden et al., 2002). The approaches are closely related and in many cases they share the same underlying representation of actions. In both cases, an agent is manipulating an environment by performing actions with uncertain outcomes. Both paradigms have been applied to domains with continuous state variables, but we assume here that the environment has a discrete set of states and that it satisfies the Markov assumption (i.e., the outcome state is independent of the entire history given the current state and action). In each move, the agent receives some reward and the overall goal is to maximize the cumulative reward.

MDPs are solved by deriving a *policy*, which maps domain states to actions, typically represented as a large table. Several existing algorithms can construct optimal policies, but the resulting plans are not easy to visualize or understand. To reduce policy size and improve the efficiency of policy construction, researchers have developed *factored* representations that address the exponential growth of the state space with state features (Boutilier et al., 1999; Feng and Hansen, 2002). Reachability analysis and heuristic search have been used to constrain the number of examined states (Feng and Hansen, 2002) and macro actions have been used to further exploit the structure of the domain (Lane and Kaelbling, 2002). These techniques can solve efficiently realistic problems, but they do little to improve understandability.

CP is another widely used approach in stochastic domains, which allows plans to include branches that may depend on arbitrary memory states. It has been used to augment classical STRIPS-style planners (Blum and Langford, 1999), to add execution-time branch conditions to stochastic plans (Dearden et al., 2002), or to represent plans with loops in a hybrid approach (Smith and Williamson, 1995). Contingency plans are typically constructed using heuristic search and are not optimal, but the representation is much more intuitive and easy to understand.

It is clear why optimal or compact plans are desirable, but the issue of understandability is less obvious. In fact, for some applications, a plan represented as a large table mapping states to actions may be perfectly suitable. However, the lack of clarity has limited the adoption of MDP planning in some application domains such as space exploration (Bresina et al., 2002). In the domain of our interest, unmanned rovers equipped with cameras and scientific instruments are sent to collect scientific data from other planets. Because communication with the rover is restricted, it is necessary to send to the rover plans to control its operation over an extended period of time. In such high-cost missions, plan *verification* is crucial. Currently, NASA does not employ MDP-based planning because it is considered too risky. MDP policies are optimal as long as the model used to create them is accurate. Otherwise, they may introduce anomalous behavior that may be hard to detect. To maximize the safety of the mission, the 1997 Sojourner rover only executed a sequence of time-stamped low-level actions. This method does provide maximum safety, but it lacks efficiency: The average downtime due to plan failure has been estimated at 50% - 70%. More recently, less conservative approaches, which introduce branching, have been developed (Bresina and Washington, 2001; Dearden et al., 2002).

The rest of the paper examines the relationship between policies and contingency plans. In Section 2, we define the two representations and a precise measure of the complexity of a contingency plan. We introduce several problem domains in Section 3 to illustrate the definitions and for later

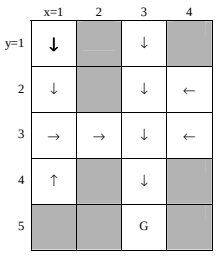


Figure 1: A simple Gridworld

evaluation. Section 4 describes an algorithm for generating and optimizing the clarity of contingency plans. Implementation and evaluation of the algorithm are discussed in Section 5. We conclude with a summary of the contribution of this work and future research directions.

Formal Problem Description

A **Markov Decision Process** (MDP) with goal states is a tuple (S, A, P, R, s_0, G). Where $S = S_1 \times \cdots \times S_n$ is a factored state space and S_i is the finite domain of feature i. We assume without loss of generality that $S_i = \{1, 2, \ldots, |S_i|\}$. A is the set of actions, $P : S \times A \rightarrow S$ is a probability distribution over successor states for each state-action pair. $R : S \times A \times S \rightarrow \mathbf{R}$ gives the expected immediate reward for each triple of state, action and successor state. $s_0 \in S$ is the initial state and $G \subseteq S$ is a set of terminal states. Note that G can be empty, so the definition includes MDPs without terminal states.

A **policy**, $\pi : S \rightarrow A$ is a mapping from states to actions. An agent acts according to policy π by executing the action $\pi(s)$ whenever it is in state $s \in S$ (initially s_0), observing the successor state $s' \in S$ and making it the current state.

Following the notation in (Littman et al., 1998), a **Contingency Plan** (CP) for a given MDP (S, A, P, R, s_0, G) is a tuple $(V, E, v_0, \rho, \delta)$, where (V, E) is a directed graph with start state $v_0 \in V$. $\rho : V \rightarrow A$ associates an action with each node of the graph. $\delta : E \rightarrow 2^S$ labels each edge with a set of states. Every pair of state sets on outgoing edges of the same node has to be disjoint; formally:

$$\forall (v, v'), (v, v'') \in E : \delta(v') \cap \delta(v'') = \emptyset$$

An agent acts according to the CP by executing the action $\rho(v)$ of the current graph node v (initially v_0), observing the successor state $s' \in S$ from the environment and succeeding to the graph node v' that satisfies $s' \in \delta(v, v')$.

A **label descriptor** $\mathcal{D}$ is a mapping $\mathcal{D}(\delta) : E \rightarrow \mathcal{L}(S)$, where $\mathcal{L}(S)$ represents a set of states in some compact and understandable language. This language could be very expressive, using various relations among state features (e.g., "the remaining distance divided by speed is less than 5."). The language used in this paper is **interval label descriptors**, which represents state sets as intervals over the feature domains defined as follows: $\mathcal{D}_{int}(\delta) : E \rightarrow \{I_1, I_2, ..., I_k\}$ where $I_j = \{[L_1; U_1], \ldots, [L_n; U_n] \mid 1 \leq L_i \leq U_i \leq |S_i| \ \forall \ 1 \leq i \leq n\}$. Note that the results we report could be generalized to more sophisticated label descriptors.

The Complexity of a Contingency Plan

We define the complexity of a plan to measure how hard it is to understand; the measure does not reflect the *computational* complexity of constructing the plan. Obviously, complexity depends on several different factors and is subjective, reflecting the perceptions and preferences of the user. Nevertheless, it is easy to identify some obvious features of a contingency plan that affect its complexity: the total number of nodes in the graph, the average branching factor, and the complexity of the labels, which depends on the representation language. These three indicators are combined into one measure of complexity by multiplying them, although our algorithm works with whatever measure of complexity is provided.

Formally, given an MDP (S, A, P, R, s_0, G), a contingency plan CP $(V, E, v_0, \rho, \delta)$ and an interval label descriptor $\mathcal{D}_{int}(\delta)$, we define a the **complexity** of the plan as follows:

$$Complex(\text{MDP}, \text{CP}, \mathcal{D}_{int}(\delta)) = |V| \cdot ABF \cdot ALDS$$

The value is nonnegative, with higher values representing more complex plans. Therefore, our goal is to increase clarity by minimizing this measure.

ABF is the average branching factor:

$$ABF = \frac{1}{|V|} \sum_{v \in V} |\{(v, v') | v' \in V, (v, v') \in E\}|.$$

ALDS is the average label descriptor size:

$$ALDS = \frac{1}{|E|} \sum_{(v,v') \in E} LabelSize(v, v')$$

where the size of a label descriptor is the number of constrained intervals.

The Value of a Contingency Plan

Given a fixed policy, an MDP becomes a Markov chain and the value of each state can be computed by solving the following set of equations:

$$Val^{\pi}(s) = \sum_{s' \in S} P(s, \pi(s), s')(R(s, \pi(s), s') + \gamma \cdot Val^{\pi}(s')$$

The value of each state is the expected discounted future reward when the agent follows π from state s (where γ is the discount factor). There are many standard algorithms for solving these equations precisely or approximately (Sutton and Barto, 1998).

The value Val^P of a contingency plan $P = (V, E, v_0, \rho, \delta)$ can be similarly computed. However, because nodes of the graph represent non-stationary memory states, it is not possible to associate a fixed value with each node of the graph. To overcome this problem, consider the Markov chain induced by the new state set $S \times V$ and transition probabilities prescribed by the MDP. The value of each state-node pair (s, v) satisfies the following Bellman equation: $Val^P(s, v) =$

$$\sum_{s' \in S} P(s, \rho(v), s')(R(s, \rho(v), s') + \gamma \cdot Val^P(s', \theta(v, s')))$$

where $\theta(v, s')$ is the successor node $v' \in V$ that satisfies $s' \in \delta(v, v')$. Obviously, if a policy π and a contingency plan P lead to identical behavior in every possible situation, their values are the same. But there are non-obvious cases. For example, a contingency plan may have an optimal value without matching any single policy, and a very compact contingency plan may have near-optimal value.

Sample Problems

The following sample problems are used to illustrate the definitions and to evaluate the algorithm we developed.

A Simple Gridworld and Mazes

Consider an agent in a Gridworld shown in Figure 1. The agent starts in $s_0 = (1, 1)$ and can move left, right, up or down: $A = \{L, R, U, D\}$. Each action moves the agent either one or two positions (avoiding the blocked grey positions) in the desired direction with probability 0.5 for each outcome. The agent receives a negative reward of $r = -1$ after each step until it reaches the absorbing state $G = (3, 5)$. The arrows show an obvious optimal policy.

Figure 2 shows three possible contingency plans. Plan (a) is obtained by simply mapping every action of the MDP to a graph node, connecting every pair of nodes with an edge and labeling the edges according to the optimal policy π. Formally: $V = \{v_1, \ldots, v_{|A|}\}$, $E = V \times V$, $\rho(v_i) = \pi(a_i)$, $\delta(v_i, v_j) = \{s \in S | \pi(s) = a_j\}$. The start node v_0 is the node labeled with $\pi(s_0)$. Obviously, this plan is not simple. Its complexity is: $4 \cdot 4 \cdot 88/16 = 88$.

Plan (b) is the result of simplifying plan (a) by eliminating unreachable states and taking advantage of the interval label descriptors. As a final step intervals are represented by equations leading to further simplification. The complexity of the plan is reduced to: $4 \cdot 10/4 \cdot 14/10 = 14$.

Plan (c) shows another improvement. Intuitively, this plan "remembers" whether the agent is walking down in the first column ($x = 1$) or the third one ($x = 3$), by splitting a node. Even though we increase the number of nodes, we improve clarity; it is now very easy to understand what the plan tells the agent, because every label consists only of one expression and the average branching factor is slightly reduced from 2.5 to 2.2. The self-loop edge of the second D-node has no label at all; once this node is entered, the agent goes down unconditionally until it reaches the goal. The complexity of this plan is: $5 \cdot 11/5 \cdot 10/11 = 10$.

The above simple Gridworld is small enough to illustrate our motivation and objectives. We have also experimented with larger, randomly generated mazes, to test the scalability of the algorithm described in Section 4.

A Planetary Rover Problem

The second set of problems we used involve a rover on a slope that has a limited amount of time to perform some scientific experiments and collect data (See Figure 3). Each state includes the current position of the rover, $1 \leq Pos \leq 7$, and the remaining time $0 \leq T \leq maxTime$. The action $Left$ and $Right$ control the movement of the rover and the action $Collect$ performs an experiment and saves the data.

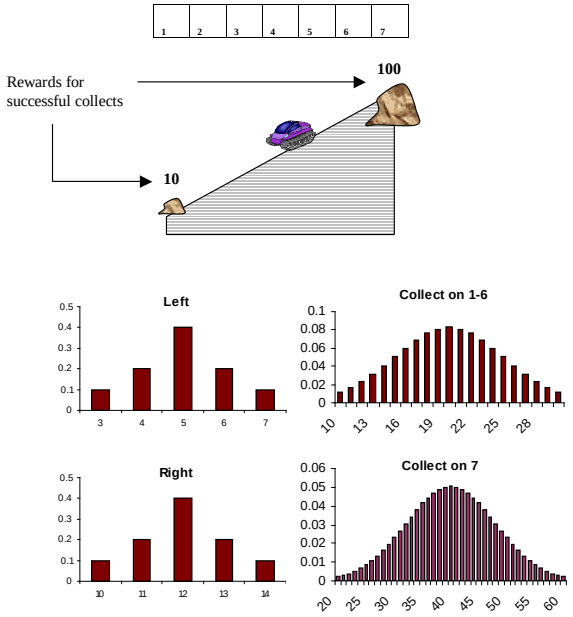


Figure 3: The planetary rover problem.

One interesting target for experimentation is at the bottom of the slope ($Pos = 1$), where each $Collect$ has a reward of 10. Another target is at the top ($Pos = 7$), with a higher reward of 100. In other positions $Collect$ does nothing (can be used to idle). The rover starts and must finish its activity in position 4; there is a penalty of $-10,000$ for not reaching this position by the deadline ($T = 0$). There is uncertainty about the duration of each action, described by a Gaussian distribution. Going right (up) takes on average more time than going left (down), and collecting data in position 7 takes significantly more time than in other positions.

With $maxTime = 240$ seconds, an optimal policy represented as a lookup table has about 1500 entries and is therefore not easy to understand. Figure 4 shows two possible contingency plans for the problem, derived from an optimal policy. The top graph shows an initial plan generated by mapping every action to one node and labeling the edges appropriately. Because the state sets on the labels are large (on average ≈ 500 elements), this plan scores poorly according to our complexity measure; its complexity is 10122.

The bottom graph shows one of the best possible contingency plans. Although the number of nodes is doubled, the labels are very compact. The average branching factor also goes down from to 3 to 1.83. The complexity of this plan is $6 \cdot 11/6 \cdot 1 = 11$. The challenge is therefore to generate such contingency plans automatically. The following sections introduce an effective algorithm for doing that and examine its performance.

Automated Contingency Plan Generation

The algorithm for automated generation of understandable contingency plans is shown in Figure 5. Our approach relies on solving first the underlying MDP, (S, A, P, R, s_0, G) and

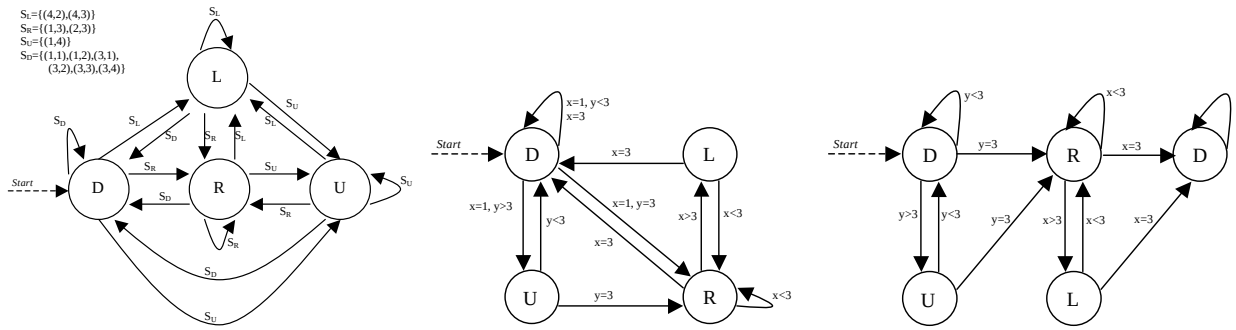


Figure 2: Three possible contingency plans for the Gridworld

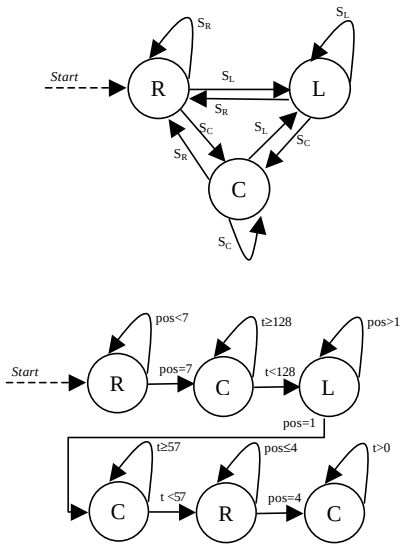


Figure 4: Two possible contingency plans for the rover.

obtaining an optimal policy π. The algorithm generates an *initial* contingency plan, constructed directly from the policy as illustrated in Figure 2(a). The resulting plan is ($V = \{v_1, \ldots, v_{|A|}\}$, $E = V \times V$, $\rho(v_i) = \pi(a_i)$, $\delta(v_i, v_j) = \{s \in S | \pi(s) = a_j\}$). Then, the following operators are used to reduce the complexity of the plan.

Reachability Analysis

Algorithms that compute policies for MDPs can be improved by taking into account the fact that some states are not reachable from the start state. Furthermore, a policy need not specify actions for states not reachable *under* that policy. Search algorithms such the LAO* take advantage of this (Hansen and Zilberstein, 2001).

The same applies to contingency plans: A state on an edge-label that can never be reached when the plan is executed can be removed. Formally, $s' \in \delta(v, v')$ can be removed from $\delta(v, v')$ if for any (s, v) that is reachable from (s_0, v_0), $P(s, \rho(v), s') = 0$. If all the states on an edge label can be removed, the edge itself can be removed. If a node becomes unreachable due to removed edges, it can also be removed. The procedure STATEREACHABILITY performs this simplification by performing a depth-first search over the set of all reachable *edges* between the state-node pairs.

Merging Intervals to n-dimensional Boxes

As illustrated in Section , a benefit of interval label descriptors is the ability to capture easily large sets of states and simplify the branch conditions. This is particularly effective when the order of feature values corresponds to a natural ordering in the domain such that neighboring values are likely to have the same optimal action. One typical example is resource variables such as time or energy, for which the optimal behavior in certain situations depends on whether the value has dropped below some critical threshold.

Formally, given a state set $\delta(v, v') \subseteq S$, label descriptor minimization involves finding $\text{argmin}_{I \subseteq \text{INTERVALS}(S)} LabelSize(I)$. Finding an optimal label descriptor is certainly intractable for larger state sets, but we developed a simple approximation that performs surprisingly well. The idea is to initially

Anytime Contingency Plan Generation

inputs: $S, A, P, R, s_0, G, \pi, \epsilon$
π is a policy for an MDP (S, A, P, R, s_0, G)
ϵ is the fraction of optimality that may be lost

$CP \leftarrow$ CREATEINITIALPLAN(π)
STATEREACHABILITY(CP)
MERGEINTERVALS(CP)
while not interrupted **do**
  $CP' \leftarrow$ COPY(CP)
  CHOOSEPARTITION(CP', S_1, S_2)
  SPLITNODES(CP', S_1, S_2)
  STATEREACHABILITY(CP', ϵ)
  MERGEINTERVALS(CP', ϵ)
  if *Complex*(CP') < *Complex*(CP)
    then $CP \leftarrow CP'$
end
return CP

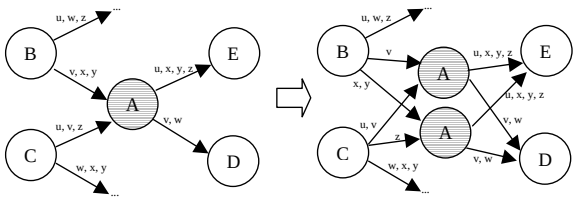
Figure 5: The contingency plan generation algorithm.

select an ordering of the state features and then combine "neighboring" intervals subsequently along the dimensions of the state space. In other words, given an ordering $\{S_1 < S_2 < \ldots < S_n\}$, the method first tries to combine sets of states that are identical in all state variables except S_1. The resulting set of intervals are being merged where possible along S_2 and so on. The procedure MERGEINTERVALS performs this optimization.

Splitting Nodes

Even with reduced label sizes and after removing unreachable states, the initial contingency plan with one node per action basically just reproduces the policy on the outgoing labels of each node. The real strength of a plan representation lies in the encoding of partial state information in the nodes, leading to multiple nodes with the same action. Therefore, further reduction in complexity is possible by splitting nodes. Intuitively, we want to identify groups of states that have the same optimal action according to the policy π. When subsets of the group can be described with a small set of descriptors, whereas the union is not easy to describe, a split could be beneficial.

The procedure SPLITNODES performs this operator on a contingency plan, splitting a node into two nodes with the same action, adapting the incoming and outgoing edges of the original node to the new nodes. For example, with a state set $S = \{u, v, w, x, y, z\}$ and the partition $S_1 = \{u, v, w\}$, $S_2 = \{x, y, z\}$, the result of a split operator is illustrated below:



Note that the gray node labeled with the action "A" is being split.

The Overall Plan Generation Algorithm

The operators described above are used by the contingency plan generation algorithm. After constructing the initial plan and performing state reachability analysis and merging intervals, the algorithm performs additional improvements in a stochastic manner and can be stopped at any time. The interruptible loop involves choosing candidates for node splitting and keeping the new plan if it is more understandable (i.e., less complex) than the current best plan. Exhaustive search for candidates for splitting is not feasible. Instead, candidates are chosen at random or using a domain specific heuristic, which can also be randomized. After splitting nodes, the algorithm performs reachability analysis and merges intervals on the resulting plan to reduce its complexity. This part of the algorithm is interruptible, offering a tradeoff between computation time and plan complexity.

The algorithm can handle effectively the problems we used for evaluation. Consider for example the Gridworld problem and the contingency plans shown in Figure 2. In this case, the STATEREACHABILITY and MERGEINTERVALS operators transform the initial plan (a) into the plan (b). To get the improved plan (c), the SPLITNODES operator could be applied to "D", using the partition $S_1 = \{(x, y)|x = 3\}, S_2 = \{(x, y)|x \neq 3\}$ before performing again reachability analysis merging intervals.

Trading off Optimality for Clarity

Besides a tradeoff between computation time and plan complexity, the final algorithm has a parameter that introduces another tradeoff: between optimality (the value of the plan) and clarity (the complexity of the plan). This tradeoff is controlled by the parameter ϵ which indicates that $(1 - \epsilon)Val^{\pi}$ is an acceptable value if it facilitates complexity reduction. When $\epsilon = 0$, the algorithm returns only optimal plans. Otherwise, the operators discussed above use ϵ in a variety of ways to achieve further reduction in complexity. Consider for example the MERGEINTERVALS operator. If the values of a feature extend to a large region except for some a small number of cases, then the operator can ignore these "holes" and construct a large interval covering the entire region. This could reduce a label descriptor size dramatically and improve the clarity of the plan, so we might be willing to accept a small loss of value (bounded by ϵ).

This leads to the following interesting optimization problem. For any given MDP (S, A, P, R, s_0, G) and a known optimal value Val_{max}, find a contingency plan and interval label descriptors that have at least a value of $(1 - \epsilon)Val_{max}$ while minimizing the complexity of the plan. In other words, we want to find a member of the following set that has the lowest complexity.

$$\{CP = (V, E, v_0, \rho, \mathcal{D}_{int}(\delta)) \mid Val^{CP} \geq (1 - \epsilon)Val_{max}\}$$

While the computational complexity of this problem has not been determined, it is clear that optimizing both plan value

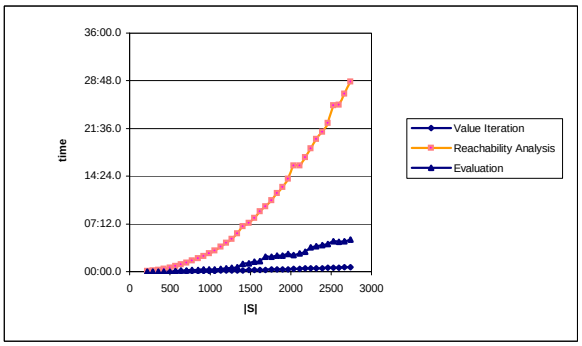


Figure 6: Execution time in minutes and seconds of value iteration, reachability analysis and plan evaluation applied to different versions of the rover problem.

and complexity is too hard. Hence, our algorithm guarantees an arbitrary level of optimality, but it does not guarantee finding the most compact plan.

Experimental Results

We have implemented the contingency planning algorithm and tested it with the Gridworld problem, maze problems ranging from 10×10 to 90×90, and rover problems with $maxTime$ between 30 and 390. The maximum number of states is 8100 for the maze problems and 2737 for the rover problems. Because the algorithm is an anytime algorithm, its overall run-time depends on the user's preferences. So we examined instead the run-time of the two operators that dominate its run time: reachability analysis and contingency plan evaluation. We also measured the computation time needed to solve the original MDP by value iteration, because that policy is needed for constructing the initial CP. Figure 6 shows execution times for different instances of the rover problem; the results with mazes show a similar behavior, although the cost of the operators is somewhat lower. As one would expect, reachability analysis is the most expensive operator, whereas node splitting and merging intervals take a negligible amount of time. Contingency plan evaluation has a significant cost, but it does not grow as fast as the cost of reachability analysis. From this we conclude that improving the efficiency of reachability analysis is needed in order to apply the technique to much larger problem instances.

Performance with Maze Problems

While the generation of the optimal contingency plan for the simple Gridworld in Figure 1 is trivial and can be found by the algorithm in a few seconds, finding optimal contingency plans is much harder for the larger maze problems. However, in our experimentation we found that the algorithm produces surprisingly good plans and exhibits very interesting behavior. One typical complexity reduction in this domain can be attributed to detecting a "bottleneck" through which the agent must pass. In this case the algorithm splits the nodes according to the partition derived from the states before and after the bottleneck. Another typical situation that the algorithm can detect involves clusters of neighboring states with the same optimal action that can be split into single nodes with one outgoing edge and one self-loop edge. This produces very simple label descriptors with low branching factor. There are additional classes of simplifications that the algorithm detects; they are best illustrated graphically, but due to space limitation we cannot include these figures.

Performance with Rover Problems

In the rover domain we have experimented with different problem instances by varying the amount of available time. The optimal plans exhibit a similar behavior over all instances. As long as there is enough time available for the fruitful, but time-consuming experiments on the right target, the rover drives there and performs these experiments. As soon as the time drops below a certain threshold, the rover drives to the leftmost position, exploiting as much of the remaining time as possible on the smaller target. When the remaining time drops further below a second critical threshold, the rover finally decides to return to the home position.

Observing these characteristics of the optimal policy, it is possible to guide the algorithm to try the split operators on the "collect" node: The different split nodes memorize whether the rover is "on its way" to the right target, the left target or the home position. This illustrates how domain structure could be exploited in order to heuristically guide the algorithm to avoid searching in the space of all possible plan transformations, resulting in a very understandable plan. Applied to the initial contingency plan at the top of Figure 4, this heuristic guides the algorithm to produce the bottom plan in less then 4 minutes. For larger instances of the problem, such heuristics cannot guarantee finding the optimal plan within a reasonable amount of time. But the algorithm can still produce understandable plans. Further ways to exploit domain structure will be examined in future experiments.

Conclusions and Future Directions

The main objective of this work has been to find solutions for decision-theoretic planning problems that are optimal (or near-optimal), compact, and understandable. Our solution leverages the optimality of MDP policies and the compactness and clarity of contingency plans to form plans that share the advantages of both paradigms. Others have introduced CP to approximate the solution to an MDP by searching in the space of finite state controllers (e.g., (Kim et al., 2000)). But the motivation has been to reduce computation time, not to improve the clarity of the result. To our knowledge, this is the first attempt to optimize the clarity of MDP policies. The resulting plans are attractive in mission-critical domains in which the ability to understand and verify a plan is as important as its optimality.

We defined a precise measure of the complexity of contingency plans, reflecting their size, branching factor and the size of the label descriptors. We then introduce several operators to reduce the complexity of plans. Because a complete search through the space of all possible transformations is obviously intractable, an iterative improvement anytime algorithm is constructed than can be guided by domain-specific heuristic knowledge. The experimental results with

small and medium size problem instances are encouraging. They show that we can automate the process of generating understandable plans that previously had to be hand crafted.

There are several interesting ways in which the algorithm can be generalized and improved. First, a better measure of the complexity of plans could be developed. For instance, it might be acceptable to have a large number of nodes, as long as the overall plan is decomposable into different regions that can be analyzed by experts independently. Second, additional operators for reducing complexity could be added. Finally, the language for representing edge labels could be enriched with various predicates and allow state features to be continuous. The results reported in this paper provide a good framework for further exploration of these research directions.

Acknowledgments

Support for this work was provided in part by NASA under grant NAG-2-1463. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of NASA.

References

J. Bresina and R. Washington. Robustness via Run-Time Adaptation of Contingent Plans. *AAAI Spring Symposium on Robust Autonomy*, 2001.

J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Planning Under Continuous Time and Resource Uncertainty: A Challenge for AI. *Conference on Uncertainty in Artificial Intelligence*, Edmonton, Alberta, July 2002.

A. Blum and J. Langford. Probabilistic Planning in the Graphplan Framework. *Proceedings of the Fifth European Conference on Planning*, 319–332, 1999.

C. Boutilier, T. Dean, and S. Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1-94, 1999.

R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Contingency Planning for Planetary Rovers. *Third International NASA Workshop on Planning & Scheduling for Space*, Houston, Texas, 2002.

Z. Feng and E.A. Hansen. Symbolic Heuristic Search for Factored Markov Decision Processes. *Eighteenth National Conference on Artificial Intelligence*, Edmonton, Alberta, July 2002.

E.A. Hansen and S. Zilberstein. LAO*:A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence*, 129(1-2):35-62, 2001.

K.-E. Kim, T.L. Dean and N. Meuleau. Approximate Solutions to Factored Markov Decision Processes via Greedy Search in the Space of Finite State Controllers. *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, Colorado, 2000.

T. Lane and L.P. Kaelbling. Nearly Deterministic Abstraction of Markov Decision Processes. *Eighteenth National Conference on Artificial Intelligence*, Edmonton, Alberta, July 2002

M.L. Littman, J. Goldsmith, and M. Mundhenk. The Computational Complexity of Probabilistic Planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

D. Smith and M. Williamson. Representation and Evaluation of Plans with Loops. *AAAI Spring Symp. on Extended Theories of Action*, Stanford, CA, 1995.

R.S. Sutton and A.G. Barto. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 1998.