Efficient Reinforcement Learning with Multiple Reward Functions for Randomized Controlled Trial Analysis

Daniel J. Lizotte† DANJL@UMICH.EDU
Michael Bowling‡ BOWLING@CS.UALBERTA.CA
Susan A. Murphy† SAMURPHY@UMICH.EDU
†University of Michigan, Ann Arbor, MI 48109 USA ‡University of Alberta, Edmonton, AB T6G 2R3 Canada

Abstract

We introduce new, efficient algorithms for value iteration with multiple reward functions and continuous state. We also give an algorithm for finding the set of all non-dominated actions in the continuous state setting. This novel extension is appropriate for environments with continuous or finely discretized states where generalization is required, as is the case for data analysis of randomized controlled trials.

1. Introduction

We begin with a motivating example. Reinforcement learning methods (Pineau et al., 2007) have been used in evidence-based medicine to analyze multi-stage randomized controlled trials (Murphy et al., 2007; Zhao et al., 2009). These trials are designed to investigate the relative effectiveness of different sequences of treatments with respect to patient outcomes. A patient's progression through the trial is divided into stages, each of which consists of random assignment to a treatment followed by monitoring of the patient's condition. The patient observations collected during each stage are very rich and commonly include several continuous variables related to symptoms, side-effects, and treatment adherence, for example.

To analyze these data using reinforcement learning methods, we consider each treatment as an action, and use the patient observations to define the resulting state and reward. Thus for the ith patient we obtain a trajectory $s_1^i, a_1^i, r_1^i, s_2^i, a_2^i, r_2^i, ..., s_T^i, a_T^i, r_T^i$. These redefined data are treated as sample trajectories from the uniform random policy. We then apply batch off-policy reinforcement learning methods to learn an optimal policy that can be used to select treatments for future patients.

One difficulty with using trial data to formulate a reinforcement learning problem is that there is no obviously correct reward function. There are many possible reward functions one could define, since each patient record includes several different measurements of that patient's overall well-being. For example, data typically include a measure of the severity of the symptoms the patient is experiencing, as well as a measure of the severity of the side-effects caused by the current treatment. These different possible reward functions are typically at odds with one another to some degree, and will therefore induce different optimal policies. For example, a policy that minimizes expected symptom measurement will tend to choose more aggressive drugs that are very effective but have a more severe side-effect profile. On the other hand, a policy that minimizes expected side-effect measurements will choose drugs that are less effective but have a milder side-effect profile.

In clinical practice, patients, doctors, and families decide on a treatment based on preferences that are not known to us at the time of data analysis. Continuing our example, these preferences may lean more toward symptom reduction or side-effect reduction, depending on the situation. However, treatment decisions are not usually influenced exclusively by one consideration or the other. *We are interested in efficient algorithms that can compute the optimal policy for a range of tradeoffs between reward functions to investigate how our tradeoff—and therefore our choice of reward function—influences the optimal policy.*

Previous work on multiple-reward problems (Barrett & Narayanan, 2008) considered a small number of discrete states in a framework where the model is known. This setting does not match our application, since we

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

have real-valued patient observations and a finite data set. In this work, we introduce a new algorithm that is asymptotically more time- and space-efficient than the Barrett & Narayanan approach, and we describe how it can be directly applied to batch data. We then show how our algorithm can be extended to the continuous-state setting using linear function approximation, a case where the previous approach is not applicable, and we give an algorithm for finding the set of all non-dominated actions in the continuous-state setting. Throughout this paper we will focus on the case of two reward functions, parameterizing the tradeoff with a scalar $\delta \in [0, 1]$. While this is sufficient to model our symptoms-versus-side-effects example, we conclude by discussing how our approach can be generalized to more than two reward functions.

2. Background

We consider sets of MDPs that all have the same state space, action space, and state transition function, but whose expected reward functions are parameterized by a scalar $\delta \in [0, 1]$. The parameter δ determines the expected reward function r as follows:

$$r(s, a, \delta) \triangleq (1 - \delta) \cdot r^{(0)}(s, a) + \delta \cdot r^{(1)}(s, a). \quad (1)$$

Each fixed δ identifies a single MDP. This parameterized expected reward function induces a corresponding parameterized optimal[1] state-action value function in the usual way via the Bellman equation:

$$Q(s, a, \delta) = r(s, a, \delta) + E_{s'|s,a}[\max_a Q(s', a, \delta)]. \quad (2)$$

We will also refer to the optimal state value function $V(s, \delta) = \max_a Q(s, a, \delta)$.

For each patient, we take an action at each timepoint $t = 1, 2, ..., T$, after which they are no longer under our care. Because we are in this finite-horizon setting, we consider a separate r_t, Q_t, and V_t function for each time-point (Bertsekas & Tsitsiklis, 1996), and we have $Q_T(s, a, \delta) \equiv r_T(s, a, \delta)$. In this framework, "value iteration" algorithms proceed by first determining the value function for time T, and then receding to the beginning of time using the recurrence $Q_t(s, a, \delta) = r_t(s, a, \delta) + E_{s'|s,a}[V_{t+1}(s', \delta)]$.

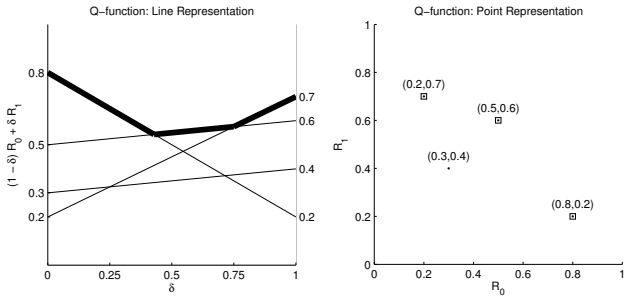
3. Value Functions for All Tradeoffs: Discrete State Space

In the discrete state-space setting, the optimal state-action value function $V_t(s, \delta)$ is piecewise linear in the tradeoff parameter δ. We use a piecewise linear spline to exactly represent $V_t(s, \delta)$ for each state and time-point, which allows us to exactly compute value backups for all δ more efficiently than the point-based representations of Barrett and Narayanan (2008). Our representation also allows identification of the set of dominated actions, i.e. the actions that are not optimal for any (s, δ) pair.

Value backups require two operations: maximization over actions, and expectation over future states. Although we are interested in settings with continuous states and linear value function approximators, the tools and concepts we need to achieve this are applicable and more easily presented in the discrete state setting. We begin by considering discrete states, and generalize our approach in Section 4.

3.1. Maximization

First, we describe how to take a function $Q_T(s, a, \cdot)$ and produce an explicit spline-representation of $V_T(s, \cdot)$ by maximizing over a. In Section 3.3, we show how this can be accomplished at earlier time-points $t < T$ using a divide-and-conquer approach.



(a) (b)

Figure 1. Computing V from Q for all δ by convex hull.

The Q-function $Q_T(s, a, \delta)$ for the last timepoint is equal to the terminal expected reward function r_T, which is linear in δ for each action as defined in (1). To represent $Q_T(s, a, \delta)$, we maintain a list of linear functions, one for each action. Figure 1(a) shows an example Q-function for a fixed state at time T. There are four actions, three of which are optimal for some δ and one which is not optimal for any δ. Each of these linear functions can be represented by a list of tradeoffs (i.e. $[0, 1]$) together with a list of their corresponding values (i.e. $[r(s, a, 0), r(s, a, 1)]$) at the tradeoff points. Each can also be represented by a point $(r(s, a, 0), r(s, a, 1))$ in the plane, as shown in Figure 1(b). These two equivalent representations offer an important conceptual and computational insight that is well-established in the multi-criterion op-

[1]Throughout this work, all Q- and V-functions are either optimal or estimates of optimal. We omit the usual * superscript throughout, and mark estimates with a hat $\hat{}$.

timization literature (Ehrgott, 2005): The set of actions that are optimal for some $\delta \in [0,1]$ are exactly those actions whose line-representations lie on the upper convex envelope of the Q-functions, and equivalently, whose point-representations lie on the upper-right convex hull of the set of Q-pairs. In general, we can recover the actions that are optimal on $\delta \in [\delta_1, \delta_2]$ by finding the upper-right convex hull of the points $\{(r(s,a_i,\delta_1), r(s,a_i,\delta_2)) : i \in \{1...|A|\}\}$. This equivalence is important because the time complexity of the convex hull operation on n points in two dimensions is $O(n \log n)$ – as fast as sorting.

We make use of this equivalence to construct our spline-representation of $V_T(s,\cdot)$. Commonly-used convex hull routines produce output that is ordered, so that it is easy to recover the list of actions that are optimal for some δ, along with the values of δ where the optimal action changes. These values are the "knots" in our spline-representation. We denote the list of knots of a piecewise linear function $f(\cdot)$ by $\Delta(f(\cdot))$. The output of a convex hull algorithm is an ordered list of points of the form $(r(s,a,0), r(s,a,1))$, which in this case is $[(0.8,0.2),(0.5,0.6),(0.2,0.7)]$. We know from the order of this list that the second knot in $V_T(s,\cdot)$ (after $\delta = 0$) occurs where the lines represented by $(0.8,0.2)$ and $(0.5,0.6)$ intersect. Thus we can compute that the line represented by $(0.8,0.2)$ is maximal from $\delta = 0$ to $\delta = 0.428..$ where it intersects the line represented by $(0.5,0.6)$. The intersection point $\delta_\times$ of two lines on the interval $[\delta_1,\delta_2]$ with end-point function values $[y_1,y_2]$ and $[z_1,z_2]$ is $\delta_\times = \delta_1 + (\delta_2 - \delta_1)\cdot(y_1 - z_1)/(y_1 - y_2 + z_2 - z_1)$.

After finding the knots, we represent the piecewise linear value function in Figure 1(a) by the knot-list $\Delta(V_T(s,\cdot)) = [0.00, 0.428.., 0.75, 1.00]$ and value-list $[0.80.., 0.54.., 0.58.., 0.70..]$, rather than by the list of points. This allows us to more efficiently evaluate $V(s,\delta) = \max_a Q(s,a,\delta)$: To evaluate $V(s,\delta)$, we use binary search to find the largest knot less than δ. This tells us which line is maximal for this δ. We then evaluate only the maximal line at δ, so that evaluating $V(s,\cdot)$ takes $O(\log|\Delta(V(s,\cdot))|)$ time, i.e. the time for the cost of the search, rather than the $O(|\Delta(V(s,\cdot))|)$ time it would take to maximize over all lines at δ.

3.2. Expectation

We now demonstrate how to efficiently compute a spline-representation of $Q_{T-1}(s,a,\delta) = r_{T-1}(s,a,\delta) + E_{s'|s,a}[V_T(s',\delta)]$ using the spline-representation of V_T. To do so, we must evaluate expectations of V_T over sets of possible future states.

Consider a two-state example where we take the expectation $E_{s'|s,a}[V_T(s',\delta)]$. Suppose there are two reachable states s'_1 and s'_2, and that the probability of arriving in state s'_i is given by θ_i. Since each $V_T(s'_i,\delta)$ is linear over the intervals between its own knots, these two functions are simultaneously linear over the intervals between $\Delta(V_T(s'_1,\cdot)) \cup \Delta(V_T(s'_2,\cdot))$, and their average is linear over the same intervals. Therefore this expectation is itself a piecewise linear function of δ with knot-list $\Delta(V_T(s'_1,\cdot)) \cup \Delta(V_T(s'_2,\cdot))$, and we can compute its spline-representation by constructing the value-list for $E_{s'|s,a}[V_T(s',\delta)]$. The new value-list is

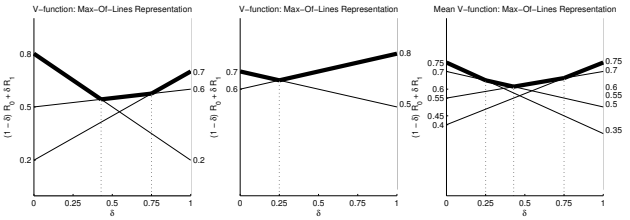
$$\left[\sum_i \theta_i V_T(s'_i,\delta) : \delta \in \Delta(V_T(s'_1,\cdot)) \cup \Delta(V_T(s'_2,\cdot))\right]. \tag{3}$$

Let $d_i = |\Delta(V_T(s'_i,\cdot))|$. This construction uses $O(d_1 + d_2)$ space and requires $O(d_1 + d_2)$ evaluations of V_T.

We contrast this with the approach of Barrett and Narayanan (2008). The expectation can also be computed using the point-based representation in Figure 1(b): Let Ψ_i be the set of points in the point-based representation of $V(s'_i,\cdot)$. One can compute an expected value function by constructing a set of points

$$\begin{aligned} &\{(\theta_1 a_1, \theta_1 b_1) + (\theta_2 a_2, \theta_2 b_2)\} \\ &\text{s.t. } (a_1,b_1) \in \Psi_1, (a_2,b_2) \in \Psi_2 \end{aligned} \tag{4}$$

and then taking the upper-right portion of the convex hull of this set. Recovering the knot-list and value-list will result in exactly the same function as the method described above. Barrett and Narayanan (2008) advocate this procedure and prove its correctness; however, they note that the set given in (4) has $|\Psi_1||\Psi_2|$ points that must be constructed and fed into the convex hull algorithm. Since $d_i = |\Psi_i| + 1$, computing the expectation in this way will take $O(d_1 d_2)$ space and $O(d_1 d_2 \log d_1 d_2)$ time, which is much less efficient than our $O(d_1 + d_2)$ spline-representation based approach.



(a) (b) (c)

Figure 2. Computing expectations using unions of spline-representations. Graphs (a) and (b) show value functions in terms of δ at two different states. Graph (c) shows the expected value function if the two states are each reached with probability 0.5.

3.3. Value Backups for $t < T-1$

The maximization procedure in Section 3.1 relies on the linearity of $Q_T(s,a,\cdot)$. However, for $t < T$, $Q_t(s,a,\cdot)$ is non-linear in general. We now show how to compute V_t and Q_t from V_{t+1} by decomposing $Q_t(s,a,\cdot)$ into linear pieces and applying the expectation and maximization operations to each piece. Recall

$$Q_t(s,a,\delta) = r_t(s,a,\delta) + E_{s'|s,a}[V_{t+1}(s',\delta)]. \quad (5)$$

For a fixed state s and action a at time T, we have shown by construction that $E_{s'|s,a}[V_T(s',\cdot)]$ is convex and piecewise linear when we have discrete states. By definition, $r_t(s,a,\delta)$ is linear in δ for all s,a,t. Thus their positive-weighted sum, i.e. Q_{T-1}, is convex and piecewise linear in δ. It follows by induction that $Q_t(s,a,\delta)$ is convex piecewise linear in δ for all $t \in 1,...,T$. To compute Q_t from V_{t+1}, we first identify the knots in $E_{s'|s,a}[V_{t+1}(s',\delta)]$ and store them; this is done in the same way as for $t+1=T$. We then compute the sum of the expected future value plus the immediate expected reward evaluated at each of the stored knots. (The immediate expected reward is linear in δ and so does not contribute additional knots.) To compute $V_t(s,\cdot) = \max_j Q(s,a^j,\cdot)$, we take the maximum over actions of these piecewise linear Q-functions using Algorithm 2. First, we decompose the problem of finding $\max_j Q_t(s,a^j,\cdot)$ for $\delta \in [0,1]$ into sub-problems of finding $\max_j Q_t(s,a^j,\cdot)$ over intervals of δ where we know the $Q_t(s,a^j,\cdot)$ are simultaneously linear. The ends of these intervals are given by $\bigcup_j \Delta(Q_t(s,a^j,\cdot))$. Next, we apply the convex hull algorithm to each of these intervals to recover any additional knots in $\max_i Q_t(s,a^j,\cdot)$.

The full backup procedure is described in Algorithm 1. In the case where the state transition model of the MDP is not known, note that we can estimate $P(s'|s,a)$ from data and the algorithm remains unchanged. Also, in practice, we can avoid running the convex hull algorithm over every interval by checking each interval's end points: If for some j we find that $Q_t(s,a^j,\cdot)$ is maximal at both ends of an interval over which the $Q_t(s,a^j,\cdot)$ are all linear, then $\max_j Q_t(s,a^j,\cdot)$ has no knots inside the interval.

3.4. Complexity of $Q_t(s,a,\cdot)$ and $V_t(s,\cdot)$

Suppose there are $|S|$ states and $|A|$ actions. For any fixed s and a, the final Q-function $Q_T(s,a,\cdot)$ has 2 knots, $\delta=0$ and $\delta=1$. Applying Algorithm 2 to these functions generates at most $|A|-1$ new internal knots, and therefore $V_T(s,\cdot)$ has at most $(|A|-1)+2$ knots. To compute $Q_{T-1}(s,a,\cdot)$, we take the expectation of $V_T(s,\cdot)$ over future states. Since $V(s,\cdot)$ might have different internal knots for every s, $Q_{T-1}(s,a,\cdot)$ may have as many as $|S|(|A|-1)+2$ knots. However, for a fixed s, these knots will be the same for all a. Thus, computing $V_{T-1}(s,\cdot)$ using Algorithm 2 adds at most $|A|-1$ new knots between each pair of existing knots, for a total of $(|A|-1)(|S|(|A|-1)+1)+2$. In general, $Q_t(s,a,\cdot)$ may have $O(|S|^{T-t}|A|^{T-t})$ knots, and $V_t(s,\cdot)$ may have $O(|S|^{T-t}|A|^{(T-t)+1})$ knots.

To compute the expectation $E_{s'|s,a}[V_{t+1}(s',\delta)]$ at time t, our approach requires $O(|S|^{T-t}|A|^{(T-t)+1})$ for each state, for a total of $O(|S|^{(T-t)+1}|A|^{(T-t)+1})$ time. In contrast, the Barrett & Narayanan approach requires $O(|S|^{2\cdot(T-t)+1}|A|^{2\cdot(T-t)+1}\log|S|^{2\cdot(T-t)+1}|A|^{2\cdot(T-t)+1})$ for each of $\log_2|S|$ pairs of piecewise linear functions.

3.5. Identifying Non-Dominated Actions

We have shown how to exactly represent the V and Q functions for all s, a, and δ at all time points when

Algorithm 1 Value Backup - Finite State Space

```
/* A ∪← B means A ← A ∪ B */
∀(s,δ), V_{T+1}(s,δ) ≜ 0. ∀s, Δ(V_{T+1}(s,·)) ≜ {0,1}.
for t = T downto 1 do
  for all s ∈ S do
    for all a ∈ A do
      Δ(Q_t(s,a,·)) ← {}
      for all s' ∈ S do
        Δ(Q_t(s,a,·)) ∪← Δ(V_{t+1}(s',·))
      end for
      for all δ ∈ Δ(Q(s,a,·)) do
        Q_t(s,a,δ) ← r(s,a,δ) +
                     Σ_{s'} P(s'|s,a) · V_{t+1}(s',δ)
      end for
    end for
    Compute Δ(V_t(s,·)) by applying Algorithm 2
    to Q_t(s,a,·), a ∈ A
  end for
end for
```

Algorithm 2 Max of Convex Piecewise Linear Fns.

```
/* A ∪← B means A ← A ∪ B */
input Convex piecewise linear functions
      f_i(·), i = 1..k defined on [δ_0, δ_1].
Δ^all = ⋃_{i=1}^k Δ(f_i(·))
Δ^out = Δ^all
for i = 2 to |Δ^all| do
  if argmax_j f_j(Δ^all_{i-1}) ≠ argmax_j f_j(Δ^all_i) then
    Δ^out ∪← Δ(max_j f_j(δ), δ ∈ (Δ^all_{i-i}, Δ^all_i))
  end if
end for
```

states are discrete, and we have shown how to identify all actions that are optimal for some δ at a particular s. This representation allows us to identify which actions are non-dominated for any δ by enumerating s.

4. Value Functions for All Tradeoffs: Linear Function Approximation

When analyzing multistage clinical trial data, assuming a small or finite state space is unreasonable. For example, patient observations will be many-valued if they come from a survey instrument that measures symptom severity, or real-valued if they come from a lab test that measures an enzyme level. Furthermore, we typically have a limited number of trajectories, and we therefore need to generalize across states in order to estimate state-action values without overfitting. Here, we demonstrate how our previously developed algorithms for value backups over all tradeoffs can be extended to the case where we have a continuous state variable and a value function estimated using linear value function approximation. We also show how to efficiently identify all non-dominated actions in this setting. Again, we begin by considering the value at time T, which has the simplest form, and later describe how to compute value functions at earlier timepoints.

Suppose that $r_T(s,a,0)$ and $r_T(s,a,1)$ are each linear functions of s. At time T, from (1), we have

$$Q_T(s,a,\delta) = (1-\delta)\cdot(\beta_{00}^{aT}+\beta_{01}^{aT}s)+\delta\cdot(\beta_{10}^{aT}+\beta_{11}^{aT}s) \quad (6)$$

where the coefficients $\beta_{00}^{aT},\beta_{01}^{aT},\beta_{10}^{aT},\beta_{11}^{aT}$ define the state-action value function at time T. For each action, $Q_T(\cdot,a,\cdot)$ is bilinear in s and δ, and $V_T(\cdot,\cdot) = \max_a Q_T(\cdot,a,\cdot)$ is piecewise bilinear in s and δ.

Recall we have a set of N trajectories of the form $s_1^i, a_1^i, r_1^{(0)i}, r_1^{(1)i}, s_2^i, a_2^i, r_2^{(0)i}, r_2^{(1)i}, ..., s_T^i, a_T^i, r_T^{(0)i}, r_T^{(1)i}$. To estimate $Q_T(s,a,0)$ and $Q_T(s,a,1)$ using ordinary least-squares regression, we find the N_T^a trajectories in our dataset where $a_T = a$, and we construct a design matrix and target vectors

$$X_T^a = \begin{bmatrix} 1 & s_T^1 \\ 1 & s_T^2 \\ \vdots & \vdots \\ 1 & s_T^{N_T^a} \end{bmatrix}, \ \mathbf{r}_T^{a,(j)} = \begin{bmatrix} r_T^{(j)1} \\ r_T^{(j)2} \\ \vdots \\ r_T^{(j)N_T^a} \end{bmatrix} \quad (7)$$

for $j=0$ and $j=1$. We then estimate parameters

$$[\hat{\beta}_{00}^{aT},\hat{\beta}_{01}^{aT}]^\mathsf{T} = (X_T^{a\mathsf{T}}X_T^a)^{-1}X_T^{a\mathsf{T}}\mathbf{r}_T^{a,(0)} \quad (8)$$

$$[\hat{\beta}_{10}^{aT},\hat{\beta}_{11}^{aT}]^\mathsf{T} = (X_T^{a\mathsf{T}}X_T^a)^{-1}X_T^{a\mathsf{T}}\mathbf{r}_T^{a,(1)}. \quad (9)$$

These estimated parameters are then substituted into definition (6), giving $\hat{Q}_T(s,a,0)$ and $\hat{Q}_T(s,a,1)$. To construct any other estimate $\hat{Q}_T(s,a,\delta)$, we could construct a scalar reward using $\mathbf{r}_T^{(0)}$, $\mathbf{r}_T^{(1)}$, and δ, and solve

$$[\hat{\beta}_{\delta 0}^{aT},\hat{\beta}_{\delta 1}^{aT}]^\mathsf{T} = (X_T^{a\mathsf{T}}X_T^a)^{-1}X_T^{a\mathsf{T}}((1-\delta)\mathbf{r}_T^{(0)}+\delta\mathbf{r}_T^{(1)}) \quad (10)$$

$$= (1-\delta)[\hat{\beta}_{10}^{aT},\hat{\beta}_{11}^{aT}]^\mathsf{T} + \delta[\hat{\beta}_{10}^{aT},\hat{\beta}_{11}^{aT}]^\mathsf{T}. \quad (11)$$

Thus we need to solve for the regression coefficients only at $\delta=0$ and $\delta=1$, after which we can produce $\hat{Q}_T(s,a,\delta)$ for any δ by combining these coefficients. Therefore, for $t=T$, it is straightforward to exactly solve for the $\hat{Q}_T$ we would estimate for every state s and every tradeoff δ simultaneously.

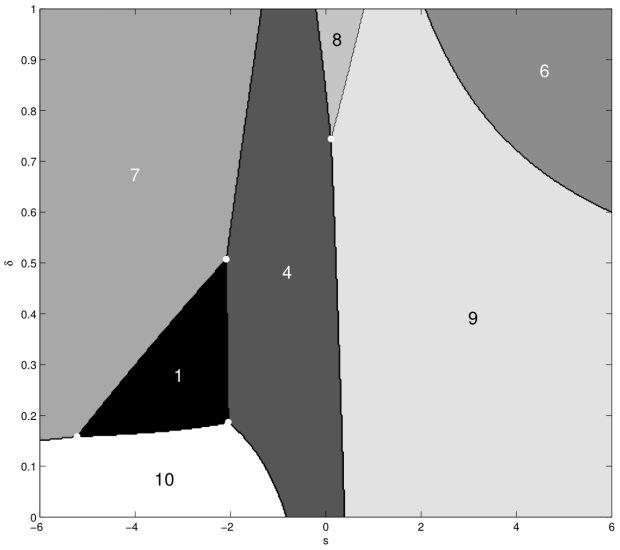


Figure 3. Diagram of the regions in (s,δ) space where different actions are optimal at time T. In this example, the state space is $\{s \in [-6,6]\}$.

4.1. Maximization

For any fixed value of s and any action a, $\hat{Q}_T(s,a,\delta)$ is a linear function of δ, and we can use the convex hull method to identify the actions that maximize value, as well as to recover the knots in the piecewise linear $\hat{V}_T(s,\cdot)$. Figure 3 is an illustration of the pieces of a hypothetical $\hat{V}_T$ that is a maximization over 10 actions. Each number in the figure marks the region where that action is optimal at time T. For example, a vertical slice at $s=-4$ of the value function has three linear pieces where actions 10, 1, and 7 are optimal.

Whereas in the discrete case we represent $V_T(s,\cdot)$ separately for each nominal state s in the MDP, in the continuous case we represent $\hat{V}_T(s,\cdot)$ for each *observed*

state $s_T^1, ..., s_T^N$, i.e. we represent a one-dimensional vertical slice of the value function for each of the s_T^i. For $i \in 1, ..., N$, we apply Algorithm 2 to construct a spline-representation for $\hat{V}_T(s_T^i, \cdot)$.

4.2. Regression of Value on s

At stage $T-1$, the parameters of our estimate $\hat{Q}_{T-1}(s, a, \delta)$ are formed as follows:

$$[\hat{\beta}_{\delta 0}^{a(T-1)}, \hat{\beta}_{\delta 1}^{a(T-1)}]^\mathsf{T} = (X_{T-1}^a{}^\mathsf{T} X_{T-1}^a)^{-1} X_{T-1}^a{}^\mathsf{T} \mathbf{y}_{T-1}^{a,(\delta)} \tag{12}$$

where

$$\mathbf{y}_t^{a,(\delta)} = ((1-\delta)\mathbf{r}_t^{a,(0)} + \delta \mathbf{r}_t^{a,(1)}) + \hat{\mathbf{v}}_{t+1}^a(\delta) \tag{13}$$

and

$$\hat{\mathbf{v}}_t^a(\delta) = \begin{bmatrix} \hat{V}_t(s_t^1, \delta) \\ \hat{V}_t(s_t^2, \delta) \\ \vdots \\ \hat{V}_t(s_t^{N_{t-1}^a}, \delta) \end{bmatrix} \tag{14}$$

for $t \in \{1, ..., T\}$. Here the s_T^i are taken from trajectories that match $a_{T-1}^i = a$. The components of the vector $\mathbf{y}_{(T-1)}^{a,(\delta)}$ are not linear in δ, so for $t < T$, solving the regression only for $\delta = 0$ and $\delta = 1$ does not completely determine $\hat{Q}_t(s, a, \cdot)$. However, the components of $\mathbf{y}_{(T-1)}^{a,(\delta)}$ are each piecewise linear in δ. We determine the intervals over which the components are simultaneously linear and then explicitly represent the state-value function at the endpoints of these intervals. This procedure is analogous to that of Section 3.2, but in the continuous case, expectation is replaced by regression. The output is a list of knots together with a list of parameter pairs $(\beta_{\delta 0}^{a(T-1)}, \beta_{\delta 1}^{a(T-1)})$, each given by $(X_{(T-1)}^a{}^\mathsf{T} X_{(T-1)}^{a(T-1)})^{-1} X_{(T-1)}^a{}^\mathsf{T} \mathbf{y}_{(T-1)}^{a,(\delta)}$.

4.3. Value Backups for $t < T-1$

Section 4.2 relies on the linearity of $\hat{Q}_T(s, a, \cdot)$. However, for $t < T$, $\hat{Q}_t(s, a, \cdot)$ is non-linear in general. We now demonstrate how to compute $\hat{V}_t$ and $\hat{Q}_t$ from $\hat{V}_{t+1}$ when $t < T-1$. To compute $\hat{Q}_t(s, a, \delta)$ from $\hat{V}_{t+1}$, we first identify the intervals of δ where the components of $\hat{\mathbf{y}}_t^{a,(\delta)}$ are simultaneously linear and store them; this is done in the same way as for $t+1 = T$. We then compute the regression coefficients of $\mathbf{y}_t^{a,(\delta)}$ on X_t^a as in (12) for each δ in the stored knots, resulting in a piecewise bilinear function $\hat{Q}_t(s, a, \delta)$. To then compute $\hat{V}_t(s_t^i, \cdot) = \max_j \hat{Q}_t(s_t^i, a^j, \cdot)$ for each s_t^i in our dataset, we apply Algorithm 2 for each s_t^i. This entire procedure is described in Algorithm 3.

Algorithm 3 Value Backup - Infinite State Space

$\forall (s, \delta),\ \hat{V}_{T+1}(s, \delta) \triangleq 0.\ \forall s,\ \Delta(\hat{V}_{T+1}(s, \cdot)) \triangleq \{0, 1\}.$
for $t = T$ **downto** 1 **do**
  for all $a \in A$ **do**
    $\Delta_a^{\hat{Q}_t} \leftarrow \{\}$
    for all $(s_t, a_t, s_{t+1}) \in \mathcal{D}$ s.t. $a_t = a$ **do**
      $\Delta_a^{\hat{Q}_t} \leftarrow \Delta_a^{\hat{Q}_t} \cup \Delta(\hat{V}_{t+1}(s_{t+1}, \cdot))$
    end for
    for all $\delta \in \Delta_a^{\hat{Q}_t}$ **do**
      $\mathbf{y}_t^{a,(\delta)} = ((1-\delta)\mathbf{r}_t^{a,(0)} + \delta \mathbf{r}_t^{a,(1)}) + \hat{\mathbf{v}}_{t+1}^a(\delta)$
      $[\hat{\beta}_{\delta 0}^{at}, \hat{\beta}_{\delta 1}^{at}]^\mathsf{T} = (X_t^a{}^\mathsf{T} X_t^a)^{-1} X_t^a{}^\mathsf{T} \mathbf{y}_t^{a,(\delta)}$
    end for
  end for
  for all $s_t^i \in \mathcal{D}$ **do**
    Compute $\Delta(\hat{V}_t(s_t^i, \cdot))$ by Algorithm 2
  end for
end for

4.4. Non-convexity of $\hat{Q}_t(\cdot, a^j, \cdot)$

Note that for $t < T$, the resulting $\hat{Q}_t(\cdot, a^j, \cdot)$ are not necessarily convex in δ. In the discrete case, we know by construction that the $Q_T(s, a, \cdot)$ are linear and that therefore $V_T(s, \cdot)$ is convex and piecewise linear. It follows that each $Q_{T-1}(s, a, \cdot)$ is also convex because each is a positive weighted sum of the convex $V_T(s, \cdot)$. In the regression case, the $\hat{Q}_{T-1}(s, a, \cdot)$ are a weighted sum of the $\mathbf{y}_{T-1}^{a,(\delta)}$ which depend on $\hat{V}_T(s_T^i, \cdot)$:

$$\hat{Q}_{T-1}(s, a, \cdot) = [1, s] \cdot (X_{T-1}^a{}^\mathsf{T} X_{T-1}^a)^{-1} X_{T-1}^a{}^\mathsf{T} \mathbf{y}_{T-1}^{a,(\delta)} \tag{15}$$

$$= \mathbf{w}(s)^\mathsf{T} \cdot \mathbf{y}_{T-1}^{a,(\delta)}. \tag{16}$$

Here, $\mathbf{w}(s)$ is a vector that depends on s and on the data, but does it not depend on δ. Elements of $\mathbf{w}(s)$ can be negative. Therefore, although each element of $\mathbf{y}_{T-1}^{a,(\delta)}$ is convex and piecewise linear in δ, the $\hat{Q}_t(s, a, \cdot)$ may not be convex for $t < T$. This non-convexity means that both the algorithm by Barrett & Narayanan (2008), as well as important algorithms from the POMDP literature (e.g. Pineau et al. 2003) that operate on convex piecewise linear value functions, are not directly applicable in our setting.

4.5. Complexity of $\hat{Q}_t(s, a, \cdot)$ and $\hat{V}_t(s, \cdot)$

Suppose there are N trajectories and $|A|$ actions. The terminal estimated Q-function $\hat{Q}_T(s, a, \cdot)$ has two knots, one at $\delta = 0$ and one at $\delta = 1$. The terminal value function $\hat{V}_T(s_T^i, \cdot)$ is constructed at N points by applying Algorithm 2 to the $\hat{Q}_T(s_T^i, a, \cdot)$ for each observed state $s_T^1, s_T^2, ..., s_T^{N_T^a}$. Each result-

Table 1. Knot counts and timings for computing $\hat{Q}(s,a,\cdot)$. Results are over 1000 randomly generated datasets using $N = 1290$, $|A| = 3$, $T = 3$.

	Min	Med	Max	Bound
Knots in $\hat{Q}_2$	687	790	910	3870
Knots in $\hat{Q}_1$	2814	3160	3916	$\approx 1.5 \cdot 10^7$
Time (s) for $\hat{Q}_2$	3.17	3.26	3.44	-
Time (s) for $\hat{Q}_1$	5.46	5.73	6.55	-

ing $\hat{V}_T(s_T^i,\cdot)$ has at most $|A|-1$ new internal knots, and therefore at most $(|A|-1)+2$ knots. To compute $\hat{Q}_{T-1}(\cdot,a,\cdot)$, we use regression with targets constructed from $\hat{V}_T(s_T^i,\cdot)$, where the observed states s_T^i come from tuples where $a_{T-1}^i = a$. There are N_{T-1}^a such observed states. Thus each $\hat{Q}_{T-1}(\cdot,a,\cdot)$ has at most $N_{T-1}^a(|A|-1)+2$ knots. The union of their knot-lists has $(\sum_{a\in A} N_{T-1}^a(|A|-1))+2 = N(|A|-1)+2$ knots. Computing $\hat{V}_{T-1}(s_{T-1}^i,\cdot)$ using Algorithm 2 adds at most $|A|-1$ new knots between each pair of knots in the union, for a total of $(|A|-1|)(N(|A|-1)+1)+2$ knots. In general, $\hat{Q}_t(s,a,\cdot)$ may have up to $O(N^{T-t}|A|^{T-t})$ knots, and $\hat{V}_t(s,\cdot)$ may have up to $O(N^{T-t}|A|^{(T-t)+1})$ knots. To compute the expectation described in Section 4.2 at time t, our approach requires $O(N^{T-t}|A|^{(T-t)+1})$ for each trajectory, for a total of $O(N^{(T-t)+1}|A|^{(T-t)+1})$ time.

To show that our approach is computationally practical, we construct an example that is representative of the clinical trial data we encounter. We use 1290 trajectories across three timepoints, and three actions per time point. This mimics a simplified version of STAR*D, of one of the largest randomized trial datasets currently available (Rush et al., 2004). Table 1 shows the number of knots and the computation time to represent $\hat{Q}(s,a,\cdot)$. Experiments were run on 1000 simulated datasets using Matlab 2009a on an 8-processor 3.0 GHz Xeon machine. In our example, the actual number of knots needed to represent a single $\hat{Q}_t(s,a,\cdot)$ was much lower than the worst case bound. Computation time was on the order of seconds and thus feasible for our purposes.

4.6. Identifying Non-Dominated Actions

We now show how to identify all of the non-dominated actions in the linear function approximation setting. In our Figure 3 example, only actions 1, 4, 6, 7, 8, 9, and 10 are represented in the diagram because it happens that these are the only actions that are non-dominated, i.e. that are estimated to be optimal for some (s,δ) pair. Actions 2, 3, and 5 were not estimated to be optimal for any combination of s and δ. We begin by considering sets of dominated and non-dominated actions at time T. (To save space we will not always write the T subscript.) To analytically identify these sets of actions, we analyze the boundaries between the bilinear regions where one action has higher value than another. These boundaries occur where $Q_T(\cdot,a_1,\cdot) = Q_T(\cdot,a_2,\cdot)$ for various actions a_1 and a_2, i.e. where

$$\begin{aligned}(1-\delta)\cdot(\beta_{00}^{a_1}+\beta_{01}^{a_1}s)+\delta\cdot(\beta_{10}^{a_1}+\beta_{11}^{a_1}s) = \\ (1-\delta)\cdot(\beta_{00}^{a_2}+\beta_{01}^{a_2}s)+\delta\cdot(\beta_{10}^{a_2}+\beta_{11}^{a_2}s)\end{aligned} \tag{17}$$

which describes the hyperbola

$$\delta = \frac{(\beta_{00}^{a_2}+\beta_{01}^{a_2}s)-(\beta_{00}^{a_1}+\beta_{01}^{a_1}s)}{\left[\begin{array}{c}(\beta_{10}^{a_1}+\beta_{11}^{a_1}s)-(\beta_{00}^{a_1}+\beta_{01}^{a_1}s)+ \\ (\beta_{00}^{a_2}+\beta_{01}^{a_2}s)-(\beta_{10}^{a_2}+\beta_{11}^{a_2}s)\end{array}\right]}. \tag{18}$$

Along these boundaries, "triple-points" occur at (s,δ) points where three or more actions are simultaneously optimal. Consider three actions a_1, a_2, a_3. The triple points for these actions occur at s values given by

$$as^2+bs+c=0 \tag{19}$$

where

$$\begin{aligned}a &= (\beta_{01}^{a_2}-\beta_{01}^{a_1})(\beta_{11}^{a_1}-\beta_{01}^{a_1}+\beta_{11}^{a_3}-\beta_{01}^{a_3})- \\ &\quad (\beta_{01}^{a_3}-\beta_{01}^{a_1})(\beta_{11}^{a_1}-\beta_{01}^{a_1}+\beta_{11}^{a_2}-\beta_{01}^{a_2}) \\ b &= (\beta_{01}^{a_2}-\beta_{01}^{a_1})(\beta_{10}^{a_1}-\beta_{00}^{a_1}+\beta_{10}^{a_3}-\beta_{00}^{a_3})- \\ &\quad (\beta_{01}^{a_3}-\beta_{01}^{a_1})(\beta_{10}^{a_1}-\beta_{00}^{a_1}+\beta_{10}^{a_2}-\beta_{00}^{a_2}) \\ c &= (\beta_{00}^{a_2}-\beta_{00}^{a_1})(\beta_{10}^{a_1}-\beta_{00}^{a_1}+\beta_{10}^{a_3}-\beta_{00}^{a_3})- \\ &\quad (\beta_{00}^{a_3}-\beta_{00}^{a_1})(\beta_{10}^{a_1}-\beta_{00}^{a_1}+\beta_{10}^{a_2}-\beta_{00}^{a_2})\end{aligned}.$$

After solving for (19) for s, we can substitute the result into (18) to recover the corresponding δ. At the solution points, either all of a_1, a_2 and a_3 are optimal, or none of them are. Knowing the location of these triple-points is useful for determining if an action is dominated for all (s,δ), as we now demonstrate.

Lemma 1. *If action a is optimal at time T for some point (s,δ) but is not optimal for any (s,δ) on the boundary of the domain, then a is optimal for some (s,δ) that is a triple-point.*

Proof. Suppose a is optimal for some (s,δ) in the domain but is not optimal for any (s,δ) on the boundary of the domain. Further suppose that a is not optimal at any triple-point. Then the region where a is optimal must be completely enclosed by the region where a *single* other action a' is optimal. However, by Equation (18), the boundary between the regions where a is superior to a' and vice-versa has infinite extent in both s and δ, and therefore must intersect the boundary of the domain of (s,δ), since a is optimal for some (s,δ) by assumption. Thus we have a contradiction. □

From Lemma 1 we know that to find all actions that are optimal for some (s, δ) we need only check the boundaries and the triple points. The boundaries can be checked using Algorithm 2. (Note that because $V(s, \cdot)$ is bilinear in δ and in s, we can also use Algorithm 2 to identify for any fixed δ the actions that are optimal for some s.) We can then enumerate the $\binom{|A|}{3}$ triple-points and check them to detect any regions that do not intersect a the boundary of the domain, like that of action 1 in Figure 3 where we have identified the triple-points with white dots. This procedure reveals all actions that are optimal for some (s, δ), and thereby identifies any actions that are not optimal for any (s, δ). Checking the $\binom{10}{3}$ intersections in the example takes approximately 0.06 seconds using Matlab 2009b on a 3.0 GHz Xeon processor. While we have described the process for bilinear functions, we can immediately extend this algorithm for piecewise bilinear functions by applying it between pairs of knots.

5. Extensions to Higher Dimensions

Although the presented algorithms are sufficient for handling the quality and quantity of data often found in the analysis of randomized clinical trials, it is interesting to consider extending the approaches to higher dimensions. Extending Algorithm 3 to accommodate more than one state variable is trivial; each regression simply results in more $\hat{\beta}$ coefficients at each knot. Accommodating more reward functions will be more challenging. In our setting, $\hat{Q}_t(s, a, \cdot)$ is piecewise linear but not convex in δ. For d reward functions and tradeoffs $\boldsymbol{\delta} = \delta_1, ..., \delta_d$ on the $(d-1)$-simplex, we will represent the $(d-1)$-dimensional regions over which $\hat{Q}_t(s, a, \cdot)$ is linear, along with its value within each region. For $d = 3$ we anticipate using triangulation algorithms that require $O(n \log n)$ time for n knots.

We conjecture that an analogue of Lemma 1 holds in higher dimensions, and that identifying all non-dominated actions for more state variables and/or reward functions will require computing intersections of hyperboloids in higher dimensions. For p state variables and d reward functions, we would need to find all points where $p + d$ actions are simultaneously optimal. This will not have a closed-form solution, and will require numerical methods for zeros of polynomials.

6. Conclusion

Data analysis of multi-stage randomized clinical trials is challenging because the priorities of future users of the analysis are unknown. A policy learned for patients who desire very low symptom levels may not be useful for patients who prefer to have few side-effects. We have shown how uncertainty about these future priorities can be expressed as a set of possible MDPs that are defined by a convex set of reward functions. We discussed algorithms for simultaneously solving all MDPs in this set. We introduced a new algorithm that is asymptotically more time- and space-efficient than previous approaches in the discrete-state case, and showed how it can be extended to the continuous-state setting using linear function approximation. We also gave an algorithm for finding the set of all non-dominated actions in the continuous-state setting. This novel extension is appropriate for environments where we must estimate state-action value functions from data, as when analyzing randomized controlled trials.

This work was supported by National Institutes of Health grants R01 MH080015 and P50 DA10075.

References

Barrett, L. and Narayanan, S. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, 2008.

Bertsekas, D. P. and Tsitsiklis, J. N. *Neuro-Dynamic Programming*, chapter 2.1, Athena Scientific, 1996.

Ehrgott, Matthias. *Multicriteria Optimization*, chapter 3. Springer, second edition, 2005.

Murphy, S. A., Oslin, S. W., Rush, A. John, and Zhu, J. Methodological challenges in constructing effective treatment sequences for chronic psychiatric disorders. *Neuropsychopharmacology*, 32:257–262, 2007.

Pineau, J., Gordon, G., and Thrun, S. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1025–1032, 2003.

Pineau, J., Bellemare, M. G., Rush, A. J., Ghizaru, A., and Murphy, S. A. Constructing evidence-based treatment strategies using methods from computer science. *Drug and Alcohol Dependence*, 88(Suppl 2): S52–S60, May 2007.

Rush, A. J., Fava, M., Wisniewski, S. R., Lavori, P. W., Trivedi, M., Sackeim, H. A., and et al. Sequenced treatment alternatives to relieve depression (STAR*D): rationale and design. *Controlled Clinical Trials*, 25(1):119–42, Feb 2004.

Zhao, Y., Kosorok, M. R., and Zeng, D. Reinforcement learning design for cancer clinical trials. *Statistics in Medicine*, 28:3294–3315, 2009.