Finding Best *k* Policies

Peng Dai[1] and Judy Goldsmith[2]

[1] Computer Science & Engineering University of Washington, Seattle WA 98195-2350
daipeng@cs.washington.edu
http://www.cs.washington.edu/homes/daipeng

[2] Univ. of Kentucky, Dept. of Comp. Sci. Lexington, KY, USA 40506-0046
goldsmit@cs.uky.edu
http://www.cs.uky.edu/~goldsmit

Abstract. An optimal probabilistic-planning algorithm solves a problem, usually modeled by a Markov decision process, by finding its optimal policy. In this paper, we study the k *best policies* problem. The problem is to find the k best policies. The k best policies, $k > 1$, cannot be found directly using dynamic programming. Naïvely, finding the k-th best policy can be Turing reduced to the optimal planning problem, but the number of problems queried in the naïve algorithm is exponential in k. We show empirically that solving k best policy problem by using this reduction requires unreasonable amounts of time even when $k = 3$. We then provide a new algorithm, based on our theoretical contribution to prove that the k-th best policy differs from the i-th policy, for some $i < k$, on exactly one state. We show that the time complexity of the algorithm is quadratic in k, but the number of optimal planning problems it solves is linear in k. We demonstrate empirically that the new algorithm has good scalability.

1 Introduction

Markov Decision Processes (MDPs) [1] are a powerful and widely-used formulation for modeling probabilistic planning problems [2,3]. For instance, NASA researchers use MDPs to model the Mars rover decision making problems [4,5]. MDPs are also used to formulate military operations planning [6] and coordinated multi-agent planning [7], *etc.*

An optimal planner typically takes an MDP model of a problem and outputs an optimal plan. This is not always sufficient. In many cases, a planner is expected to generate more than one solution.

Furthermore, in the modeling phase, not every aspect of nature can be easily factored in a problem representation. For the case of NASA rover, for example, there are many safety constraints that need to be satisfied [5]. An optimal plan might be very close to a risky value—but another may not have many risks and so it is better to prefer the slightly suboptimal one. Similarly there are many decision criteria—probability of reaching the goal, expected reward, expected risk, various preferences, *etc.* Combining them into a single criterion is hard, and multi-objective planning is too slow [8,9]. Thus, a good alternative is to look for many suboptimal plans given a single criterion and later pick one that looks the best according to all criteria.

In this paper, we look at the *k best policies* problem. Given an MDP model, the problem is to find the k best policies, ranked by the expected value of the initial state, tie-broken by the "closeness" to a better policy, followed by lexical order of the policies. The classical optimal planning problem is a special case of the k best policy problem where $k = 1$. The optimal planning problem can be solved by *dynamic programming*, as the property of the optimality of sub-problems holds. The k best policy problem be directly solved by dynamic programming. However, finding the k-th best policy can be brute-force reduced to exponentially many instances of the optimal planning problem. Our experiments show that solving the k best policy problem this way requires unreasonable time even when $k = 3$.

A very similar problem has been explored by Nielsen, et al. [10,11,12]. Nielsen and Kristensen observed that the problem of finding optimal *history-dependent* policies (maps from the state space crossed with the time step to the action space) can be modeled as finding "a minimum weight hyperpath" in directed hypergraphs. A vertex in the hypergraph represents a state of the MDP at a particular time; the hypergraphs are, therefore, acyclic. They present an elegant and efficient algorithm for finding the k best time-dependent policies for an MDP. However, their algorithm cannot handle MDPs with probabilistic cycles, therefore its usefulness is limited.

Our new solution to the k best policy problem follows from the property: *The k-th best policy differs from a better policy on exactly one state.* We propose an original algorithm for the k best policy problem that leverages this property. We demonstrate both theoretically and empirically that the new algorithm has low complexity and good scalability.

2 Background

2.1 Markov Decision Processes

AI researchers often use MDPs to formulate probabilistic planning problems. An MDP is defined as a four-tuple $\langle \mathcal{S}, \mathcal{A}, T, C \rangle$, where $\mathcal{S}$ is a finite set of discrete states, $\mathcal{A}$ is a finite set of all applicable actions, T is the transition matrix describing the domain dynamics, and C denotes the cost of action transitions.

The agent executes its actions in discrete time steps called *stages*. At each stage, the system is at one distinct state $s \in \mathcal{S}$. The agent can pick any action a from a set of *applicable actions* $Ap(s) \subseteq \mathcal{A}$, incurring a cost of $C(s, a)$. The action takes the system to a new state s' stochastically, with probability $T_a(s'|s)$.

The *horizon* of an MDP is the number of stages for which costs are accumulated. We focus our attention on a special set of MDPs called *stochastic shortest path* (SSP) problems. The horizon in such an MDP is indefinite and the costs are accumulated with no discounting. There are an initial state s_0, and a set of sink *goal states* $\mathcal{G} \subseteq \mathcal{S}$. Reaching any state $g \in \mathcal{G}$ terminates the execution. The cost of the execution is the sum of all costs along the path from s_0 to g. Any infinite horizon discounted reward MDP can easily be converted to an undiscounted SSP [13].

To solve the MDP we need to find an *optimal policy* ($\pi^* : \mathcal{S} \rightarrow \mathcal{A}$), a probabilistic execution plan that reaches a goal state with the minimum expected cost. We evaluate any policy π by a *value function.*

$$V_\pi(s) = C(s, \pi(s)) + \sum_{s' \in \mathcal{S}} T_{\pi(s)}(s'|s)V_\pi(s').$$

Any optimal policy must satisfy the following system of *Bellman equations*:

$$\begin{aligned} V^*(s) &= 0 \quad \text{if } s \in \mathcal{G} \text{ else} \\ V^*(s) &= \min_{a \in Ap(s)} [C(s,a) + \sum_{s' \in \mathcal{S}} T_a(s'|s)V^*(s')]. \end{aligned} \tag{1}$$

The corresponding optimal policy can be extracted from the value function:

$$\pi^*(s) = argmin_{a \in Ap(s)} [C(s,a) + \sum_{s' \in \mathcal{S}} T_a(s'|s)V^*(s')].$$

2.2 Dynamic Programming

We define a *sub-problem* of an MDP with state space $\mathcal{S}' \subseteq \mathcal{S}$ to be a self-contained MDP with state space $\mathcal{S}'$ and associated action transitions. We define the *sub-policy* of a policy π given a sub-problem with state space $\mathcal{S}' \subseteq \mathcal{S}$ to be the mapping from all $s \in \mathcal{S}'$ to $\pi(s)$. An optimal policy satisfies the following necessary and sufficient condition: for any sub-problem, the corresponding sub-policy is also optimal. Many optimal MDP algorithms are based on dynamic programming. Its usefulness was first proved by a simple yet powerful algorithm called *value iteration* (VI) [1]. Value iteration first initializes the value function arbitrarily. Then the values are updated iteratively using an operator called *Bellman backup* to create successively better approximations per state per iteration. Value iteration stops updating when the value function converges (one future backup can change a state value by at most ϵ, a pre-defined threshold).

Another algorithm, named *policy iteration* (PI) [14], starts from an arbitrary policy and iteratively improves the policy. Each iteration of PI consists of two sequential steps. The first step, *policy evaluation*, finds the value function of the current policy. Values are calculated by solving the system of linear equations (in the original PI algorithm), or by iteratively updating the value functions in the VI manner till convergence (modified policy iteration [15]). The second step, *policy improvement*, updates the current policy by choosing a greedy action per state by a one step lookahead, based on the value function calculated in the policy evaluation step. PI stops when the policy improvement step doesn't change the policy.

3 *k* Best Policy Problem

Classical dynamic programming successfully finds *one* optimal policy of an MDP in time polynomial in $|\mathcal{S}|$ and $|\mathcal{A}|$ [16,17]. In this paper, we find the k best policies of an MDP. We first give the formal definition of the k best policy problem. Then we introduce the main theoretical contribution of the paper by proving a very strong result about the k-th best policy.

Let M be an MDP, π a policy for M. We define the *policy graph* of M given π, denoted by G_π, to be a graph constructed by: (1) the set of states (vertices) that are reachable from s_0 given π, and (2) their corresponding transitions in π (edges).

Let s and s' be states of M. We say that s' is a *policy descendant* of s with respect to π if there is a path from s to s' in G_π or if $s = s'$. We define $Policydesc(s, \pi)$ to be the set of all policy descendant states of s under policy π. We assume that, for every state $s \in \mathcal{S}$, there are at least two possible actions for s.

Note that, for a given MDP and a given value function, there may be multiple policies with that value function. We define a notion of "best among equals", namely, the "closest" to better policies followed by a lexicographic ordering, so that the notion of "best policy" is well defined.

Lemma 1. *Using value iteration, we can find an optimal value function for M, and the optimal $V_{\pi^*}(s_0)$. We can then find the lexicographically least policy, π_1, that has that value for $V_{\pi_1}(s_0) = V_{\pi^*}(s_0)$.*

The proof of Lemma 1 is straightforward. Given the value function, for each state, we choose the lexicographically first action that achieves the desired value. (If $\mathcal{A} = \{a_0, a_1, \ldots, a_j\}$, the lexicographically first action satisfying a property is the lowest-numbered a_i with that property.) Once we have the *best* policy, we then need to define an ordering on policies so that we may define the k-th best.

Definition 1. *Given two policies π and π', we can consider them as vectors of length $|\mathcal{S}|$ over alphabet $|\mathcal{A}|$, and define the Hamming distance $Ham(\pi, \pi')$ to be the number of states on which π and π' differ. We also define $<_{lex}$ to be the lexicographic ordering on such vectors.*

Finally, we define an order on policies.

Definition 2. *Given an MDP M and a dynamic list of p best policies generated so far $\{\pi_1 \ldots, \pi_p\}$, the next best policy is computed based on the following ordering $\prec$ on the rest of the policies for M.*

$$\begin{array}{lll} \pi \prec \pi' & \textbf{if} & V_\pi(s_0) < V_{\pi'}(s_0) \\ & \textbf{else if} & min_{j \leq p} Ham(\pi_j, \pi) < min_{j \leq p} Ham(\pi_j, \pi') \\ & \textbf{else if} & \pi <_{lex} \pi'. \end{array}$$

Intuitively, two policies with the same initial state value are first compared by how "close" each one is to some better policy, followed by lexicographic order if they are equally close.

Theorem 1. *Let M be an MDP, and let $\{\pi_1, \ldots, \pi_k\}$ be the k best policies for M, in order. Let $k \geq 1$. Then there is some $m < k$ such that π_k differs from π_m on exactly one state.*

The proof sketch to Theorem 1 is provided in the Appendix.

4 Algorithm

Consider the k-th ($k > 1$) best policy of an MDP M, called π_k. The necessary and sufficient condition of the optimality on sub-problems does not hold. With the loss of the optimality on sub-problems, dynamic programming is not immediately applicable. However, we can reduce it to many optimal planning problems, each solved by dynamic programming. Before illustrating the reduction, we present the high-level idea of our first algorithm in Algorithm 1. We call it *k best naïve* algorithm (KBN), as it is a brute force algorithm that doesn't use Theorem 1. KBN is based on the following observation: The $k + 1$-st best policy must differ from each of the k best policies on at least one state. We can enumerate the possible sets of state/action pairs the new policy must avoid, and find an optimal policy for each thus-constrained MDP, then take the best of those policies.

Algorithm 1. k best naïve (KBN)

```
1: **Input:** $M$ (an MDP), $k$
2: find best policy $\pi_1$ by VI
3: $\Pi \leftarrow \{\pi_1\}$
4: **for** $i \leftarrow 2$ to $k$ **do**
5:     $\pi_i \leftarrow$ best policy that differs from any policy $\pi \in \Pi$ by at least one state
6:     $\Pi \leftarrow \Pi \cup \{\pi_i\}$
7: return $\pi_1, \ldots, \pi_k$
```

For instance, given the best and second best policies, π_1 and π_2, to find π_3, we say that either it differs from π_1 on s_0 and from π_1 on s_0, or from π_1 on s_0 and from π_1 on s_1, or.... In this case, we solve $|\mathcal{S}|^2$ many optimal planning problems. To find the k-th best policy, we solve $|\mathcal{S}|^k$ many. Each newly-computed policy will be compared with the best policy computed so far, so that the number of comparisons is linear in the number of policies computed. Suppose we use VI to solve those optimal planning problems, KBN has a complexity $|\mathcal{S}|^k \times O(VI)$, an exponential function of k.

Some of these combinations of constraints may constrain away all actions for a particular state, so do not yield a next-best policy. However, the next best policy must be among those computed, and will be the best such.

Using Theorem 1, we have a new algorithm, called *k best improved* (KBI). The KBI pseudo-code is shown in Algorithm 2. KBI keeps a set of candidate policies $\mathcal{P}$, which is initially empty. We first find the optimal policy by value iteration. To find the i-th best policy, we generate $k - i + 1$ distinct policies as candidates. These candidates (1) must not be duplicates of any policy in $\mathcal{P}$, and (2) each differs from π_{i-1} on exactly one state. We have the following theorem.

Theorem 2. *The i-th best policy must be an element of $\mathcal{P}$.*

Proof. As we know from Theorem 1 that the i-th ($i \leq k$) best policy is exactly one state different from one of $\pi_1, \ldots, \pi_{i-1}$, say, π_j, where $j < i$. Therefore, it must have been generated when π_{j+1} was computed. Since it is the i-th best policy, it would

Algorithm 2. k best improved (KBI)

```
1: **Input:** $M$ (an MDP), $k$
2: find best policy $\pi_1$ by VI
3: $\mathcal{P} \leftarrow$ empty set
4: **for** $i \leftarrow 2$ to $k$ **do**
5:     generate distinct $k-i+1$ best policies that each differs from $\pi_{i-1}$ on exactly one
       state and differs from $\{\pi_1, \ldots, \pi_{i-1}\}$ and insert them into $\mathcal{P}$ in order, discarding
       duplicates
6:     $\pi_i \leftarrow$ the best policy in $\mathcal{P}$
7:     delete $\pi_i$ from $\mathcal{P}$
8: return $\pi_1, \ldots, \pi_k$
```

have been amongst the $i-j$-th best of those policies that are one state different from π_j, so it belongs to the $k-j$ best policies added to $\mathcal{P}$ at stage $j+1$.

Thus, we find the i-th best policy by picking the best policy in $\mathcal{P}$. There are $(|\mathcal{A}|-1)\times|\mathcal{S}|$ policies that are exactly one state different from π_i. Finding the best $k-i$ of them has a complexity $|\mathcal{A}| \times |\mathcal{S}| \times O(\textit{policy evaluation})$, plus the complexity of keeping the list $\mathcal{P}$ in sorted order ($O(k^2 \log k)$). KBI computes these policies $k-1$ times, so its complexity is $(k-1) \times |\mathcal{A}| \times |\mathcal{S}| \times O(\textit{policy evaluation})$, a linear function of k. (Note that the sorting term is dominated by $|\mathcal{A}| \times |\mathcal{S}| \times O(\textit{policy evaluation})$.)

5 Experiments

We address the following three questions in our experiments: (1) How does KBI compare with KBN on different problems and k values? (2) Does KBI scale well on large k values? (3) How different are the k best policies from the optimal policy?

We implemented KBN and KBI in C. We performed all experiments on a 2.2GHz Dual-Core Intel(R) Core(TM)2 Processor with 6GB memory. We picked problems from three domains, namely Racetrack [18], Single-arm pendulum (SAP) and Double-arm pendulum (DAP) [19]. We used a threshold value of $\epsilon = 10^{-6}$.

5.1 Comparing KBI and KBN

We compare KBN and KBI on a suite of six problems of various sizes. The running times of both algorithms when $k = 2$ are listed in Table 1. We see that KBI outperforms KBN on all problems. In four problems, the speedup is an order of magnitude. According to our analysis in the Algorithm section, when k increases by 1, the running time of KBN increases by a factor of $|\mathcal{S}|$, so for cases $k = 3$ and $k = 4$ we take the expectations of its running time based on its performance on the same problem when $k = 2$. Even for small k values, the running times of KBN are prohibitively high. For example, in SAP 2 problem, its expected running time is approximately one thousand hours for $k = 3$ and tens of millions of hours for $k = 4$.

Table 1. Running time (seconds) of KBN and KBI in various problems with different k values. The running time of KBN on $k > 2$ are expectations. KBI outperforms KBN on most problems by an order of magnitude even when $k = 2$.

Domain	States	$k = 2$		$k = 3$		$k = 4$	
	$\|\mathcal{S}\|$	KBN	KBI	KBN	KBI	KBN	KBI
				(expected)		(expected)	
DAP 1	625	0.90	0.44	10^2	0.87	10^5	1.32
Racetrack 1	1,847	0.56	0.07	10^3	0.14	10^6	0.21
SAP 1	2,500	12.39	2.58	10^4	4.93	10^7	7.29
SAP 2	10,000	461.87	66.15	10^6	131.30	10^{10}	196.46
DAP 2	10,000	944.14	333.97	10^6	665.89	10^{10}	1001.23
Racetrack 2	21,371	11.10	2.02	10^5	4.03	10^9	6.02

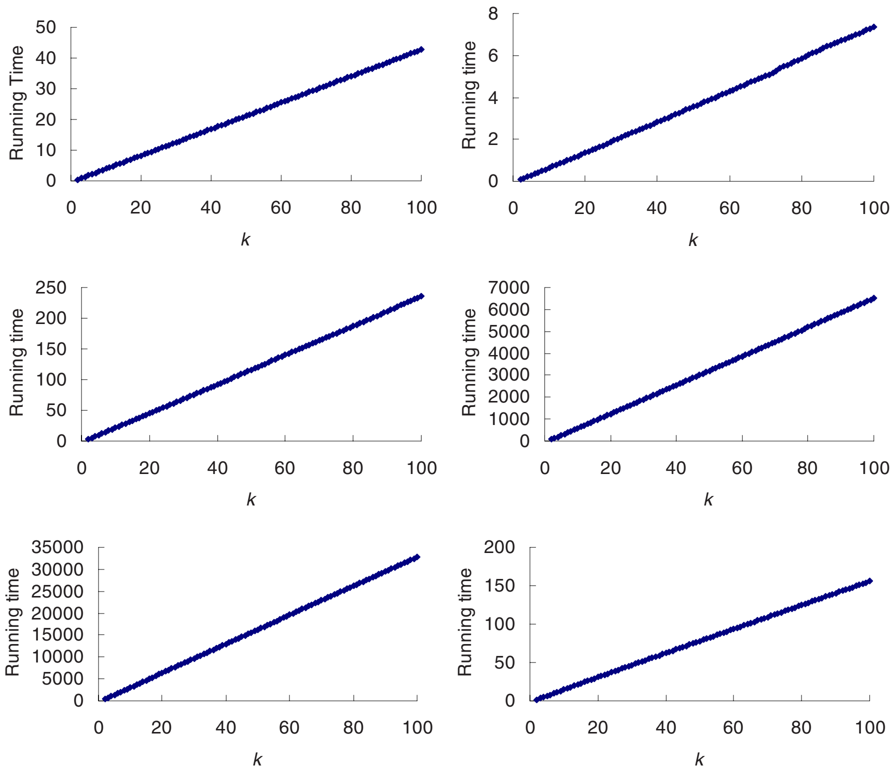

Fig. 1. Running time (seconds) of KBI when $k = 2, \ldots, 100$ on DAP 1, Racetrack 1, SAP 1, SAP 2, DAP 2, Racetrack 2 problems (left to right, top to bottom). The running times increase linearly in k for all problems.

5.2 The Scalability of KBI

In this experiment we investigate whether the KBI algorithm scales to large k values. We run KBI for $k = 100$ on the same set of problems, and record the elapsed times when it finishes generating the i-th best policy ($i = 2, \ldots, k$) of each problem. Figure 1 clearly shows that, for all problems KBI spends times linear in k when calculating k-th best policies. This experiment indicates that KBI has good scalability.

5.3 How k Best Policies Differ from the Optimal

We are also curious to know how the k best policies differ from the optimal policy. We analyze the list of k best policies calculated in the previous experiment, and compare the total number of different states, d, between each of these policies and the optimal policy π_1 for each problem. When d is small for a problem, it means that the k best policies are very similar to the optimal policy. This shows

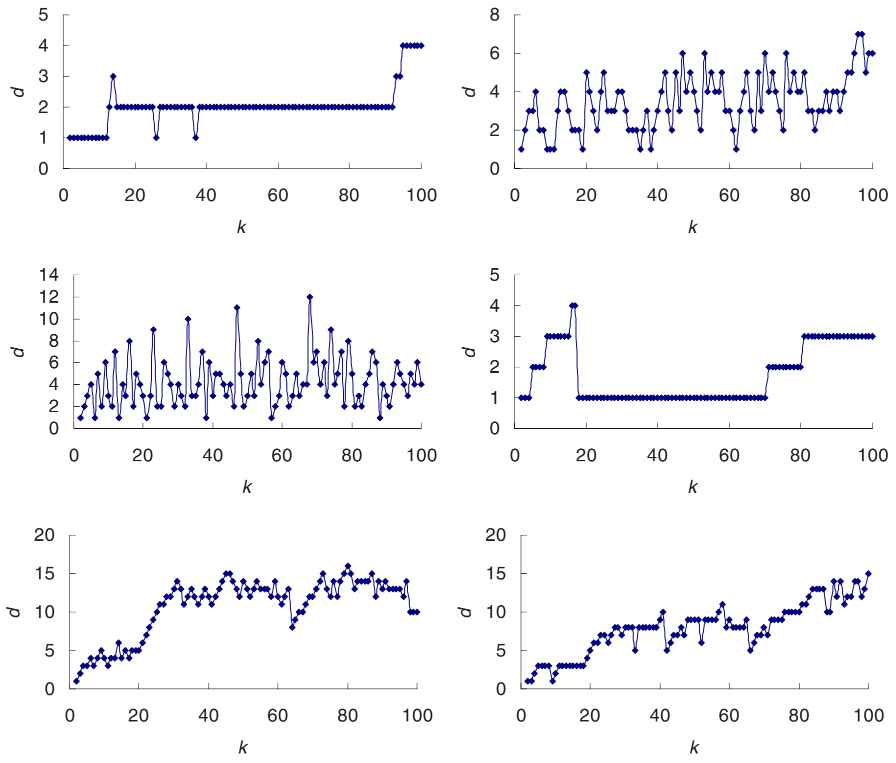

Fig. 2. The total number of different states between the k-th best policy and the optimal policy when $k = 2, \ldots, 100$ on DAP 1, Racetrack 1, SAP 1, SAP 2, DAP 2, Racetrack 2 problems (left to right, top to bottom). All k best policies are quite close to their π_1's.

that zmany good policies can be generated by a few small changes to the optimal policy. In other words, changes to few states can have very little impact on the optimality of the rest of the policy. When d is large, the optimal policy is more tightly coupled. When a sub-optimal action is chosen for a state, in order to get a good sub-optimal plan, changes to other states are usually also required.

We plot the d values for the k best policies on the same set of problems in Figure 2. These problems have relatively low d values (< 20 for all k). This shows that the k best policies are always quite close the the optimal policies. Some problems have relatively higher d values than others, namely SAP 1, DAP 2, and Racetrack 2, which means they have relatively tightly coupled optimal policies. As these problems are from diverse domains and of different sizes, it seems that the tightness of coupling of the optimal policies is probably problem-dependent.

6 Conclusions

This paper makes several contributions. First, we introduce the k best policy problem, and argue for its importance. Second, we prove a strong and useful theorem that the k-th best policy differs from some $m(< k)$-th best policy on exactly one state. Without that result, the brute-force algorithm for solving the k best policy problem (KBN) has time complexity exponential in k. Third, we propose a new algorithm, named k best policy improved (KBI), based on our theorem. We show that the time complexity of KBI is dominated by a computation linear in k. Fourth, we demonstrate that KBI outperforms KBN by an order of magnitude when $k = 2$ in most cases. The KBN algorithm does not scale to larger k values, as its running time increases exponentially in k. On the other hand, the running time of KBI increases only linearly in k. This makes KBI suitable for problems for which we want a long list of best policies. Fifth, we notice that the k best policies for different MDPs are quite similar to the optimal policies, though some problems' optimal policies are more tightly coupled than others'.

This is just the beginning of work on k best policies. There is much to be done in improving the algorithms, and in looking at applications-driven variants.

Acknowledgments

Dai was partially supported by Office of Naval Research grant N00014-06-1-0147. Goldsmith was partially supported by NSF grant ITR–0325063. We thank Mausam for helpful discussions on the problem.

References

1. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
2. Boutilier, C., Dean, T., Hanks, S.: Decision-theoretic planning: Structural assumptions and computational leverage. J. of Artificial Intelligence Research 11, 1–94 (1999)

3. Bonet, B., Geffner, H.: Planning with incomplete information as heuristic search in belief space. In: ICAPS, pp. 52–61 (2000)
4. Bresina, J.L., Dearden, R., Meuleau, N., Ramkrishnan, S., Smith, D.E., Washington, R.: Planning under continuous time and resource uncertainty: A challenge for AI. In: UAI, pp. 77–84 (2002)
5. Bresina, J.L., Jónsson, A.K., Morris, P.H., Rajan, K.: Activity planning for the mars exploration rovers. In: ICAPS, pp. 40–49 (2005)
6. Aberdeen, D., Thiébaux, S., Zhang, L.: Decision-theoretic military operations planning. In: ICAPS, pp. 402–412 (2004)
7. Musliner, D.J., Carciofini, J., Goldman, R.P., Durfee, E.H., Wu, J., Boddy, M.S.: Flexibly integrating deliberation and execution in decision-theoretic agents. In: ICAPS Workshop on Planning and Plan-Execution for Real-World Systems (2007)
8. Galand, L., Perny, P.: Search for compromise solutions in multiobjective state space graphs. In: ECAI, pp. 93–97 (2006)
9. Bryce, D., Cushing, W., Kambhampati, S.: Probabilistic planning is multiobjective! Technical Report ASU CSE TR-07-006 (June 2007)
10. Nielsen, L.R., Kristensen, A.R.: Finding the k best policies in finite-horizon mdps. European Journal of Operational Research 175(2), 1164–1179 (2006)
11. Nielsen, L.R., Pretolani, D., Andersen, K.A.: Finding the k shortest hyperpaths using reoptimization. Oper. Res. Lett. 34(2), 155–164 (2006)
12. Nielsen, L.R., Andersen, K.A., Pretolani, D.: Finding the k shortest hyperpaths. Computers & OR 32, 1477–1497 (2005)
13. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific (1996)
14. Howard, R.: Dynamic Programming and Markov Processes. MIT Press, Cambridge (1960)
15. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley, New York (1994)
16. Littman, M.L., Dean, T., Kaelbling, L.P.: On the complexity of solving Markov decision problems. In: UAI, pp. 394–402 (1995)
17. Bonet, B.: On the speed of convergence of value iteration on stochastic shortest-path problems. Mathematics of Operations Research 32(2), 365–373 (2007)
18. Barto, A., Bradtke, S., Singh, S.: Learning to act using real-time dynamic programming. Artificial Intelligence J. 72, 81–138 (1995)
19. Wingate, D., Seppi, K.D.: Prioritization methods for accelerating MDP solvers. JMLR 6, 851–881 (2005)
20. Munos, R., Moore, A.: Influence and variance of a Markov chain: Application to adaptive discretization in optimal control. In: CDC (1999)
21. Bertsekas, D.P., Tsitsiklis, J.N.: An analysis of stochastic shortest path problems. Mathematics of Operations Research 16(3), 580–595 (1991)

Appendix

In order to prove Theorem 1, we consider the effects of changing a policy one state at a time.

Lemma 2. *Let M be an MDP, and π and π' be two policies for M that differ only on state s. Suppose that $V_\pi(s) \leq V_{\pi'}(s)$. Then $V_\pi(s_0) \leq V_{\pi'}(s_0)$. More strongly, if $s \in Policydesc(s_0, \pi)$ (which implies $s \in Policydesc(s_0, \pi')$) and $V_\pi(s) < V_{\pi'(s)}$, then $V_\pi(s_0) < V_{\pi'(s_0)}$.*

Proof. We know the values of $V_\pi(s)$ and $V_{\pi'}(s)$ are two unknown constants with $V_\pi(s) \leq V_{\pi'}(s)$. We write the two systems of linear equations with respect to π and π' by ignoring variables $V_\pi(s)$ and $V_{\pi'}(s)$ on the left hand side, and replacing them with their values whenever they are on the right hand side. We find the two systems of equations have the same set of coefficients, but the one given π has smaller or equal constant values on the right hand sides. If we solve the equations by factoring out all the variables on the right hand side iteratively, the same process as replacing a variable by its corresponding state's *influence* [20], we finally get the same value for all states where s is not a policy descendant given π', since all states' influences are the same in π and π', and a better value in π for all states where s is a policy descendant given π', since the influence of s on them is decreased (due to a smaller value), where the influence of other states remain unchanged. We call this property *monotonicity of influence.* This implies $V_\pi(s_0) \leq V_{\pi'}(s_0)$. Here, we actually proved a more general result, namely that $\forall s' \in \mathcal{S}[V_\pi(s') \leq V_{\pi'}(s')]$.

Lemma 3. *Let M be an MDP, and π and π' be two policies for M that differ only on state s. Suppose that $V_\pi(s_0) < V_{\pi'}(s_0)$. Then $V_\pi(s) < V_{\pi'}(s)$. More strongly, $\forall s' \in \mathcal{S}$, $[V_\pi(s') \leq V_{\pi'}(s')]$.*

Proof (Sketch). Suppose that $V_\pi(s) \geq V_{\pi'}(s)$.

We divide the states in *Policydesc*(s_0, π') into two subsets: (1) *policy ancestors of s given π'*, the set of states where s is a policy descendant given π', and (2) *non-policy ancestors of s given π'*, the complement of (1).

We claim that the values of the non-policy ancestors of s given π' are the same as those given π. This is because the values of those states do not depend on s or any policy ancestors of s given π', so their values are not influenced by any potential value changes caused by s. For policy ancestors of s given π', their values cannot be improved, by the monotonicity of influence. Because their coefficients remain unchanged while the constants (values of non-policy ancestors of s given π' and value of s) are equal or larger. This contradicts the assumption that $V_\pi(s_0) < V_{\pi'}(s_0)$. Now, we know that $V_\pi(s) < V_{\pi'}(s)$. From Lemma 2 we have that $\forall s' \in$ *Policydesc*(s_0, π') $[V_\pi(s) \leq V_{\pi'}(s)]$.

Lemma 4. *Let M be an MDP, and π and π' be two policies for M that differ only on two states s^1 and s^2. Suppose that $V_\pi(s_0) \leq V_{\pi'}(s_0)$. Consider the following two policies π^1, π^2 obtained from by starting with π by replacing exactly one distinct action each from $\pi(s)$, $s \in \{s^1, s^2\}$, with the corresponding $\pi'(s)$. Without loss of generality, suppose $\pi^i(s^i) = \pi'(s^i)$. Then π^1 and π^2 cannot both have larger initial state values than π' does.*

Proof (Sketch). For either s^i, if s^i is not a policy descendant of s_0 given π or π', then $V_{\pi'}(s_0) = V_{\pi^i}(s_0)$, and we're done.

Now suppose $V_{\pi'}(s_0) < V_{\pi^i}(s_0)$ for $i = 1, 2$. From Lemma 3, we have

$$\forall s' \in \mathcal{S}[V_{\pi'}(s') \leq V_{\pi^1}(s')], \text{ and } V_{\pi'}(s^2) < V_{\pi^1}(s^2), \tag{2}$$

$$\forall s' \in \mathcal{S}[V_{\pi'}(s') \leq V_{\pi^2}(s')], \text{ and } V_{\pi'}(s^1) < V_{\pi^2}(s^1). \tag{3}$$

There are three cases. Case 1: Neither s^1 nor s^2 is a policy descendant of the other given π. From Equation 2 we know $V_{\pi'}(s^2) < V_{\pi^1}(s^2) = V_\pi(s^2)$, as the values of all policy descendants of s_2 given π^1 and π are the same, and $\pi^1(s^2) = \pi(s^2)$. From Equation 3 we know $V_{\pi'}(s^1) < V_{\pi^2}(s^1) = V_\pi(s^1)$ for the same reason. Then from the monotonicity of influence together with all derived inequalities, we know $V_{\pi'}(s_0) < V_\pi(s_0)$. A contradiction.

Case 2: s^2 is a policy descendant of s^1 given π, but s^1 is not a policy descendant of s^2 given π (or vice versa). From Equation 2 we first know $V_{\pi'}(s^2) < V_{\pi^1}(s^2) = V_\pi(s^2)$. From Equation 3, and $V_{\pi'}(s^2) < V_\pi(s^2)$, by the monotonicity of influence we know $V_{\pi'}(s^1) < V_\pi(s^1)$. Then, from the monotonicity of influence together with all derived inequalities, we know $V_{\pi'}(s_0) < V_\pi(s_0)$. A contradiction.

Case 3: s^1 and s^2 are both policy descendants of each other given π'. From both Equations 2 and 3 and the monotonicity of influence we can prove $V_{\pi'}(s^1) < V_\pi(s^1)$ and $V_{\pi'}(s^2) < V_\pi(s^2)$. Then from the monotonicity of influence together with all derived inequalities, we know $V_{\pi'}(s_0) < V_\pi(s_0)$. A contradiction.

Lemma 5. *Let M be an MDP, and π and π' be two policies for M that differ only on m states $s^1, s^2, \ldots, s^m$, $m > 1$. Suppose that $V_\pi(s_0) = V_{\pi'}(s_0)$. Consider the 2^m distinct policies π^T, $T \subseteq \{s^1, s^2, \ldots, s^m\}$ that agree with π on all states not in T, and agree with π' on T. Then for at least one such T of size* 1, *$V_{\pi^T}(s_0) \leq V_{\pi'}(s_0)$.*

This Lemma can be proved inductively from Lemma 5.

Note that a fundamental assumption underlying dynamic programming algorithms for MDPs is: If M is a MDP and π a non-optimal policy (in the sense of having a non-optimal value function), then there is some $s \in \mathcal{S}$ and $a \in \mathcal{A}$ such that $v_\pi(s) > C(s,a) + \gamma \sum_{s' \in \mathcal{S}} T_a(s'|s) \cdot v_\pi(s')$. Bertsekas and Tsitsiklis showed that this holds for stochastic shortest path problems, when $\gamma = 1$ [21]. Their proof can be extended.

Lemma 6. *If If $V_\pi(s_0)$ is not optimal, there must be an $s^+ \in$ Policydesc(s_0, π) and $a \in \mathcal{A}$ such that $v_\pi(s^+) > C(s^+, a) + \sum_{s' \in \mathcal{S}} T_a(s'|s^+,) \cdot v_\pi(s')$. If we let $\pi'(s) = \pi(s)$ for $s \neq s^+$, and let $\pi'(s^+) = a$, then $V_{\pi'}(s_0) < V_{\pi(s_0)}$.*

Proof (Theorem 1). Let M be an MDP, and $\Pi_i = \{\pi_1, \ldots, \pi_i\}$ be the list of i best policies, for $i \leq k$. We claim that, for $k > 1$, there is some $j < k$ and state s such that π_j differs from π_k exactly on s.

If $V_{\pi_k}(s_0) = V_{\pi_1}(s_0)$, the theorem follows from Lemma 5.

If $V_{\pi_k}(s_0) > V_{\pi_1}(s_0)$, the theorem follows from Lemma 6.