Point-Based Planning for Multi-Objective POMDPs

Diederik M. Roijers[1], **Shimon Whiteson**[1], and **Frans A. Oliehoek**[1,2]
[1]Informatics Institute, University of Amsterdam, The Netherlands
[2]Department of Computer Science, University of Liverpool, United Kingdom
{d.m.roijers, s.a.whiteson, f.a.oliehoek}@uva.nl

Abstract

Many sequential decision-making problems require an agent to reason about both multiple objectives and uncertainty regarding the environment's state. Such problems can be naturally modelled as *multi-objective partially observable Markov decision processes (MOPOMDPs)*. We propose *optimistic linear support with alpha reuse (OLSAR)*, which computes a bounded approximation of the optimal solution set for all possible weightings of the objectives. The main idea is to solve a series of scalarized single-objective POMDPs, each corresponding to a different weighting of the objectives. A key insight underlying OLSAR is that the policies and value functions produced when solving scalarized POMDPs in earlier iterations can be reused to more quickly solve scalarized POMDPs in later iterations. We show experimentally that OLSAR outperforms, both in terms of runtime and approximation quality, alternative methods and a variant of OLSAR that does not leverage reuse.

1 Introduction

Many real-world planning problems require reasoning about incomplete knowledge of the environment's state. These problems are often naturally modelled as *partially observable Markov decision processes (POMDPs)* [Kaelbling *et al.*, 1998]. Since optimal POMDP planning is intractable, much research focusses on efficient approximations.

However, planning is often further complicated by the presence of multiple objectives. For example, an agent might want to maximize the performance of a computer network while minimizing its power consumption [Tesauro *et al.*, 2007]. Such problems can be modeled as *multi-objective partially observable Markov decision processes (MOPOMDPs)* [Soh and Demiris, 2011a; 2011b].

Solving MOPOMDPs does not always require special solution methods. For example, when the vector-valued reward function can be *scalarized*, i.e., converted to a scalar function, before planning, the original problem may be solvable with existing single-objective methods. Unfortunately, a priori scalarization is not possible when the *scalarization weights*, i.e., the parameters of the scalarization, are not known in advance. For example, a company that produces different resources whose market prices vary may not have enough time to re-solve the decision problem for each price change. In such cases, multi-objective methods are needed to compute a set of solutions optimal for all scalarizations.

Little research has been conducted on such methods for MOPOMDPs. A naive approach is to translate the MOPOMDP to a single-objective POMDP with an augmented state space that includes as a feature a single, hidden "true" objective; a similar approach has been proposed for MOMDPs [White and Kim, 1980]. However, this approach precludes the use of POMDP methods that exploit an initial belief (such as point-based methods [Shani *et al.*, 2013]) since specifying an initial belief fixes the scalarization weights. Another naive approach is to uniformly randomly sample scalarization weights and then solve each resulting scalarized problem with a single-objective POMDP planner. Unfortunately, this requires a large amount of sampling to get good coverage of the space of scalarization weights.

In this paper, we propose a new MOPOMDP solution method called *optimistic linear support with alpha reuse (OLSAR)*. Our approach is based on *optimistic linear support (OLS)* [Roijers *et al.*, 2014a; 2015], a general framework for solving multi-objective decision problems that uses a priority queue to make smart choices about which scalarized problem instances to solve.

OLSAR contains two key improvements over OLS that are essential to making it tractable for MOPOMDPs. First, it uses a novel *OLSAR-compliant* version of the point-based solver *Perseus* [Spaan and Vlassis, 2005]. OLSAR-compliant Perseus solves a given scalarized POMDP and simultaneously computes the resulting policy's *multi-objective* value. Doing so avoids the need for the separate policy evaluation step typically employed by OLS [Roijers *et al.*, 2014b]. Since POMDP policy evaluation is very expensive, the use of OLSAR-compliant Perseus is key to OLSAR's efficiency. Second, rather than solving each scalarized POMDP from scratch, OLSAR reuses the α-matrices that represent each policy's multi-objective value to form an initial lower bound for subsequent calls to OLSAR-compliant Perseus. Such reuse leads to dramatic reductions in runtime in practice.

Furthermore, OLSAR avoids the problems of the naive approaches: it can make use of POMDP methods that require an initial belief but does not require inefficient sampling of the scalarization weight space.

We show that OLSAR is guaranteed to find a bounded approximate solution in a finite number of iterations. In addition, we show experimentally that it outperforms uniform random sampling of scalarization weights, both with and without α-matrix reuse, as well as OLS without α-matrix reuse.

2 Background

We start with background on multi-objective decision problems, POMDPs, MOPOMDPs, and OLS.

2.1 Multi-Objective Decision Problems

In single-objective decision problems, an agent must find a policy π that maximizes V, the expected value of, e.g., a sum of discounted rewards. In multi-objective decision problems, there are n objectives, yielding a vector-valued reward function. As a result, each policy has a vector-valued expected value $\mathbf{V}$ and, rather than having a single optimal policy, there can be multiple policies whose value vectors are optimal for different preferences over the objectives. Such preferences can be expressed using a scalarization function $f(\mathbf{V}, \mathbf{w})$ that is parameterized by a parameter vector $\mathbf{w}$ and returns $V_{\mathbf{w}}$, the *scalarized* value of $\mathbf{V}$. When $\mathbf{w}$ is known beforehand, it may be possible to *a priori* scalarize the decision problem and apply standard single-objective solvers. However, when $\mathbf{w}$ is unknown during planning, we need an algorithm that computes a set of policies containing at least one policy with maximal scalarized value for *each possible* $\mathbf{w}$.

Which policies should be included in this set depends on what we know about f, as well as which policies are allowed. In many real-world problems, f is linear, i.e.,

$$f(\mathbf{V}, \mathbf{w}) = \mathbf{w} \cdot \mathbf{V},$$

where $\mathbf{w}$ is a vector of non-negative weights that sum to 1. In this case, a sufficient solution is the *convex hull (CH)*, the set of all undominated policies under a linear scalarization:

$$CH(\Pi) = \{\pi : \pi \in \Pi \wedge \exists \mathbf{w} \forall (\pi' \in \Pi)\, \mathbf{w} \cdot \mathbf{V}^{\pi} \geq \mathbf{w} \cdot \mathbf{V}^{\pi'}\}, \quad (1)$$

where Π is the set of allowed policies. However, the entire CH may not be necessary. Instead, it also suffices to compute a *convex coverage set (CCS)*, a lossless subset of the CH. For each possible $\mathbf{w}$, a CCS contains at least one vector from the CH that has the maximal scalarized value for $\mathbf{w}$.

When f is not linear, we might require the *Pareto front (PF)*, a superset of the CH containing all policies for which no other policy has a value that is at least equal in all objectives and greater in at least one objective. However, when *stochastic policies* are allowed, all values on the PF can be constructed from mixtures of CCS policies [Vamplew *et al.*, 2009]. Therefore, the CCS is inadequate only if the scalarization function is nonlinear *and* stochastic policies are forbidden. For simplicity, we assume linear scalarizations in this paper. However, our methods are also applicable to nonlinear scalarizations as long as stochastic policies are allowed.

Using the CCS, we can define a *scalarized value function*:

$$V^*_{CCS}(\mathbf{w}) = \max_{\mathbf{V} \in CCS} \mathbf{w} \cdot \mathbf{V}, \quad (2)$$

which returns, for each $\mathbf{w}$, the maximal scalarized value achievable for that weight. Finding this function is equivalent to finding the CCS and solving the decision problem. $V^*_{CCS}(\mathbf{w})$ is a *piecewise-linear and convex (PWLC)* function over weight space, a property that can be exploited to construct a CCS efficiently [Roijers *et al.*, 2014a; 2015].

When the CCS cannot be computed exactly, an ε-CCS can be computed instead. A set X is an ε-CCS is when the maximum scalarized error across all weights is at most ε:

$$\forall \mathbf{w}, V^*_{CCS}(\mathbf{w}) - (\max_{\mathbf{V} \in X} \mathbf{w} \cdot \mathbf{V}) \leq \varepsilon. \quad (3)$$

2.2 POMDPs

An infinite-horizon single-objective POMDP [Kaelbling *et al.*, 1998; Madani *et al.*, 2003] is a sequential decision problem that incorporates uncertainty about the state of the environment, and is specified as a tuple $\langle S, A, R, T, \Omega, O, \gamma \rangle$ where S is the state space; A is the action space; R is the reward function; T is the transition function, giving the probability of a next state given an action and a current state; Ω is the set of observations; O is the observation function, giving the probability of each observation given an action and the resulting state; and γ is the discount factor.

Typically, an agent maintains a belief b over which state it is in. The value function for a single-objective POMDP, V_b, is defined in terms of this belief and can be represented by a set $\mathcal{A}$ of α-*vectors*. Each vector α (of length $|S|$) gives a value for each state s. The value of a belief b given $\mathcal{A}$ is:

$$V_b = \max_{\alpha \in \mathcal{A}} b \cdot \alpha. \quad (4)$$

Each α-vector is associated with an action. Therefore, a set of α-vectors $\mathcal{A}$ also provides a *policy* $\pi_{\mathcal{A}}$ that for each belief takes the maximizing action in (4).

While infinite-horizon POMDPs are in general undecidable [Madani *et al.*, 2003], an ε-approximate value function can in principle be computed using techniques like *value iteration (VI)* [Monahan, 1982] and *incremental pruning* [Cassandra *et al.*, 1997]. Unfortunately, these methods scale poorly in the number of states. However, *point-based* methods [Shani *et al.*, 2013], which perform approximate backups by computing the best α-vector only for a set B of sampled beliefs, scale much better. For each $b \in B$, a *point-based backup* is performed by first computing for each a and o, the *back-projection* $g_i^{a,o}$ of each next-stage value vector $\alpha_i \in \mathcal{A}_k$:

$$g_i^{a,o}(s) = \sum_{s' \in S} O(a, s', o) T(s, a, s') \alpha_i(s'). \quad (5)$$

This step is identical for (and can be shared amongst) all $b \in B$. For each b, the back-projected vectors $g_i^{a,o}$ are used to construct $|A|$ new α-vectors (one for each action):

$$\alpha_{k+1}^{b,a} = r^a + \gamma \sum_{o \in \Omega} \arg\max_{g^{a,o}} b \cdot g^{a,o}, \quad (6)$$

where r^a is a vector containing the immediate rewards for performing action a in each state. Finally, the $\alpha_{k+1}^{b,a}$ that maximizes the inner product with b (cf. (4)) is retained as the new α-vector for b:

$$\texttt{backup}(\mathcal{A}_k, b) = \arg\max_{\alpha_{k+1}^{b,a}} b \cdot \alpha_{k+1}^{b,a}. \quad (7)$$

Point-based methods typically perform several point-based backups using $\mathcal{A}_k$ for different b to construct the set of α-vectors for the next iteration, $\mathcal{A}_{k+1}$. By constructing the α-vectors only from the $g^{a,o}$ that are maximizing for the given b, point-based methods avoid generating the much larger set of α-vectors that an exhaustive backup of $\mathcal{A}_k$ would generate.

2.3 MOPOMDPs

A MOPOMDP [Soh and Demiris, 2011b] is a POMDP with a vector-valued reward function $\mathbf{R}$ instead of a scalar one. The value of a MOPOMDP policy given an initial belief b_0 is thus also a vector $\mathbf{V}_{b_0}$. The scalarized value given $\mathbf{w}$ is then $\mathbf{w} \cdot \mathbf{V}_{b_0}$. Note that in an MOPOMDP, a nonlinear scalarization would make it impossible to construct a POMDP model for a particular $\mathbf{w}$, since a nonlinear f does not distribute over the expected sum of rewards. Therefore, nonlinear scalarization would preclude the application of dynamic-programming-based POMDP methods [Roijers *et al.*, 2013].

Because $\mathbf{R}$ is vector valued, each element of each α-vector, i.e., each $\alpha(s)$, is itself a vector, indicating the value in all objectives. Thus, each α-vector is actually an α-matrix $\mathbf{A}$ in which each row $\mathbf{A}(s)$ represents the multi-objective value vector for s. The multi-objective value of taking the action associated with $\mathbf{A}$ under belief b (provided as a row vector) is then $b\mathbf{A}$. When $\mathbf{w}$ is also given (as a column vector), the scalarized value of taking the action associated with $\mathbf{A}$ under belief b is $b\mathbf{A}\mathbf{w}$. Given a set of α-matrices $\mathcal{A}$ that approximates the multi-objective value function, we can thus extract the scalar value given a belief b for every $\mathbf{w}$:

$$V_b(\mathbf{w}) = \max_{\mathbf{A} \in \mathcal{A}} \; b\mathbf{A}\mathbf{w}. \tag{8}$$

Since each α-matrix is associated with a certain action, a policy $\pi^{\mathbf{w}}_{\mathcal{A}}$ can be distilled from $\mathcal{A}$ given $\mathbf{w}$. The multi-objective value for a given b_0 under policy π is denoted $\mathbf{V}^{\pi}_{b_0} = b_0\mathbf{A}$.

A naive approach to solving MOMDPs, which we refer to as *random sampling (RS)* [Roijers *et al.*, 2014b], was introduced in the context of multi-objective MDPs. RS samples many $\mathbf{w}$'s and creates a scalarized POMDP according to each $\mathbf{w}$. Unless some prior knowledge about $\mathbf{w}$ is available, sampling is uniformly random. Each scalarized POMDP can then be solved with any POMDP method, including point-based ones. The resulting multi-objective value vectors are maintained in a set X, which forms a lower bound on the scalarized value function:

$$V^*_X(\mathbf{w}) = \max_{\mathbf{V} \in X} \mathbf{w} \cdot \mathbf{V}.$$

An important downside of this method is that although X will converge to the CCS in the limit, we do not know when this is the case. Furthermore, because $\mathbf{w}$ are sampled at random, RS might do a lot of unnecessary work.

2.4 Optimistic Linear Support

This paper builds on *optimistic linear support (OLS)* [Roijers *et al.*, 2014a; 2015] — a general scheme for solving multi-objective decision problems. Like RS, OLS repeatedly calls a single-objective solver to solve scalarized instances of the multi-objective problem and maintains a set X that forms a lower bound on the value function $V^*_X(\mathbf{w})$. Contrary to RS

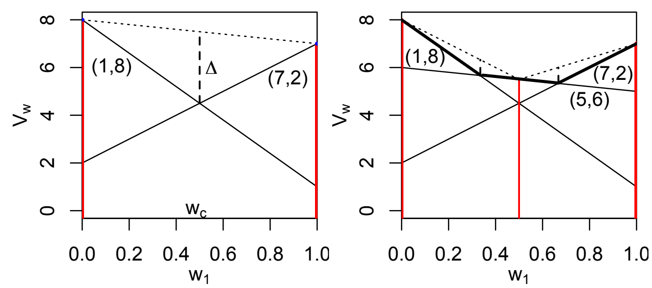


Figure 1: Finding value vectors, corner weights, and the maximal possible improvement for a partial CCS X in OLS.

however, OLS selects the weight vectors $\mathbf{w}$ instances intelligently. In order to select good $\mathbf{w}$'s for scalarization, OLS exploits the observation that $V^*_X(\mathbf{w})$ is PWLC over the weight simplex. In particular, OLS selects only so-called *corner weights* that lie at the intersections of line segments of the PWLC function $V^*_X(\mathbf{w})$ that correspond to the value vectors found so far. OLS prioritizes these corner weights according to an optimistic estimate of their potential error reduction.

First, let us assume OLS has access to an optimal single-objective solver. For this case, the workings of OLS are illustrated for a 2-objective problem in Figure 1. The first corner weights (denoted with red vertical line segments) are the extrema of the weight simplex. On the left, OLS finds an optimal value vector for both extrema: $(1, 8)$ and $(7, 2)$. After finding these first two value vectors, OLS identifies a new corner weight: $\mathbf{w}_c$. At this corner weight, the maximal possible improvement consistent with scalarized values found at the extrema of the weight simplex is Δ. OLS now calls the single-objective solver for $\mathbf{w}_c$, and discovers a new value vector, $(5, 6)$, leading to two new corner weights. OLS first processes the corner weight with the largest Δ, which is an optimistic estimate of the error reduction that processing that corner weight will yield. In Figure 1 (right), OLS selects the corner weight on the right, because its maximal possible improvement, Δ, is highest.

If no improving value vector for a corner weight is found by the single-objective solver, the value for that corner weight is optimal, and no new corner weights are generated. When none of the remaining corner weights yield an improvement, the intermediate set X is a CCS and OLS terminates.

When the single-objective solver is exact, OLS is thus guaranteed to produce an exact CCS after a finite number of calls to the single-objective solver. Furthermore, even when a bounded approximate single-objective solver is used instead, OLS retains this quality bound.

Lemma 1. *Given an ε-optimal approximate single-objective solver, OLS produces an ε-CCS [Roijers* et al.*, 2014b].*

3 OLS with Alpha Reuse

We now present our main contribution, a new algorithm for MOPOMDPs called *optimistic linear support with alpha reuse (OLSAR)*. OLSAR builds on OLS but significantly improves it in two ways. First, by reusing the results of earlier

Algorithm 1: $\texttt{OLSAR}(b_0, \eta)$

1 $X \leftarrow \emptyset$; // partial CCS of multi-objective value vectors $\mathbf{V}_{b_0}$
2 $WV_{old} \leftarrow \emptyset$; // searched weights and scalarized values
3 $Q \leftarrow$ priority queue with weights to search;
4 Add extrema of the weight simplex to Q with infinite priority;
5 $\mathcal{A}_{all} \leftarrow$ a set of α-matrices forming a lower bound on the value;
6 $B \leftarrow$ set of sampled belief points (e.g., by random exploration);
7 **while** $\neg Q.\texttt{isEmpty}() \wedge \neg$*timeOut* **do**
8 $\quad \mathbf{w} \leftarrow Q.\texttt{dequeue}()$; // Retrieve a weight vector
9 $\quad \mathcal{A}_r \leftarrow$ select the best $\mathbf{A}$ from $\mathcal{A}_{all}$ for each $b \in B$, given $\mathbf{w}$;
10 $\quad \mathcal{A}_{\mathbf{w}} \leftarrow \texttt{solveScalarizedPOMDP}(\mathcal{A}_r, B, \mathbf{w}, \eta)$;
11 $\quad \mathbf{V}_{b_0} \leftarrow \max_{\mathbf{A} \in \mathcal{A}_{\mathbf{w}}} b_0 \mathbf{A} \mathbf{w}$;
12 $\quad \mathcal{A}_{all} \leftarrow \mathcal{A}_{all} \cup \mathcal{A}_{\mathbf{w}}$;
13 $\quad WV_{old} = WV_{old} \cup \{(\mathbf{w},\ \mathbf{w} \cdot \mathbf{V}_{b_0})\}$;
14 $\quad$ **if** $\mathbf{V}_{b_0} \not\in X$ **then**
15 $\quad\quad X \leftarrow X \cup \{\mathbf{V}_{b_0}\}$;
16 $\quad\quad W \leftarrow$ compute new corner weights and maximum possible improvements $(\mathbf{w}, \Delta_{\mathbf{w}})$ using WV_{old} and X;
17 $\quad\quad Q.\texttt{addAll}(W)$;
18 $\quad$ **end**
19 **end**
20 **return** X;

iterations, OLSAR greatly speeds up later iterations. Second, by using an *OLSAR-compliant* point-based solver as a subroutine, it obviates the need for separate policy evaluations.

We first describe OLSAR and specify the criteria that the single-objective POMDP method that OLSAR calls as a subroutine, which we call `solveScalarizedPOMDP`, must satisfy to be OLSAR-compliant. Then, Section 3.1 describes an instantiation of `solveScalarizedPOMDP` that we call *OLSAR-Compliant Perseus*. Finally, Section 3.2 discusses OLSAR's theoretical properties, which leverage existing theoretical guarantees of point-based methods and OLS.

OLSAR maintains an approximate CCS, X, and thereby an approximate scalarized value function V^*_X. By extending X with new value vectors $\mathbf{V}_{b_0}$, V^*_X approaches V^*_{CCS}. The vectors $\mathbf{V}_{b_0}$ are computed using sets of α-matrices. OLSAR finds these sets of α-matrices by solving a series of scalarized problems, each of which is a decision problem over belief space for a different weight $\mathbf{w}$. Each scalarized problem is solved by a single-objective solver we call `solveScalarizedPOMDP`, which computes the value function of the MOPOMDP scalarized by $\mathbf{w}$.

OLSAR, given in Algorithm 1, takes an initial belief b_0 and a convergence threshold η as input. The value vectors $\mathbf{V}_{b_0}$ found for different $\mathbf{w}$ are kept in a partial approximate CCS, X (line 1). OLSAR keeps track of which $\mathbf{w}$ have been searched and the associated scalarized values in a set of tuples WV_{old} (line 2). The weights that still need to be investigated are kept in a priority queue Q, and prioritized by their maximal possible improvement $\Delta_{\mathbf{w}}$ (lines 3–4), which is calculated via a linear program [Roijers *et al.*, 2014a; 2015]. OLSAR also maintains $\mathcal{A}_{all}$, a set of all α-matrices returned by calls to `solveScalarizedPOMDP` to reuse (line 5) and B, a set of sampled beliefs (line 6).

In the main loop, (lines 7–18) OLSAR repeatly pops a corner weight off the queue. For each popped $\mathbf{w}$, it selects the α-matrices from $\mathcal{A}_{all}$ to initialize $\mathcal{A}_r$, a set of α-matrices that form a lower bound on the (multi-objective) value (line 9). Initially, this lower bound is a heuristic lower bound. In the single-objective case, this usually consists of a minimally realisable value heuristic, in the form of α-vectors. In order to enable α-reuse for the multi-objective version, these heuristics must be in the form of α-matrices. For example, if we denote the minimal reward for each objective i as R^i_{min}, one lower bound heuristic α-matrix $\mathbf{A}_{min}$ is the vectors consisting of $R^i_{min}/(1-\gamma)$ for each objective and state.

OLSAR selects the maximizing α-matrix (2) for each belief $b \in B$ and the given $\mathbf{w}$ and puts them in a set $\mathcal{A}_r$. Using $\mathcal{A}_r$, OLSAR calls `solveScalarizedPOMDP`. After obtaining a new set of α-matrices $\mathcal{A}_{\mathbf{w}}$ from `solveScalarizedPOMDP`, OLSAR calculates $\mathbf{V}_{b_0}$; the maximizing multi-objective value vector for b_0 at $\mathbf{w}$ (line 11). If $\mathbf{V}_{b_0}$ is an improvement to X (line 14), OLSAR adds it to X and calculates the new corner weights and their priorities.

A key insight behind OLSAR is that if we can retrieve the α-matrices underlying the policy found by `solveScalarizedPOMDP` for a specific $\mathbf{w}$, we can reuse these α-matrices as a lower bound for subsequent calls to `solveScalarizedPOMDP` with another weight $\mathbf{w}'$. Especially when $\mathbf{w}$ and $\mathbf{w}'$ are similar, we expect this lower bound to be close to the α-matrices required for $\mathbf{w}'$.

However, to exploit this insight, `solveScalarizedPOMDP` must return the α-matrices explicitly, not just the scalarized value or the single-objective α-vectors, as standard single-objective solvers do. A naive way to retrieve the α-matrices is to perform a separate policy evaluation on the policy returned by `solveScalarizedPOMDP`. However, since POMDP policy evaluation is expensive, we instead require `solveScalarizedPOMDP` to be *OLSAR-compliant*: i.e., to return a set of α-matrices, while computing the same scalarized value function as a single-objective solver. We propose an OLSAR-compliant solver in Section 3.1.

The α-matrix reuse enabled by an OLSAR-compliant subroutine is key to solving MOPOMDPs efficiently. Intuitively, the corner weights that OLSAR selects lie increasingly closer together as the algorithm iterates. Consequently, the policies and value functions computed for those weights lie closer together as well and `solveScalarizedPOMDP` needs less and less time to improve upon the α-matrices that begin increasingly close to their converged values. In fact, late in the execution of OLSAR, corner weights are tested that yield no value improvement, i.e., no new α-matrices are found. These tests serve only to confirm that the CCS has been found, rather than to improve it. While such confirmation tests would be expensive in OLS, in OLSAR they are trivial: the already present α-matrices suffice and `solveScalarizedPOMDP` converges immediately. As we show in Section 4, this greatly reduces OLSAR's runtime.

3.1 OLSAR-Compliant Perseus

OLSAR requires an OLSAR-compliant implementation of `solveScalarizedPOMDP` that returns the multi-objective value of the policy found for a given $\mathbf{w}$. This requires redefining the point-based backup such that it returns an α-matrix rather than an α-vector. Starting from a set of α-matrices $\mathcal{A}_k$, we now perform a backup for a given b *and* $\mathbf{w}$. As in (9), the

Algorithm 2: `OCPerseus`$(\mathcal{A}, B, \mathbf{w}, \eta)$

1 $\mathcal{A}' \leftarrow \mathcal{A}$;
2 $\mathcal{A} \leftarrow \{-\vec{\infty}\}$; // worst possible vector in a singleton set
3 **while** $\max_b \max_{\mathbf{A}' \in \mathcal{A}'} b\mathbf{A}'\mathbf{w} - (\max_{\mathbf{A} \in \mathcal{A}} b\mathbf{A}\mathbf{w}) > \eta$ **do**
4 $\quad \mathcal{A} \leftarrow \mathcal{A}'$; $\mathcal{A}' \leftarrow \emptyset$; $B' \leftarrow B$;
5 $\quad$ **while** $B' \neq \emptyset$ **do**
6 $\quad\quad$ Randomly select b from B';
7 $\quad\quad$ $\mathbf{A} \leftarrow$ `backupMO`$(\mathcal{A}, b, \mathbf{w})$;
8 $\quad\quad$ $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{ \underset{\mathbf{A}' \in (\mathcal{A} \cup \{\mathbf{A}\})}{\arg\max} \; b\mathbf{A}'\mathbf{w}\}$;
9 $\quad\quad$ $B' \leftarrow \{b \in B' : \max_{\mathbf{A}' \in \mathcal{A}'} b\mathbf{A}'\mathbf{w} < \max_{\mathbf{A} \in \mathcal{A}} b\mathbf{A}\mathbf{w}\}$;
10 $\quad$ **end**
11 **end**
12 **return** $\mathcal{A}'$;

new backup first computes the back-projections $\mathbf{G}_i^{a,o}$ (for all a, o) of each next-stage α-matrix $\mathbf{A}_i \in \mathcal{A}_k$. However, these $\mathbf{G}_i^{a,o}$ are now matrices instead of vectors:

$$\mathbf{G}_i^{a,o}(s) = \sum_{s' \in S} O(a, s', o) T(s, a, s') \mathbf{A}_i(s'). \tag{9}$$

As before, the back-projected matrices $\mathbf{G}_i^{a,o}$ are identical for (and can be shared amongst) all $b \in B$. For the given b and $\mathbf{w}$, the back-projected matrices are used to construct $|A|$ new α-matrices (one for each action):

$$\mathbf{A}_{k+1}^{b,a} = r^a + \gamma \sum_{o \in \Omega} \underset{G^{a,o}}{\arg\max}\, b\, \mathbf{G}^{a,o}\mathbf{w}. \tag{10}$$

Note that the vectors $\mathbf{G}^{a,o}\mathbf{w}$ can also be shared between all $b \in B$. Therefore, we cache the values of $\mathbf{G}^{a,o}\mathbf{w}$.

Finally, the $\mathbf{A}_{k+1}^{b,a}$ that maximizes the *scalarized value* given b and $\mathbf{w}$ is selected by the `backupMO` operator:

$$\texttt{backupMO}(\mathcal{A}_k, b, \mathbf{w}) = \underset{\mathbf{A}_{k+1}^{b,a}}{\arg\max}\, b\mathbf{A}_{k+1}^{a,b}\mathbf{w}. \tag{11}$$

We can plug `backupMO` into any point-based method. In this paper, we use Perseus [Spaan and Vlassis, 2005] because it is fast and can handle large sampled belief sets. The resulting OLSAR-compliant Persues is given in Algorithm 2. It takes as input an initial lower bound $\mathcal{A}$ on the value function in the form of a set of α-matrices, a set of sampled beliefs, a scalarization weight $\mathbf{w}$, and a precision parameter η. It repeatedly improves this lower bound (lines 3-10) by finding an improving α-matrix for each sampled belief. To do so, it selects a random belief from the set of sampled beliefs (line 6) and, if possible, finds an improving $\mathbf{A}$ for it (line 7). When such an α-matrix also improves the value for another belief point in B, this belief point is ignored until the next iteration (line 9). This results in an algorithm that generates few α-matrices in early iterations, but improves the lower bound on the value function, and gets more precise, i.e., generates more α-matrices, in later iterations.

3.2 Analysis

Point-based methods like Perseus have guarantees on the quality of the approximation. These guarantees depend on the density δ_B of the sampled belief set, i.e., the maximal distance from the closest $b \in B$ to any other belief in the belief set [Pineau *et al.*, 2006].

Lemma 2. *The error ε on the lower bound of the value of an infinite-horizon POMDP after convergence of point-based methods is:*

$$\varepsilon \leq \frac{\delta_B(R_{max} - R_{min})}{(1-\gamma)^2}, \tag{12}$$

where R_{max} and R_{min} are the maximal and minimal possible immediate rewards [Pineau et al.*, 2006].*

Using Lemma 2 we can bound the error of the approximate CCS computed by OLSAR.

Theorem 1. *OLSAR implemented with* `OCPerseus` *using belief set B converges in a finite number of iterations to an ε-CCS, where $\varepsilon \leq \frac{\delta_B(R_{max}-R_{min})}{(1-\gamma)^2}$.*

Proof. Because OLSAR follows the same iterative process as OLS, Lemma 1 applies to it, i.e., it converges after finite calls to `solveScalarizedPOMDP`. Because `OCPerseus` differs from regular Perseus only in that it returns the multi-objective value rather than just the scalarized value, Lemma 2 also holds for `OCPerseus` for any $\mathbf{w}$. In other words, `OCPerseus` is an ε-approximate single-objective solver, with ε as specified by Lemma 2. Therefore, it follows that OLSAR produces an ε-CCS. □

A nice property of OLSAR is thus that it inherits any quality guarantees of the `solveScalarizedPOMDP` implementation it uses. Better initialization of `OCPerseus` due to α-matrix reuse does not effect the guarantees in any way. On the contrary, it affects only empirical runtimes. The next section presents experimental results that measure these runtimes.

4 Experiments

In this section, we empirically compare OLSAR to three baseline algorithms. The first is random sampling (RS), described in Section 2.3, which does not use OLS or alpha reuse. The second is *random sampling with alpha reuse (RAR)*, which does not use OLS. The third is OLS+OCPerseus, which does not use alpha reuse. We tested the algorithms on three MOPOMDPs based on the Tiger [Cassandra *et al.*, 1994] and Maze20 [Hauskrecht, 2000] benchmark POMDPs. Because we use infinite-horizon MOPOMDPs – which are undecidable – we cannot obtain the true CCS. Therefore, we compare our algorithms' solutions to *reference sets* obtained by running OLSAR with many more sampled belief points and η set 10 times smaller.

4.1 Multi-Objective Tiger Problem

In the Tiger problem [Cassandra *et al.*, 1994], an agent faces two doors: behind one is a tiger and behind the other is treasure. The agent can *listen* or *open* one of the doors. When it listens, it hears the tiger behind one of the doors. This observation is accurate with probability 0.85. Finding the treasure, finding the tiger, and listening yield rewards of 10, -100, and -1, respectively. We use a discount factor of $\gamma = 0.9$.

While the single-objective Tiger problem assumes all three rewards are on the same scale, it is actually difficult in practice to quantify the trade-offs between, e.g., risking an encounter with the tiger and acquiring treasure. In *two-objective MO-Tiger (Tiger2)*, we assume that the treasure and the cost of listening are on the same scale but treat avoiding the tiger as a separate objective. In *three-objective MO-Tiger (Tiger3)*, we treat all three forms of reward as separate objectives.

We conducted 25 runs of each method on both Tiger2 and Tiger3. We ran all algorithms with 100 belief points generated by random exploration, $\eta = 1 \times 10^{-6}$, and b_0 set to a uniform distribution. The reference set was obtained using 250 belief points.

On Tiger2 (Figure 2(a)), OLS+OCPerseus converges in $0.87s$ on average, with a maximal error in scalarized value across the weight simplex w.r.t. the reference set of 5×10^{-6}. OLSAR converges significantly faster (t-test: $p < 0.01$), in on average $0.53s$, with similar error 4×10^{-6} (this difference in error is not significant). RAR does not converge as it just keeps sampling new weights. However, we can measure error w.r.t. the reference set. After $2s$ (about four times what OLSAR needs to converge), RAR has an error with respect to the reference set of 4.5×10^{-3} (a factor of 10^3 higher than OLSAR after convergence). RS is even worse and does not come further than a 0.78 error after $2s$. OLS+OCPerseus and OLSAR perform similarly in the first half: they use about the same time on the first four iterations ($0.35s$). However, OLSAR is significantly faster for the second half. Thus, on Tiger2, OLS-based methods are significantly better than random sampling and alpha reuse significantly speeds the discovery of good value vectors.

The results for the Tiger3 problem are in Figure 2(b). RS and RAR both perform poorly, with RAR better by a factor of two. OLS+OCPerseus is faster than RAR but OLSAR is the fastest and converges in about $2s$. OLS+OCPerseus eventually converges to the same error but takes 10 times longer. As before, OLS-based methods outperform random sampling. Furthermore, for this 3-objective problem, alpha reuse speeds convergence of OLSAR by an order of magnitude.

4.2 Multi-Objective Maze20 Problem

In Maze20 [Hauskrecht, 2000], a robot must navigate through a 20-state maze. It has four actions to move in the cardinal directions, plus two sensing actions to perceive walls in the north-south or east-west directions. Transitions and observations are stochastic. In the single-objective problem, the agent gets reward of 2 for sensing, 4 for moving while avoiding the wall, and 150 for reaching the goal. In *multi-objective Maze 20 (MO-Maze20)*, the reward for reaching the goal is treated as a separate objective, yielding a two-objective problem.

To test performance of OLSAR on MO-Maze20, we first created a reference set by letting OLSAR converge using 1500 sampled beliefs. This took 11 hours. Then we ran OLSAR, OLS+OCPerseus, RAR, and RS with 1000 sampled beliefs. Due to the computational expense, we conducted only three runs per method. Figure 2(c) shows the results. OLSAR converges after (on average) 291 minutes ($4.90hrs$) at an error of 0.09, less than 0.5% w.r.t. the reference set. We let OLS+COPerseus run for 11 hours but it did not converge in that time. OLS+OCPerseus, RS, and RAR have similar performance after around 300 minutes. However, until then, OLS+OCPerseus does better (and unlike RS and RAR, OLS+OCPerseus is guaranteed to converge). OLSAR converges at significantly less error than what the other methods have attained after 400 minutes. We therefore conclude that OLSAR reduces error more quickly than the other algorithms and converges faster than OLS+OCPerseus.

5 Related Work

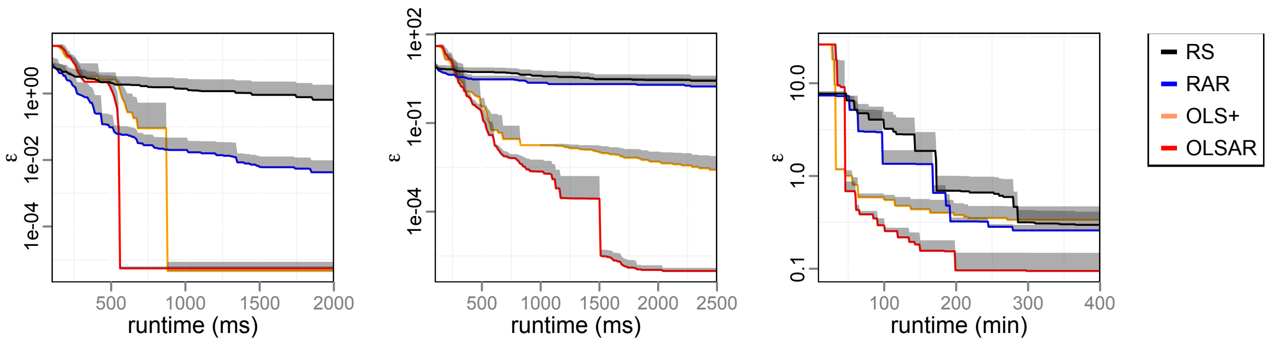
As previously mentioned, little research has been done on the subject of MOPOMDPs. White and Kim (1980) proposed a method to translate a multi-objective MDP into a POMDP, which can also be applied to a MOPOMDP and yields a single-objective POMDP. Intuitively, this translation assumes there is only one "true" objective. Since the agent does not know which objective is the true one, this yields a single-objective POMDP whose state is a tuple $\langle s, d \rangle$ where s is the original MOPOMDP state and $d \in \{1 \dots n\}$ indicates the true objective. The POMDP's state space is of size $|S|n$. The observations provide information about s but not about d. The resulting POMDP can be solved with standard methods but only those that do not require an initial belief, as such a belief would fix not only a distribution over s but also over d, yielding a policy optimal only for a given $\mathbf{w}$. Since this precludes efficient solvers like point-based methods, it is a severe limitation. We tested this method, and it proved prohibitively computationally expensive for even the smallest problems.

Other research on MOPOMDPs [Soh and Demiris, 2011a; 2011b] uses evolutionary methods to approximate the PF. However, no guarantees can be given on the quality of the approximation. Furthermore, as discussed in Section 2.1, finding a PF is only necessary when the scalarization function is nonlinear and stochastic policies are forbidden. In this paper, we focus on finding bounded approximations of the CCS.

6 Conclusions & Future Work

In this paper we proposed, analyzed, and tested OLSAR, a novel algorithm for MOPOMDPs that intelligently selects a sequence of scalarized POMDPs to solve. OLSAR uses OCPerseus, a scalarized MOPOMDP solver that returns the multi-objective value of the policies it finds, as well as the α-matrices that describe them. A key insight underlying OLSAR is that these α-matrices can be reused in subsequent calls to OCPerseus, greatly reducing runtimes in practice. Furthermore, since OCPerseus returns ε-optimal polcies, OLSAR is guaranteed in turn to return an ε-CCS. Finally, our experiments results show that OLSAR greatly outperforms alternatives that do not use OLS and/or α-matrix reuse.

In future work, we aim to extend OLSAR to make use of upper bounds as well as lower bounds, in the style of POMDP methods such as GapMin [Poupart *et al.*, 2011] or SARSOP [Kurniawati *et al.*, 2008]. Furthermore, it has been shown that the solution to a *constrained POMDP (CPOMDP)* — in which one primary objective must be maximized while bounding the value of one or more additional objectives —



(a) (b) (c)

Figure 2: The error with respect to a reference set as a function of the runtime for (a) Tiger 2, (b) Tiger3, and (c) MO-Maze20. The shaded regions represent standard error. Note the log scale in the y-axis. In order to avoid clutter in the plot (due to the log-scale) we only show the standard error above the lines.

corresponds to the solution of an MOPOMDP linearly scalarized according to some $\mathbf{w}$ [Poupart *et al.*, 2015]. Using this result, we aim to investigate how an ε-CCS computed by OLSAR could be used to explore what sets of constraints are feasible in a CPOMDP.

Acknowledgments

This research is supported by NWO DTC-NCAP (#612.001.109) project and the NWO Innovational Research Incentives Scheme Veni (#639.021.336).

References

[Cassandra *et al.*, 1994] A.R. Cassandra, L.P. Kaelbling, and M.L. Littman. Acting optimally in partially observable stochastic domains. In *AAAI*, volume 94, pages 1023–1028, 1994.

[Cassandra *et al.*, 1997] A.R. Cassandra, M.L. Littman, and N.L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *UAI*, pages 54–61, 1997.

[Hauskrecht, 2000] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *JAIR*, 13:33–94, 2000.

[Kaelbling *et al.*, 1998] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[Kurniawati *et al.*, 2008] H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, 2008.

[Madani *et al.*, 2003] O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *AIJ*, 147(1):5–34, 2003.

[Monahan, 1982] G.E. Monahan. State of the art — a survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[Pineau *et al.*, 2006] J. Pineau, G.J. Gordon, and S. Thrun. Anytime point-based approximations for large POMDPs. *JAIR*, 27:335–380, 2006.

[Poupart *et al.*, 2011] P. Poupart, K.E. Kim, and D. Kim. Closing the gap: Improved bounds on optimal POMDP solutions. In *ICAPS*, 2011.

[Poupart *et al.*, 2015] P. Poupart, A. Malhotra, P. Pei, K.E. Kim, B. Goh, and M. Bowling. Approximate linear programming for constrained partially observable Markov decision processes. In *AAAI*, 2015.

[Roijers *et al.*, 2013] D.M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *JAIR*, 47:67–113, 2013.

[Roijers *et al.*, 2014a] Diederik M. Roijers, Shimon Whiteson, and Frans A. Oliehoek. Linear support for multi-objective coordination graphs. In *AAMAS*, pages 1297–1304, May 2014.

[Roijers *et al.*, 2014b] D.M. Roijers, J. Scharpff, M.T.J. Spaan, F.A. Oliehoek, M.M. de Weerdt, and S. Whiteson. Bounded approximations for linear multi-objective planning under uncertainty. In *ICAPS*, pages 262–270, 2014.

[Roijers *et al.*, 2015] Diederik M Roijers, Shimon Whiteson, and Frans A Oliehoek. Computing convex coverage sets for faster multi-objective coordination. *JAIR*, 52:399–443, 2015.

[Shani *et al.*, 2013] G. Shani, J. Pineau, and R. Kaplow. A survey of point-based POMDP solvers. *JAAMAS*, 27(1):1–51, 2013.

[Soh and Demiris, 2011a] H. Soh and Y. Demiris. Evolving policies for multi-reward partially observable Markov decision processes (MR-POMDPs). In *GECCO*, pages 713–720, 2011.

[Soh and Demiris, 2011b] H. Soh and Y. Demiris. Multi-reward policies for medical applications: Anthrax attacks and smart wheelchairs. In *GECCO*, pages 471–478, 2011.

[Spaan and Vlassis, 2005] M.T.J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *JAIR*, pages 195–220, 2005.

[Tesauro *et al.*, 2007] G. Tesauro, R. Das, H. Chan, J. O. Kephart, C. Lefurgy, D. W. Levine, and F. Rawson. Managing power consumption and performance of computing systems using reinforcement learning. In *NIPS*, 2007.

[Vamplew *et al.*, 2009] P. Vamplew, R. Dazeley, E. Barker, and A. Kelarev. Constructing stochastic mixture policies for episodic multiobjective reinforcement learning tasks. In *Advances in Artificial Intelligence*, pages 340–349. 2009.

[White and Kim, 1980] C. C. White and K. M. Kim. Solution procedures for solving vector criterion Markov decision processes. *Large Scale Systems*, 1:129–140, 1980.