Polynomial Time Approximate Solutions to Markov Decision Processes with Many Objectives

Davis Foote
UC Berkeley
djfoote@berkeley.edu

Brian Chu
UC Berkeley
brian.c@berkeley.edu

Abstract

We study the approximation of Pareto sets of deterministic policies in finite multi-objective Markov decision processes (MOMDPs). We first show that the "exact" version of the single-objective MDP problem in which we determine whether, for a certain target T, there exists a policy which achieves expected return equal to T, is strongly NP-hard, and thus certain general results about the approximability of multiobjective problems from Papadimitriou and Yannakakis [2000] do not apply here. We then delve into ad-hoc algorithms designed specifically to approximate solutions to MOMDPs. In the first approach, we allow solutions to be lotteries over policies so that we can assume the Pareto set to be convex. With this assumption, we can use previous results by Diakonikolas and Yannakakis [2008] to find a *minimal* ϵ-Pareto set for the bi-objective case via two reductions. This suffers from limitations pervasive throughout the literature: this problem has been explored from an AI standpoint, where algorithms for approximating the Pareto set are heuristic and offer no error bound or worst-case guarantees, and it has been offhandedly mentioned as a toy example in several theory papers which give performance guarantees but consider only simple cases with two or three objectives. To the best of our knowledge, no polynomial time algorithm to approximate the general case within some error bound has ever been proposed. To this end we introduce an algorithm called ϵ-Upper Envelope Value Iteration, a relaxation of the Convex Hull Value Iteration algorithm by Barrett and Narayanan [2008]. Where the latter algorithm runs in worst-case exponential time in the number of states, ours can be shown to run in time polynomial in the size of the MDP and an error parameter (and exponential in the number of objectives). We offer an error bound for this relaxation and compare our algorithm to the two reductions.

1 Introduction

Typically, in artificial intelligence, we are concerned with designing decision-making agents that maximize a utility function over outcomes which depend probabilistically on the agent's choice of action. The "expected utility hypothesis"–that this is a complete description of rational behavior–rests on the Von Neumann-Morgenstern (VNM) utility theorem, which states that any agent with a total preference ordering over outcomes subject to some simple rules (called the "axioms of rationality") can be modeled as maximizing the expectation of a real-valued utility function defined over outcomes. For this reason, nearly all problems studied in artificial intelligence take this form.

Results in the field of multiobjective optimization (MOO) suggest an alternative approach to certain AI problems. In MOO, we consider maximization of several objectives at once. A typical AI approach to such a problem is to assert that utility must be an increasing function of all of the objectives, determine what that function must be, and return to the domain of single-objective optimization by maximizing that utility function. However, meaningful results for the multiobjective case allow us to

UC Berkeley: CS 270 Final Project.

make progress on planning when the utility function is not known. This is a noteworthy scenario: humans tend to be very bad at specifying their utility over outcomes in an abstract way, where simply identifying the most preferable out of a finite set of outcomes tends to be fairly straightforward. It would be useful, then, to have algorithms for reducing the former problem to the latter problem in decision scenarios studied in AI. This is exactly the role played by MOO.

In MOO, we are concerned with the concept of a *Pareto set*. We consider problems with multiple maximization objectives. A feasible solution s is said to be Pareto-efficient if there is no other feasible solution s' that is at least as good as s in all objectives and strictly better than s in at least one objective; that is, if s is not strictly dominated by any other feasible solution. The Pareto set for a given MOO problem is, naturally, the set of all Pareto-efficient solutions. Translating this back into the language of rationality, it is easy to see that any solution preferred by a rational agent must lie in the Pareto set.

We move to sequential decision-making and focus on a common class of problems studied in AI, Markov decision processes (MDPs). An MDP consists of a set of states, a set of actions, a transition distribution, and a reward signal. At each state, an agent must take an action which sends it to some successor state according to the transition distribution (which depends on the chosen action), and it is given a reward value for that transition. Our goal is to determine a policy mapping states to actions which maximizes the expected sum of the received reward signal. The natural extension we consider is the multiobjective Markov decision process (MOMDP), where the reward signal is a vector function of the transition rather than a scalar and we seek to maximize all components of the expected sum of received reward vectors.

2 Preliminaries

2.1 Markov Decision Processes

A Markov decision process (MDP) M is defined as a 5-tuple (S, A, P, R, γ) where S and A are the state space and action space, $P : S \times A \times S \to [0, 1]$ is the probability $P(s' \mid s, a)$ of transitioning to state s' after taking action a from state s, $R : S \times A \times S \to \mathbb{R}$ is the reward signal as a function of the transition triple (s, a, s'), and $\gamma \in [0, 1)$ is the time-discounting factor. In this paper we consider *finite* MDPs in which S and A are finite sets.

Our goal in an MDP is to maximize the expectation of the *return*, which is defined as the sum of time-discounted rewards, $\sum_{t=0}^{\infty} \gamma^t r_t$ where r_t is the reward gained after time step t. A Markov decision process is so named because it exhibits the *Markov property*: the future behavior of the system does not depend on the past given the current configuration. Thus if a^* is an optimal action from state s at time step t, it is also an optimal action from state s at any other time step t'. This means that there exists an optimal policy that is stationary and we aim to find an optimal stationary policy $\pi : S \to A$. In this paper, we consider only deterministic policies; henceforth "policy" will be assumed to refer to a stationary deterministic policy.

2.1.1 Solving MDPs

The most common approach to solving an MDP is *value iteration*. We define the Q-value of a state-action pair s, a for a policy π as the expected return of starting in state s, taking action a, and subsequently following policy π. We notate this as

$$Q^\pi(s, a) = \mathbb{E}_{r \sim \pi, P} \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s, a \right].$$

$Q^\pi(s, a)$ can be recursively defined in terms of other Q-values as

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s, a, s') + \gamma Q^\pi(s', \pi(s')) \right].$$

We denote by $Q^*(s, a)$ the Q-value of (s, a) under any optimal policy (by definition, all optimal policies have the same expected return from all states, so this is well-defined). We have the following recurrence relation for Q^*, called the Bellman optimality condition:

$$Q^*(s, a) = \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right].$$

Value iteration is an algorithm obtained by simply turning the Bellman equation into an update rule and iteratively improving until convergence.[1] Once we have obtained Q^*, it is simple to extract the optimal policy: for all $s \in S$, $\pi^*(s) = \arg\max_a Q^*(s, a)$.

Algorithm 1 Value Iteration

Input: An MDP $M = (S, A, P, R, \gamma)$.
Output: $Q^*(s, a)$, the optimal Q-function.
1: **procedure** VALUE-ITERATION(M)
2: Initialize $Q_0(s, a) = 0$ for all $(s, a) \in S \times A$
3: $t \leftarrow 0$
4: **while** not converged **do**
5: $t \leftarrow t + 1$
6: **for each** $(s, a) \in S \times A$ **do**
7: $Q_t(s, a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} [R(s, a, s') + \gamma \max_{a'} Q_{t-1}(s', a')]$
8: **return** Q_t

It is easy to see that an update to a single $Q(s, a)$ (i.e. a single execution of line 7) takes polynomial time in $|S|$ and $|A|$. A polynomial bound in $|S|$, $|A|$, and $\frac{1}{1-\gamma}$ on the number of iterations necessary for convergence is given in Littman et al. [1995]. Thus value iteration runs in polynomial time in $|S|$, $|A|$, and $\frac{1}{1-\gamma}$.

2.2 Multiobjective Optimization Problems

We use the formulation of optimization problems from Papadimitriou and Yannakakis [2000]. A single-objective optimization problem is fully specified by its set of *instances*, its set of *solutions*, and its *objective function*. For an instance x of an optimization problem (e.g. a specific MDP as an instance of the problem of choosing the best policy in an MDP), the set of solutions is the feasible set $F(x)$. For a given solution in $F(x)$, the objective function is $f(x, s)$; we seek to find the $s \in F(x)$ that maximizes $f(x, s)$. For reasons that will become important for later complexity results, we also assume that solutions are polynomially recognizable in the size of the instance.

In multiobjective optimization (MOO) problems, we instead have a vector objective function $\vec{f}(x, s) = (f_1(x, s), \ldots, f_k(x, s))$. We seek to maximize all objectives simultaneously. Since the objectives may be conflicting, it is not sufficient to choose a single solution as the optimum. How would we choose between $(2, 1)$ and $(1, 2)$? Here the language of utility theory again becomes useful: there is some scalar utility function U of all the objectives that is increasing with each component, but we do not know what it is. If a point $\vec{p}$ in objective space dominates another point by being greater than or equal to another point $\vec{q}$ in every component, then since U is increasing, $U(\vec{p}) \geq U(\vec{q})$. If $\vec{q}$ is strictly greater in at least one component than $\vec{q}$, then $U(p) > U(q)$. It follows from this that for a solution s to be optimal in an instance x under *any* increasing, it is necessary and sufficient for there to *not* exist a solution $s' \in F(x)$ such that $\vec{f}(x, s')$ dominates $\vec{f}(x, s)$. If is for this reason that we are concerned with the problem of identifying the *Pareto set* of solutions. A solution is Pareto-efficient if its objective vector is not dominated by that of any other feasible solution; the Pareto set is the set of all Pareto-efficient solutions. It follows from the preceding discussion that the Pareto set is exactly the set of optimal policies that can be optimal under some increasing utility function.

2.3 Multiobjective Markov Decision Processes

Here we introduce the problem that will be studied in this paper by combining the ideas in the two preceding subsections. A Multiobjective Markov Decision Process (MOMDP) is a 5-tuple $M = (S, A, P, \vec{R}, \gamma)$, where S, A, P, and γ are defined as in an MDP and $R : S \times A \times S \to \mathbb{R}^k$ is a vector of reward signals. Other than the fact that our rewards are now vectors rather than scalars, a

[1] Many sources call this algorithm *Q-value iteration* and use "value iteration" to refer to a similar algorithm that computes the optimal *state-value function* V^* defined as $V^*(s) = \max_a Q^*(s, a)$. Another form of the Bellman optimality condition in terms of V^* admits an algorithm nearly identical to the one described here. See Sutton and Barto [1998] for further justification of both of these algorithms.

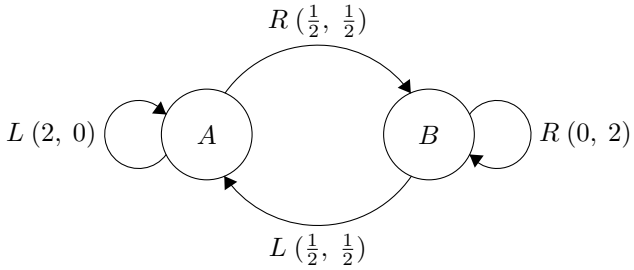
MOMDP follows the same rules as an MDP. Now, instead of a policy which maximizes the expected discounted sum of returns, we aim to find the Pareto set of policies.

To be concrete, we present a formulation of the MOMDP problem as a MOO problem. An instance of this problem is simply a specific MOMDP M. The set of feasible solutions for any $M = (S, A, P, \vec{R}, \gamma)$ is the set of all policies $\pi : S \to A$. The objective function maps M and π to the vector of expected time-discounted returns each taking the form $\mathbb{E}_{r_t \sim \pi, P}\left[\sum_{t=0}^{\infty} \gamma^t r_t^{(1)}\right]$, where $r_t^{(i)}$ is the ith componenet of the r vector received at time t.

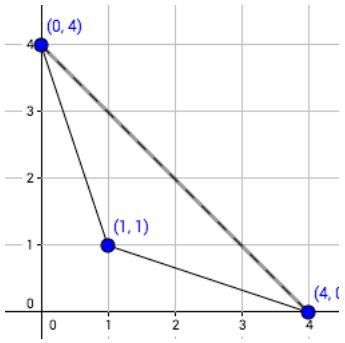
3 Attempts to Approximate MOMDPs by Standard MOO Methods

As discussed in Papadimitriou and Yannakakis [2000], for a general MOO problem the Pareto set can be exponentially large. In that paper, a new notion is introduced: ϵ-Pareto efficiency. A point p is ϵ-Pareto efficient if it is not dominated by $(1+\epsilon)q$ for some other point q.[2] An ϵ-Pareto set is a set of solutions which are not dominated by any other by a ratio of more than $1+\epsilon$. They show that given a MOO problem M and, ϵ, there exists an ϵ-Pareto set of size polynomial in $|M|$ and $\frac{1}{\epsilon}$. The paper proceeds to outline sufficient (but not necessary) conditions for the existence of a polynomial-time algorithm to compute this ϵ-Pareto set.

First note that MOMDP is inherently a combinatorial problem: for each of our $|S|$ buckets, take one of its $|A|$ elements; each different sequence of elements of A defines a distinct policy. So we must analyze it as a discrete value. As a consequence, we are left with Pareto curves (and by extension ϵ-Pareto curves) which are not necessarily convex. Consider the following example:



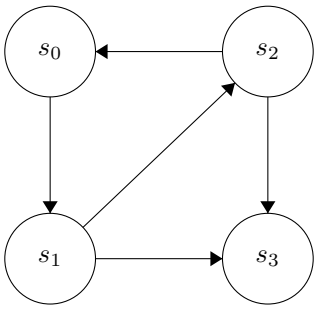
In this MOMDP, $S = \{A, B\}$, $A = \{L, R\}$, $\gamma = \frac{1}{2}$, transitions are deterministic, and the rewards for transitions are as shown in the diagram. There are $2^2 = 4$ possible policies. Leaving out a bit of detail, it can be seen that the expected return for the thrashing policy is not strictly dominated by either policy that favors one direction, but that it *is* dominated by a policy with expected return a convex combination of the expected return of the two directed policies.



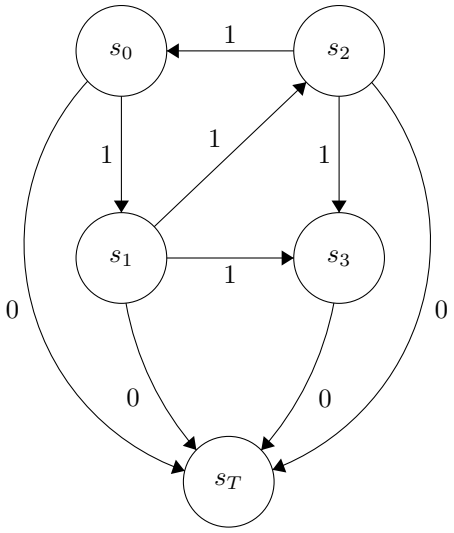
So we have that for MOMDPs, the Pareto set need not be convex in objective space.

Another difficulty arises in trying to analyze MOMDPs using the results of Papadimitriou and Yannakakis [2000]: they study objectives that are linear in the solutions (e.g. sum of edge weights

[2]Because all problems considered henceforth are multiobjective problems, we do away with the cumbersome arrow notation for vectors; it should be clear from context when we are dealing with vectors versus scalars.



(a) An instance of HAMILTONIAN-PATH



(b) An instance of EXACT-MDP with $\gamma = \frac{1}{2}, T = \frac{7}{4}$

Figure 1: An illustration of the reduction from HAMILTONIAN-PATH to EXACT-MDP

where the solution is a set of edges). The expected return in an MDP is not in any way linear as a function of the policy (it is linear as a function of the reward signal, but that is not useful here). One possibility for getting around this is restating our multiobjective problem as an optimization over objective space. Concretely, our instance M is still a MOMDP, but instead of our solution space being the set of policies, we let $F(M)$ be the set of return vectors r for which there exists some policy π that achieves exactly r. The objective function of a point in objective space is simply the identity map. The only missing nuance is a point brought up earlier: a solution must be polynomially recognizable in the size of the MOMDP, i.e. we must have a polynomial-time routine that can check feasibility of a proposed solution. In other words, given a reward vector, we must be able to determine in polynomial time whether there exists a policy achieving expected return exactly equal to r.

Suppose for a moment that such a procedure exists. Then this formulation of the problem is valid. Further, Papadimitriou and Yannakakis [2000] show that a pseudopolynomial-time algorithm for the single-objective version of this problem (call it EXACT-MDP) admits a fully polynomial-time approximation scheme for constructing an approximate Pareto curve for MOMDP. This is exactly our goal, so it would be very valuable to determine whether such an algorithm exists. Unfortunately, this is not the case.

Theorem 1. EXACT-MDP *is strongly NP-complete; that is, unless P=NP, there is no pseudopolynomial-time algorithm to solve it.*

Proof. We show this via a pseudopolynomial reduction from HAMILTONIAN-PATH. Let $G = (V, E)$. We wish to construct a pair (T, M) where $T \in \mathbb{R}$ is a target and M is an MDP with a policy that achieves expected reward exactly equal to T if and only if G has a Hamiltonian path.

Let $S = V \cup \{s_T\}$ and $A = E \cup \{(s, s_T) : s \in S\}$ (from any vertex $u \in S$, (u, v) is an available action if and only if $(u, v) \in E$ or $v = s_T$). All transitions are deterministic, and the action (u, v) taken at state u takes you to state v. All rewards are 1 on transitions between non-terminal states (states other than s_T) and are 0 on transitions to the terminal state. γ can be picked arbitrarily; for concreteness, we use $\gamma = \frac{1}{2}$. Let $T = \sum_{i=0}^{|V|-2} \gamma^i = \sum_{i=0}^{|V|-2} \frac{1}{2^i} = 2 - \frac{1}{2^{|V|-2}}$. See Figure 1 for an illustration.

Suppose there is a Hamiltonian path in G. Then the policy in M that follows the Hamiltonian path to the end then transitions to the terminal state achieves reward exactly T.

Suppose there is no Hamiltonian path in G. Since our policy must be deterministic and the environment is deterministic, if ever we visit the same non-terminal state twice, we will be stuck in an infinite loop between non-terminal states which incurs reward $\sum_{i=0}^{\infty} \frac{1}{2^i} = 2 > T$, so there is no policy that

visits the same non-terminal state twice and achieves return T. For any policy π, let n_π be the number of non-terminal states visited by following π. Then the return is equal to $\sum_{i=0}^{n_\pi - 2} \frac{1}{2^i}$, which equals T if and only if $n_\pi = |V|$. But such a π immediately admits a path visiting every state exactly once. Since there is no Hamiltonian path in G, there is no policy in M achieving return exactly T.

Since all numerical values in M as constructed are polynomially bounded in the size of the original problem $|G|$ (in fact it is constantly bounded–all numbers are 0 or 1), this reduction shows that the existence of a pseudopolynomial-time algorithm for EXACT-MDP admits a polynomial-time algorithm for HAMILTONIAN-PATH.[3] □

Not only does this deny the opportunity to derive an FPTAS for MOMDP by this strategy, it terminates this entire branch of inquiry: clearly EXACT-MOMDP is at least as hard as EXACT-MDP, so Theorem 1 implies that the feasibility problem in this formulation of MOMDP as a MOO problem cannot be solved in polynomial time (unless P=NP), rendering this formulation fundamentally flawed.

We move on from attempting to connect MOMDP to general approximability results for MOO problems and into ad-hoc algorithms constructed specifically for this problem (and reductions to other multiobjective problems for which ad-hoc algorithms have been given).

4 Minimal ϵ-Convex Pareto Sets of Policies via Reductions

To serve as an algorithmic baseline, we consider the special case of finding ϵ-convex Pareto sets for bi-objective MDPs.

We call a *convex Pareto set* the the set of solutions undominated by convex combinations of other solutions. Further, an *ϵ-convex Pareto set* is the set of solutions whose convex combinations are not dominated by more than a factor of $(1 + \epsilon)$ by other solutions.

In general, the solutions to MDPs are not convex. However, if we allow random lotteries over policies in MDPs, those solutions are convex. This is not always acceptable but it presents a relatively simple case to analyze.

Diakonikolas and Yannakakis [2008] proved that finding the optimal ϵ-convex Pareto set for bi-objective linear programming only requires solving $2k + 3$ LP's, where k is the size of the minimal ϵ-convex Pareto set, and where each LP's size is on the same order as the single-objective case.

Solving an MDP via value iteration can be reduced to a linear program, as shown by Littman et al. [1995], in $O(|S|^2|A|)$ time with $O(|S||A|)$ variables and $O(|S|)$ constraints. Thus we can easily present an upper bound on the runtime of finding the optimal ϵ-convex Pareto set for a bi-objective MDP: $O(kP + k|S|^2|A|)$, where $O(P)$ is the runtime of a linear solver. If we take that runtime to be $O(n^3 L)$, where L is the number of bits in the input, and n is the number of variables, and if we assume $|A| < |S|$, we can simplify the runtime expression to $\tilde{O}(kS^4B)$, with B as the maximum number of bits needed for any single number of the MDP.

In practice, however, linear programming is rarely used as it tends to be slower than iterative methods. This motivates the exploration of algorithms that exploit the structure of MDPs.

5 ϵ-Upper Envelope Value Iteration

We now take a different approach toward approximating the Pareto curve of policies. Barrett and Narayanan [2008] give an algorithm based on value iteration that, for each (s, a)-pair, converges to a set $\mathring{Q}^*(s, a)$ containing all the Q-vectors that are maximal according to any linear utility function over the objectives. Since the set of possible Q-vectors is a discrete set in a finite MOMDP with k objectives, the convex hull of that set is a k-dimensional polytope F. Given a particular linear utility function parametrized as a weight vector w, we can extract $Q^*_w(s, a)$, the Q-value in the MDP where, if r is the reward vector from our original MOMDP, our single objective is $w \cdot r$, by maximizing that value over all reward vectors in the polytope F. Since this is a linear objective being optimized, it is guaranteed to be optimal at a vertex of the polytope. Moreover, since this region is specified by its vertices rather than by a set of linear constraints, we can optimize a linear function over it by simply

[3] See Littman et al. [1995] for more details on strong NP-completeness.

evaluating that function at every vertex and taking the max of those results. This affords two useful observations:

1. $\mathring{Q}^*(s,a)$ is the set of vertices of the convex hull of all the Q-vectors.
2. For any linear utility function parametrized as a weight vector w,

$$Q_w^*(s,a) = \max_{q \in \mathring{Q}^*(s,a)} w \cdot q.$$

We can then rephrase the description of the algorithm as an iterative method based on value iteration that converges to the convex hull of all Q-vectors.[4] In order to account for the max in every direction, we take the union over convex hulls (and take the convex hull of the result). The pseudocode is in Algorithm 2.[5]

Algorithm 2 Convex Hull Value Iteration

Input: A MOMDP $M = (S, A, P, R, \gamma)$
Output: $\mathring{Q}^*$, the function mapping (s,a)-pairs to the convex hulls of all Q-vectors at (s,a)

procedure HULL-ITERATION(M)
  Initialize $\mathring{Q}_0(s,a)$ to $\{0\}$ for all (s,a)
  $t \leftarrow 0$
  while not converged **do**
    $t \leftarrow t+1$
    for each $(s,a) \in S \times A$ **do**
      $\mathring{Q}_t(s,a) \leftarrow \mathbb{E}_{s'}\left[\vec{R}(s,a,s') + \gamma \cdot \text{HULL}(\bigcup_{a'} \mathring{Q}_{t-1}(s',a'))\middle| s,a\right]$
  return $\mathring{Q}_t$

Unfortunately, analysis in Barrett and Narayanan [2008] reveals that since the $\mathring{Q}$ sets can get exponentially large in the worst case, this algorithm has a worst-case exponential runtime. This should be expected, as it gives us an exact solution to a multiobjective problem. We try to rectify this by introducing an approximate version of this algorithm.

5.1 The Algorithm

The first change we introduce from Barrett and Narayanan [2008] in order to better fit our treatment of the problem is to only consider increasing linear utility functions, i.e. utility functions parametrized by a weight vector with all nonnegative entries. This allows us to only consider points on the upper envelope, rather than the entire convex hull, and it makes analysis of approximation simpler. The upper envelope is simply the geometric equivalent of the Pareto set. In fact, if we consider a MOO problem where our instance is a set of points in $\mathbb{R}^k$, our solution space is the same set of points, and our objective functions are the projections of the points onto their various components, then we see that the Pareto set for this MOO problem is precisely the upper envelope of the set of points.

Next we define the ϵ-upper envelope of a set of points S as a set $H \subseteq S$ such that for all $s \in S$, s is dominated by $(1+\epsilon)h$ for some $h \in H$. Before proceeding to the ϵ-Upper Envelope Value Iteration algorithm, we need an auxiliary function that computes an ϵ-upper envelope subject to some constraints. Namely, we would like an algorithm Envelope$_\epsilon$ that, given a set $|S|$ of n non-negative points in $\mathbb{R}^k$ with at most b-bit entries, computes an ϵ-upper envelope H of S with $|H|$ bounded above by an expression that does not depend on n. The algorithm should run in time polynomial in b, $\frac{1}{\epsilon}$, and n (exponential in k is okay). Surprisingly, it appears from the literature that this is not a problem of interest to computational geometers; most work in that field on approximate convex hulls sacrifices accuracy for *speed*, not for succinctness. Algorithm 3 to solve this problem is based on an

[4] Note the distinction between the convex hull of the Q-vectors and the Pareto set. The former is a set of expected return vectors (points in objective space) and the latter is a set of policies (points in solution space).

[5] In Barrett and Narayanan [2008] a definition of addition of convex hulls is given so that taking the expectation over convex hulls makes sense.

idea from Papadimitriou and Yannakakis [2000].[6] The proof that it is correct is nearly identical to the corresponding proof in that paper, so it is omitted here.

Algorithm 3 Our algorithm to compute a bounded-size approximation to the upper envelope of a set of points.

Input: A set S of n points in $\mathbb{R}^k$.

Output: A set $H \subseteq S$ such that for any $s \in S$, some convex combination of elements in H ϵ-dominates s and $|H|$ is bounded above by an expression that does not depend on n.

1: **procedure** ENVELOPE$_\epsilon$(S)
2: $b \leftarrow$ the maximum number of bits necessary to represent any entry of any vector in S
3: $X \leftarrow$ a partitioning of the region $[1/2^b, 2^b]^k$ into hyperrectangles such that, in each dimension, the ratio of the larger to the smaller coordinate is $1 + \epsilon$
4: $H \leftarrow \emptyset$
5: **for each** hyperrectangle $x \in X$ **do**
6: Iterate over S and take the first $s \in S$ such that s is in the interior of the hyperrectangle x, if such an s exists
7: **if** such an s exists **then** $H \leftarrow H \cup \{s\}$
8: **return** H

In the original Convex Hull Value Iteration algorithm, calls to CONVEX-HULL were made in two places: the sum of hulls implicit in the expectation, and the hull of the union of Q-hulls over a'. To greatly simplify analysis, we shift focus to deterministic MOMDPs and do away with the first of those calls to CONVEX-HULL. Deterministic MOMDPs can be modeled as a special case of general MOMDPs, but to simplify notation we give them their own definition. A deterministic MOMDP is the 4-tuple $M = (S, A, R, \gamma)$ where S and γ are exactly as before, $A : S \to S$ maps states to successor states, and $R : S \times A \to \mathbb{R}^k$ no longer depends on s' (since there is only one possible value for s' anyway). We handle the other call to CONVEX-HULL by replacing it with a call to Envelope$_\epsilon$. The pseudocode is given in Algorithm 4 and a proof of its correctness is give in Section 7.

Algorithm 4 The ϵ-Upper Envelope Iteration for deterministic environments, modified from the Convex Hull Value Iteration algorithm of Barrett and Narayanan [2008].

Input: A deterministic MOMDP $M = (S, A, \vec{R}, \gamma)$.

Output: An approximation to $\mathring{Q}^*(s, a)$, a function mapping (s, a) pairs into convex hulls from which we can easily extract $Q^*_w(s, a)$ for any weight vector w over the reward components.

1: **procedure** ENVELOPE$_\epsilon$-ITERATION(M)
2: Initialize $\mathring{Q}_0(s, a) = \{0\}$ for all $(s, a) \in S \times A$
3: $t \leftarrow 0$
4: **while** not converged **do**
5: $t \leftarrow t + 1$
6: **for each** $(s, a) \in S \times A$ **do**
7: $\mathring{Q}_t(s, a) \leftarrow \vec{R}(s, a) + \gamma \cdot \text{ENVELOPE}_\epsilon(\bigcup_{a'} \mathring{Q}_{t-1}(a(s), a'))$
8: **return** $\mathring{Q}_t$

5.2 Complexity

We first analyze the complexity of Envelope$_\epsilon$. As shown in Papadimitriou and Yannakakis [2000], there are $O((2b)^k/\epsilon^k)$ hyperrectangles that partition the space. Because this is an upper bound on the size of the output that does not depend on n, we see that the algorithm works as intended. Further, since the inner loop takes $O(n)$ time, the overall time complexity of Envelope$_\epsilon$is $O(n(2b)^k/\epsilon^k)$. We can optionally run UPPER-ENVELOPE on the resulting set of size $O((2b)^k/\epsilon^k)$ in time $O((2b/\epsilon)^k \cdot k(\log 2b - \log \epsilon))$. Thus contributes little to the time complexity and empirically gives better results

[6] Ironically, that paper is largely about using geometric techniques to solve multiobjective optimization problems. Here, we use its results, which are algorithms and bounds for multiobjective optimization, to solve what is fundamentally a geometric problem.

(see Section 6), though no rigorous analysis of the effects of this alternate implementation on the asymptotic size of the returned set has been completed at this time, so we proceed without assuming this has been done.

Now, since we have an upper bound on the size of Envelope$_\epsilon$that depends only on the number of bits in any numerical entry in the set of points, our dimensionality, and our error parameter, we can bound the size of *all calls to Envelope$_\epsilon$throughout the entire course of Envelope$_\epsilon$-iteration*. Let B be the maximum number of bits needed to express any entry of any reward signal in our deterministic MOMDP M. Then the maximum entry of any Q-vector is bounded by $\frac{1}{1-\gamma} \cdot 2^B$, the return if 2^B reward is gained every time step to infinity. So no entry of any vector being passed to Envelope$_\epsilon$will have any entries larger than this, and we have $\log\left(\frac{1}{1-\gamma} \cdot 2^B\right) = B - \log(1-\gamma)$ as an upper bound on b for any call to Envelope$_\epsilon$. Thus for all calls to Envelope$_\epsilon$, the size of the returned set is in $O((2(B - \log(1-\gamma)))^k/\epsilon^k) \triangleq O(X)$. Note that X is polynomial in B and $\frac{1}{\epsilon}$, polylogarithmic in $\frac{1}{1-\gamma}$, and exponential in k.

The preceding paragraph essentially guarantees that outside the call to Envelope$_\epsilon$, we are guaranteed that everything will be small. In order to confirm that one iteration of the algorithm runs in polynomial time, we need only to verify that Envelope$_\epsilon$itself does not take exponential time. $\left|\bigcup_{a'} \mathring{Q}_{t-1}(a(s), a')\right| \leq \sum_{a'} \left|\mathring{Q}_{t-1}(a(s), a')\right| \leq \sum_{a'} O(X) \leq O(|A| \cdot X)$. Calling Envelope$_\epsilon$on a set of this size takes time $O(|A| \cdot X^2)$, which is polynomial.

All that remains is to show that we reach convergence in polynomially-many iterations; in Section 7 I show that this converges at least as fast as normal value iteration (in terms of the number of iterations). It is shown in Littman et al. [1995] that value iteration converges in polynomially-many iterations. Thus Envelope$_\epsilon$Iteration is a polynomial-time algorithm.

5.3 Error Bounds

As shown in Section 7, once projected onto any weight vector w, if $w \cdot r(s,a) + \gamma Y$ is the amount that would be backed up in a particular iteration of value iteration for $Q_w(s,a)$ where $Y = \max_{a'} Q_w(a(s), a')$, the the actual amount backed up in this direction by Envelope$_\epsilon$-Iteration is $w \cdot r(s,a) + \gamma Z$ where $Y \geq Z \geq \frac{1}{1+\epsilon} Y$. In other words, in the worst case, we back up with $Z = \frac{1}{1+\epsilon} Y$ in every iteration. If this is the case, then the value we ultimately converge to will be equal to the value normal value iteration would converge to if our discount factor were $\gamma \cdot \frac{1}{1+\epsilon}$. We seek to bound the multiplicative distance between this value and the true value. In other words, if OPT is the true optimum Q-value and ALG is the value returned by the algorithm, we wish to find a lower bound on ALG/OPT.

If our discount factor is $\gamma \cdot \frac{1}{1+\epsilon}$, the structure of the state graph in the MDP has not changed whatsoever, so it is still possible to follow the same policy we would follow if the discount factor were γ; call this policy π. Since we can do at least as well (and likely better if we were to change our policy to account for the changed discount factor) the value obtained by following π is a lower bound on the actual Q-value in the changed MDP. Since the two discount factors are being applied to the same reward signal, the problem of finding this lower bound can be stated as follows:

$$\text{Minimize } \frac{\sum_{i=0}^{\infty} \alpha^i x_i}{\sum_{i=0}^{\infty} \gamma^i x_i} \text{ over all sequences } (x_i)_{i=0}^{\infty} \text{ with } 0 \leq x_i \leq 1 \text{ for all } i$$

where $\alpha = \gamma \cdot \frac{1}{1+\epsilon}$. (Really the constraint should be over general bounded positive sequences, i.e. $0 \leq x_i \leq M$ for some M, but we can scale both sequences down by a factor of M without changing the ratio, and dealing with sequences bounded by 1 makes the analysis simpler.)

It is tempting to say that the sequence which is worst for that ratio is the all-ones sequence, but in fact we can much worse.[7] If we use a Kronecker-delta sequence Δ_i, i.e. a sequence of zeros everywhere and a one at index i, then the ratio is α^i/γ^i. Since $\alpha = \gamma \cdot \frac{1}{1+\epsilon} \leq \gamma$, we can push the ratio to be arbitrarily small. The greatest lower bound is zero.

We do, however, have a further constraint on the type of sequence to consider. This is the reward sequence from a walk in a deterministic MDP according to a stationary deterministic policy. Thus the first time a state is revisited, we are stuck in a cycle, and the reward sequence is periodic. This means that the latest the first positive reward can come (if there is any at all) is at the $|S|$th entry, after which point any suffix of the first $|S|$ elements of the sequence can repeat periodically (the particular suffix depending on which state our policy maps the final state in the walk to). Sequences of this form lead to the ratio above being equal to

$$\frac{\frac{\alpha^{|S|}}{1-\alpha^p}}{\frac{\gamma^{|S|}}{1-\gamma^p}} = \frac{\alpha^{|S|}}{\gamma^{|S|}} \cdot \frac{1-\gamma^p}{1-\alpha^p}$$

where $1 \leq p \leq |S|$ is the periodicity of the cycle we are eventually stuck in. This is minimized with $p = 1$, which is achieved in an MDP in which our policy visits all states in some permutation and ends at a terminal state in which it repeatedly follows a self-loop worth reward one. Thus the error bound is

$$\begin{aligned}
\frac{\text{ALG}}{\text{OPT}} &\geq \left(\frac{\alpha}{\gamma}\right)^{|S|} \cdot \frac{1-\gamma}{1-\alpha} \\
&= \left(\frac{1}{1+\epsilon}\right)^{|S|} \cdot \frac{1-\gamma}{1-\gamma \cdot \frac{1}{1+\epsilon}} \\
&= \left(\frac{1}{1+\epsilon}\right)^{|S|} \cdot \frac{\frac{1}{1-\gamma \cdot \frac{1}{1-\epsilon}}}{\frac{1}{1-\gamma}} \\
&= \left(\frac{1}{1+\epsilon}\right)^{|S|} \cdot \frac{(1+\epsilon)(1-\gamma)}{1+\epsilon-\gamma} \\
&= \left(\frac{1}{1+\epsilon}\right)^{|S|} \cdot \left(1 - \frac{\epsilon\gamma}{1+\epsilon-\gamma}\right).
\end{aligned}$$

6 Empirical Evaluation

We present a preliminary empirical comparison of the convex hull value iteration and ϵ-upper envelope value iteration methods.

We tested these two algorithms on a very simple bi-objective environment proposed by Vamplew et al. [2011], called the Deep Sea Treasure (DST). The environment is shown in figure 2. It consists of the agent starting at the top-left corner of the environment that seeks to optimize two objectives: 1) maximize the treasure captured, and 2) minimize the number of moves taken. The game ends when the submarine has captured a single treasure.

[7] This was an error in my initial version of the project that has since been corrected. The result stated there was a bound of $\frac{\frac{1}{1-\gamma \cdot \frac{1}{1-\epsilon}}}{\frac{1}{1-\gamma}} = \frac{(1+\epsilon)(1-\gamma)}{1+\epsilon-\gamma} = 1 - \frac{\epsilon\gamma}{1+\epsilon-\gamma}$, which is, admittedly, a much more exciting result than what I was actually able to prove in the end.

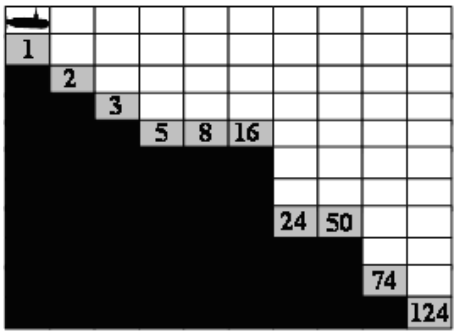


Figure 2: Deep Sea Treasure

We also attempted to implement the reduction to bi-objective linear programming summarized in Section 3, but found that there were numerous numerical issues.

Thus, in figure 3, we compare the convex hull found by convex hull value iteration to the convex hull of the ϵ-upper-envelope approximation (over Q-values at state 0, action 0). Qualitatively, it compares favorably. There are two additional empirical advantages:

1. The ϵ-upper-envelope approximation is smaller in size than the convex hull: in this example it contains 19 vectors, versus the 101 vectors in the convex hull value iteration solution.
2. The ϵ-upper-envelope approximation can be easily tuned to increase or decrease in size by increasing or decreasing ϵ.

That being said, the algorithm's runtime performance was poor in our experiments, likely because of the expensive brute-force search over hyper-rectangles when constructing the envelope. The convex hull value iteration example for this plot took 4.15 seconds to generate, whereas the ϵ-upper-envelope approximation took over a minute. Algorithmic improvements to avoid iterating over the product of the points and the hyperrectangles and writing more efficient code to match efficient library implementations of convex hull solvers seem to have the potential to bridge this gap substantially for small problems. In theory, even with the current implementations, our algorithm should surpass Convex Hull Value Iteration for problems of substantially larger size.

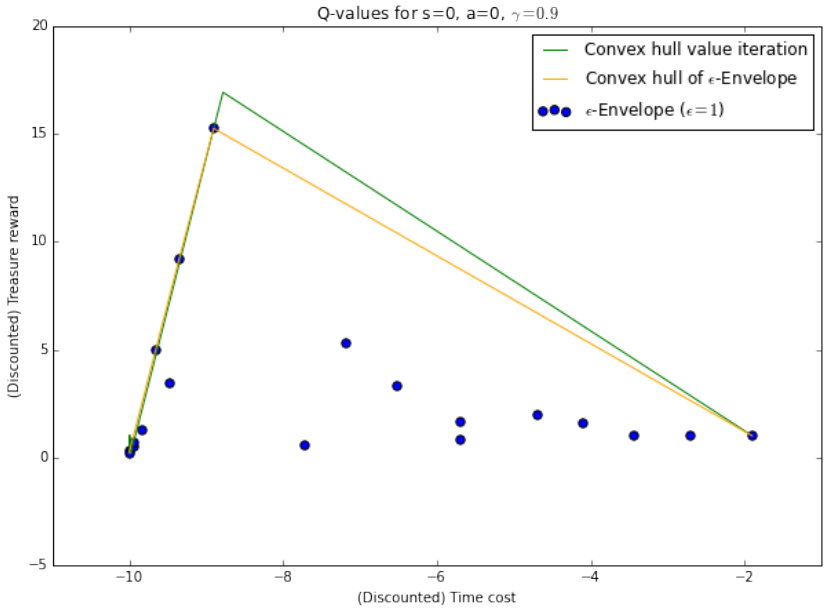


Figure 3: Deep Sea Treasure

7 Appendix: Proof of Correctness of Epsilon Upper Envelope Iteration

The structure of the first part of the proof is largely based on the proof of Convex Hull Value Iteration in Barrett and Narayanan [2008]. We prove that for any positive weight vector w, ϵ-Upper Envelope Value Iteration (henceforth "the algorithm") performs a single value iteration backup with the correct immediate reward and a correct estimate of future discounted rewards up to a multiplicative factor of $\frac{1}{1+\epsilon}$. Start by recalling the basic backup step of the algorithm:

$$\mathring{Q}_t(s,a) \leftarrow R(s,a) + \gamma \cdot \textsc{Envelope}_\epsilon \left(\bigcup_{a'} \mathring{Q}_{t-1}(a(s),a') \right).$$

The Q-value for a particular w will be extracted according to the equation

$$Q^*_w(s,a) = \max_{q \in \mathring{Q}^*(s,a)} w \cdot q.$$

as detailed in Section 5. Now apply this equation to both sides of the recurrence:

$$\begin{aligned}
Q^*_w(s,a) &\leftarrow \max \left\{ w \cdot q \,\middle|\, q \in R(s,a) + \gamma \cdot \textsc{Envelope}_\epsilon \left(\bigcup_{a'} \mathring{Q}^*(a(s),a') \right) \right\} \\
&\leftarrow \max \left\{ w \cdot q \,\middle|\, q \in \left\{ R(s,a) + \gamma q'_{a(s)} \,\middle|\, q'_{a(s)} \in \textsc{Envelope}_\epsilon \left(\bigcup_{a'} \mathring{Q}^*(a(s),a') \right) \right\} \right\} \\
&\leftarrow \max \left\{ w \cdot \left(R(s,a) + \gamma q'_{a(s)} \right) \,\middle|\, q'_{a(s)} \in \textsc{Envelope}_\epsilon \left(\bigcup_{a'} \mathring{Q}^*(a(s),a') \right) \right\} \\
&\leftarrow w \cdot R(s,a) + \gamma \max \left\{ w \cdot q'_{a(s)} \,\middle|\, q'_{a(s)} \in \textsc{Envelope}_\epsilon \left(\bigcup_{a'} \mathring{Q}^*(a(s),a') \right) \right\}.
\end{aligned}$$

The first term in this recurrence is the exact correct immediate reward term. We now seek bounds on the second term in this recurrence in terms of the correct sum-of-discounted-future-rewards term. Let $\text{OPT} = \max_{a'} Q^*_w(a(s),a')$, the correct term. Finding an upper bound on the approximate update hinges on the fact that the maximum of a positive linear objective on a set of points lies at a vertex on the upper envelope, and an ϵ-upper envelope is completely covered ("dominated" in multiobjective optimization terms) by the true upper envelope. This means that the maximum of a positive linear objective over an ϵ-upper envelope of a set of points is bounded above by the maximum over that set of points, and we have the following:

$$\begin{aligned}
Q^*_w(s,a) &\leftarrow w \cdot R(s,a) + \gamma \max \left\{ w \cdot q'_{a(s)} \,\middle|\, q'_{a(s)} \in \textsc{Envelope}_\epsilon \left(\bigcup_{a'} \mathring{Q}^*(a(s),a') \right) \right\} \\
&\leq w \cdot R(s,a) + \gamma \max \left\{ w \cdot q'_{a(s)} \,\middle|\, q'_{a(s)} \in \bigcup_{a'} \mathring{Q}^*(a(s),a') \right\} \\
&= w \cdot R(s,a) + \gamma \max \left\{ w \cdot q'_{a(s)} \,\middle|\, q'_{a(s)} \in \mathring{Q}^*(a(s),a'), a' \in A \right\} \\
&= w \cdot R(s,a) + \gamma \max_{a' \in A} \max_{q'_{a(s)} \in \mathring{Q}^*(a(s),a')} w \cdot q'_{a(s)} \\
&= w \cdot R(s,a) + \gamma \max_{a' \in A} Q^*_w(a(s),a') \\
&= w \cdot R(s,a) + \text{OPT}.
\end{aligned}$$

In order to get a lower bound, we use the property defining an ϵ-upper envelope: any point in the set is covered by an ϵ-upper envelope projected outward by a factor of $(1+\epsilon)$, which means that the

maximum of a positive linear objective over an ϵ-upper envelope is bounded below by a factor of $\frac{1}{1+\epsilon}$ times the maximum of that objective over the entire set of points.

$$
\begin{aligned}
Q^*_w(s,a) &\leftarrow w \cdot R(s,a) + \gamma \max \left\{ w \cdot q'_{a(s)} \,\middle|\, q'_{a(s)} \in \textsc{Envelope}_\epsilon \left(\bigcup_{a'} \mathring{Q}^*(a(s),a') \right) \right\} \\
&\geq w \cdot R(s,a) + \frac{1}{1+\epsilon} \cdot \gamma \max \left\{ w \cdot q'_{a(s)} \,\middle|\, q'_{a(s)} \in \bigcup_{a'} \mathring{Q}^*(a(s),a') \right\} \\
&= w \cdot R(s,a) + \frac{1}{1+\epsilon} \cdot \gamma \max \left\{ w \cdot q'_{a(s)} \,\middle|\, q'_{a(s)} \in \mathring{Q}^*(a(s),a'), a' \in A \right\} \\
&= w \cdot R(s,a) + \frac{1}{1+\epsilon} \cdot \gamma \max_{a' \in A} \max_{q'_{a(s)} \in \mathring{Q}^*(a(s),a')} w \cdot q'_{a(s)} \\
&= w \cdot R(s,a) + \frac{1}{1+\epsilon} \cdot \gamma \max_{a' \in A} Q^*_w(a(s),a') \\
&= w \cdot R(s,a) + \frac{1}{1+\epsilon} \cdot \text{OPT}.
\end{aligned}
$$

This gives the desired result.

References

Leon Barrett and Srini Narayanan. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th international conference on Machine learning*, pages 41–47. ACM, 2008.

Ilias Diakonikolas and Mihalis Yannakakis. Succinct approximate convex pareto curves. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 74–83. Society for Industrial and Applied Mathematics, 2008.

Michael L Littman, Thomas L Dean, and Leslie Pack Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 394–402. Morgan Kaufmann Publishers Inc., 1995.

Christos H Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 86–92. IEEE, 2000.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. A Bradford book. Bradford Book, 1998. ISBN 9780262193986. URL `https://books.google.com/books?id=CAFR6IBF4xYC`.

Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Mach. Learn.*, 84 (1-2):51–80, July 2011. ISSN 0885-6125. doi: 10.1007/s10994-010-5232-5. URL `http://dx.doi.org/10.1007/s10994-010-5232-5`.