Bounded Approximations for Linear Multi-Objective Planning under Uncertainty

Diederik M. Roijers[1] **Joris Scharpff**[2] **Matthijs T. J. Spaan**[2] **Frans A. Oliehoek**[1]
Mathijs de Weerdt[2] **Shimon Whiteson**[1]
{d.m.roijers, f.a.oliehoek, s.a.whiteson}@uva.nl
[1]University of Amsterdam, The Netherlands
{j.c.d.scharpff, m.t.j.spaan, m.m.deweerdt}@tudelft.nl
[2]Delft University of Technology, The Netherlands

Abstract

Planning under uncertainty poses a complex problem in which multiple objectives often need to be balanced. When dealing with multiple objectives, it is often assumed that the relative importance of the objectives is known a priori. However, in practice human decision makers often find it hard to specify such preferences, and would prefer a decision support system that presents a range of possible alternatives. We propose two algorithms for computing these alternatives for the case of linearly weighted objectives. First, we propose an anytime method, *approximate optimistic linear support* (AOLS), that incrementally builds up a complete set of ϵ-optimal plans, exploiting the piecewise-linear and convex shape of the value function. Second, we propose an approximate anytime method, *scalarised sample incremental improvement* (SSII), that employs weight sampling to focus on the most interesting regions in weight space, as suggested by a prior over preferences. We show empirically that our methods are able to produce (near-)optimal alternative sets orders of magnitude faster than existing techniques.

1 Introduction

Many real-world problems involve parallel scheduling of activities of uncertain duration, while considering multiple objectives (Alarcon-Rodriguez et al. 2009; Mouaddib 2004; Calisi et al. 2007). For example, a factory that produces on demand wants to minimise orders delays while keeping production costs in check. This is difficult because, although planning a tight schedule might improve the efficiency and hence decrease costs, it leaves little room for dealing with setbacks and thereby increases expected delay.

Planning problems that deal with uncertain actions and multiple objectives can be naturally expressed using the *multi-objective Markov decision process* (MOMDP) framework (Roijers et al. 2013). The presence of multiple objectives does not necessarily require specialised solution methods; when the problem can be *scalarised*, i.e., translated to a single objective problem using a predefined *scalarisation function*, single objective methods can be applied. However, defining the scalarisation function requires specifying weights for each objective that are often hard to quantify without prior knowledge of the trade-offs inherent to the problem. When such knowledge is not available, solving an MOMDP requires computing the set of solutions optimal for all possible weights.

When policies are deterministic and the scalarisation function is monotonically increasing in all objectives, then this solution set is the Pareto front. However, this set can be large and difficult to compute.[1] Fortunately, it is often not necessary to compute the Pareto front. If the scalarisation function is linear, the *convex coverage set* (CCS) suffices. In addition, when policies can be stochastic, optimal policies for every monotonically increasing scalarisation can be constructed by mixing policies from the CCS (Vamplew et al. 2011). Many examples of MOMDPs with either linear scalarisation or stochastic policies can be found in the MOMDP survey by Roijers et al. (2013).

When we apply linear scalarisation to MOMDPs, we can define an optimal value function as a function of the scalarisation weights. Due to the linearity, this is a piecewise linear and convex (PWLC) function. Existing methods, such as *optimistic linear support* (OLS) (Roijers, Whiteson, and Oliehoek 2014), exploit this shape in order to guarantee an optimal solution for each weight while solving only a finite number of scalarised problems. However, in order to do this, OLS requires an exact single objective method as a subroutine. In the stochastic planning domain this corresponds to solving *Markov decision processes* (MDPs), which is prohibitively expensive when the state space is large and renders this approach infeasible for many realistic planning problems.

Typically, we would therefore consider using approximate single-objective algorithms for large MOMDPs. However, without an exact algorithm to evaluate single-objective solutions, OLS can no longer bound the number of weights it has to consider. To address this, we present *approximate OLS* (AOLS), which converges to an approximation of the CCS when an approximate MDP solver is used as a subroutine. Moreover, we prove that when the MDP solver is ϵ-optimal, i.e., produces a solution that has a value of at least $1 - \epsilon$ times the optimum, AOLS guarantees an ϵ-optimal solution for every weight. Like OLS, AOLS is an *anytime* algorithm,

[1]This is because for arbitrary monotonically increasing scalarisation functions it does not suffice to consider only stationary policies (Roijers et al. 2013).

i.e., at every iteration AOLS produces an intermediate result, for which maximum the error in scalarised value goes down with every iteration. Unlike OLS however, AOLS does not converge to the optimal solution, but to an ϵ-optimal solution, due to the usage of ϵ-approximate single-objective algorithms as subroutines.

In addition to AOLS, we present scalarised sampling incremental improvement (SSII), a different anytime approximation algorithm that is expected to perform better when some information about the weights is available in the form of a prior distribution over weights, e.g. "objective 1 is definitely more important than objective 2". The SSII algorithm draws samples from this prior and distributes its available runtime over these samples, thereby iteratively improving these samples. By focusing on these samples, it is likely to find a better solution *in the region that corresponds to the prior*.

We experimentally compare the algorithms we propose against the optimal OLS algorithm on moderately sized instances of the *maintenance planning problem* (Scharpff et al. 2013). For all algorithms we analyse the runtime and approximation quality versus the optimal solution set. We find that both AOSL and SSII are able to produce (near-)optimal solutions within minutes, whereas the exact method requires several hours. Moreover, we establish evidence that the SSII algorithm performs slightly better when a prior is specified.

2 Background

In this section, we first discuss *Markov decision processes* (MDPs) and the multi-objective variant MOMDP, which we use to model the problem of planning under uncertainty. We then discuss an algorithm to solve MOMDPs exactly given a linear scalarisation function.

2.1 Markov Decision Processes

We model the planning problem as a *Markov decision process* (MDP) (Bellman 1957a). An MDP is a model for sequential decision-making under uncertainty in which an autonomous agent interacts with its environment. An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function and $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Re$ is a real-valued reward function.

The agent aims to compute a *policy* $\pi : \mathcal{S} \times t \to \mathcal{A}$ (where t indicates time) that optimises the *expected return*, or *value function*, defined by the Bellman equation for MDPs:

$$V^\pi(s,t) = \sum_{s' \in \mathcal{S}} T(s, \pi(s,t), s')(R(s, \pi(s,t), s') + V^\pi(s', t+1)).$$

We can determine a state-independent value V^π by fixing $t = 0$ and marginalising over an initial state distribution μ_0. For convenience we consider only a finite-horizon setting.

Finding optimal policies can be done via standard stochastic solving techniques such as SPUDD (Hoey et al. 1999). SPUDD uses algebraic decision diagrams to represent values and states compactly, and finds the optimal policy through a modified version of structured value iteration (maximising the above equation). Since exact MDP solution methods suffer from the infamous curse of dimensionality, we have to rely on approximate methods for large MDPs. In this work we consider the UCT* algorithm (Keller and Helmert 2013), which evaluates states in a Monte-Carlo fashion and outputs partial policies, i.e., an (estimated) best action for each state-time pair it has evaluated.

2.2 MOMDP

A *multi-objective Markov decision process* (MOMDP) (Roijers et al. 2013) is a tuple $\langle \mathcal{S}, \mathcal{A}, T, \mathbf{R} \rangle$, where $\mathcal{S}$, $\mathcal{A}$ and T are specified as before, but $\mathbf{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Re^d$ is a d-dimensional vector-valued reward function. In this case the Bellman equation specifies a multi-objective value function:

$$\mathbf{V}^\pi(s,t) = \sum_{s' \in \mathcal{S}} T(s, \pi(s,t), s')(\mathbf{R}(s, \pi(s,t), s') + \mathbf{V}^\pi(s', t+1)).$$

As before, we can obtain the state-independent value of a policy $\mathbf{V}^\pi$ by marginalising over the initial state distribution, and filling in $t = 0$.

The presence of multiple objectives does not necessarily mean that special solution methods are required. If the problem can be *scalarised*, i.e., converted to a problem with a scalar reward function R_{so}, the problem may be solvable with existing single-objective methods. A *scalarisation function* f converts a payoff vector (either reward $\mathbf{R}$ or value $\mathbf{V}^\pi$) to a scalar, using parameter $\mathbf{w}$ that expresses the relative importance of each objective. In order to scalarise an MOMDP to an MDP, the values $V^\pi_{so}(s,t)$ of the scalarised problem must be consistent with the scalarised value of the original problem, $f(\mathbf{V}^\pi(s,t),\mathbf{w})$ for all policies π, states s and time steps t. To achieve this, the reward function needs to be scalarised to a single objective reward R_{so}:

$$V^\pi_{so}(s,t) = f(\mathbf{V}^\pi(s,t),\mathbf{w}) = \sum_{s' \in \mathcal{S}} T(s, \pi(s,t), s') \left(R_{so}(s, \pi(s,t), s') + V^\pi_{so}(s', t+1)\right).$$

This equation holds only if f distributes over the sum of rewards, which is true only if f is linear, i.e., $f(\mathbf{V}^\pi, \mathbf{w}) = \mathbf{w} \cdot \mathbf{V}^\pi$, where '$\cdot$' denotes the inner product:

$$V^\pi_{so}(s,t) = f(\mathbf{V}^\pi(s,t),\mathbf{w}) = \sum_{s' \in \mathcal{S}} T(s, \pi(s,t), s') \left(\mathbf{w} \cdot \mathbf{R}(s, \pi(s,t), s') + \mathbf{w} \cdot \mathbf{V}^\pi(s', t+1)\right).$$

After scalarisation with a specific $\mathbf{w}$, the problem becomes a regular MDP and can be solved using the previously mentioned methods to find (approximately) optimal policies.

However, a priori scalarisation of the problem is not possible when the weights of the scalarisation function are not known in advance. When, for example, a group of human decision makers needs to decide on which policy to follow, an a priori specification would require specifying the preferences of the group for every hypothetical multi-objective value in advance. This is a very difficult process, and humans are notoriously bad at it. It is much easier for humans to be presented with a set of alternatives, and decide which

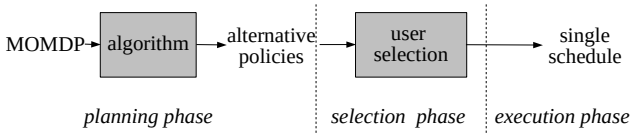


Figure 1: Decision support for MOMDPs

element of this set would be most preferred (Clemen 1997). Such a scenario is called decision support, i.e., the actual values for alternatives $\mathbf{V}^\pi$ are input to the discussion between human decision makers (Clemen 1997), as illustrated in Figure 1. In this case, we need an MOMDP method that returns the best possible value $\mathbf{V}^\pi$ for each possible $\mathbf{w}$, from which in the *selection phase* human decision makers can select one to execute.

The solution to an MOMDP for linear scalarisations is called the convex coverage set (CCS)[2,3]

$$CCS = \left\{ \mathbf{V}^\pi : \exists \mathbf{w} \forall \pi' \; \mathbf{w} \cdot \mathbf{V}^\pi \geq \mathbf{w} \cdot \mathbf{V}^{\pi'} \right\}.$$

We assume that all objectives are desirable, i.e., all weights are positive. And, because only relative importance between the objectives is important when computing optimal solutions, we assume without loss of generality that the weights lie between 0 and 1 and sum to 1. To compute the CCS we can thus restrict ourselves to weights on a simplex (see Figure 2).

In order to find the CCS we can make use of specialised algorithms. One such method is *optimistic linear support* (Roijers, Whiteson, and Oliehoek 2014) which was originally proposed for multi-objective coordination graphs but can be applied to MOMDPs as well. OLS requires a subroutine that solves a linearly scalarised version of a problem exactly. We first explain OLS and known MDP techniques that we use to form an MOMDP method.

2.3 Optimistic Linear Support

Roijers, Whiteson, and Oliehoek (2014) propose a CCS algorithm called *optimistic linear support* (OLS), that works by computing the exact solution to weighted MDPs for specifically chosen weights by scalarising a multi-objective decision problem and running a single-objective solver. They show that the single-objective method only needs to be called for a finite number of weights in order to obtain the exact CCS. In order to see why only a finite number of weights are necessary to compute the full CCS, we first need to define the scalarised value function:

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

and note that this is a *piecewise-linear and convex* (PWLC) function. Note that we only need to define this function on the weight simplex. We plot the PWLC function V^*_{CCS} of a two-objective problem as a function of the weight for the first objective in Figure 2a. Because the scalarisation is linear, the value of a policy becomes a line. The function V^*_{CCS} is indicated by the bold line segments.

OLS builds the CCS incrementally from an initially empty partial CCS, S, adding vectors that are optimal for specific weights on the weight simplex. In order to find the optimal solution for a specific weight, OLS runs an exact single objective solver `solveMDP` on an instance of the MOMDP scalarised using $\mathbf{w}$. S contains the known value vectors, i.e., the vectors we know are definitely in the CCS. The scalarised value function over S is defined similarly as the one for the complete CCS: $V^*_S(\mathbf{w}) = \max_{\mathbf{V}^\pi \in S} \mathbf{w} \cdot \mathbf{V}^\pi$, and again, is PWLC.

OLS starts by looking at the extrema of the weight simplex. Figure 2a shows the two value vectors found at the extrema of the weight simplex of a 2-objective example. The weights at which the vectors are found are indicated with bold red vertical lines. The area above the function $V^*_S(\mathbf{w})$ is a polyhedron (Bertsimas and Tsitsiklis 1997) whose vertices correspond to unique weights called *corner weights*. Using only the corner weights, OLS can find the exact CCS. This is possible due to a theorem by Cheng (1988) that states that there is a corner weight of $V^*_S(\mathbf{w})$ that maximises $\Delta(\mathbf{w}) = V^*_{CCS}(\mathbf{w}) - V^*_S(\mathbf{w})$, i.e., the maximal improvement of S that can be made by adding a value vector to it, with respect to the true CCS, is at one of the corner weights. This results from two observations: the values at the corner weights of $V^*_S(\mathbf{w})$ are smaller than or equal to those of $V^*_{CCS}(\mathbf{w})$ by definition and $S \subseteq CCS$. Therefore the maximal error can only be in a region for which the optimal vector is not in S. Under this region the difference $V^*_{CCS}(\mathbf{w}) - V^*_S(\mathbf{w})$ is a concave function, with identical corner weights $V^*_S(\mathbf{w})$ that hence necessarily specify the unique maximum for these regions.

Because of Cheng's theorem, if all the corner points are checked and no improvement has been found, the residual error must be zero. OLS checks the corner weights of S, prioritising on maximal possible improvement. As the exact CCS is unknown, OLS measures the maximal possible improvement with respect to the most optimistic hypothetical CCS, $\overline{CCS}$, that is still consistent with S, and the values $V^*_S(\mathbf{w})$ at the weights where the vectors in S have been found (at these weights OLS knows that $V^*_S(\mathbf{w}) = V^*_{CCS}(\mathbf{w}) = V^*_{\overline{CCS}}(\mathbf{w})$).

Figure 2 illustrates how the process of checking at corner weights leads to convergence. In (a), all value vectors are shown for an example problem. In (b) OLS has discovered the optimal value vectors for the extrema of the weight space, and identified a new corner weight. This corner weight is checked, leading to a new vector and two new corner weights. Then, in (c) OLS has checked both these corner weights but has not found a new vector, and has thus converged. The corner weights that have been checked are indicated with red vertical lines. Note that OLS thus needs to check only 5 corner weights to discover the CCS.

[2]The convex coverage set is often called the *convex hull*. We avoid this term because it is imprecise: the convex hull is a superset of the convex coverage set.

[3]The CCS is a subset of the Pareto front. Please refer to (Roijers et al. 2013) for an extensive discussion on both.

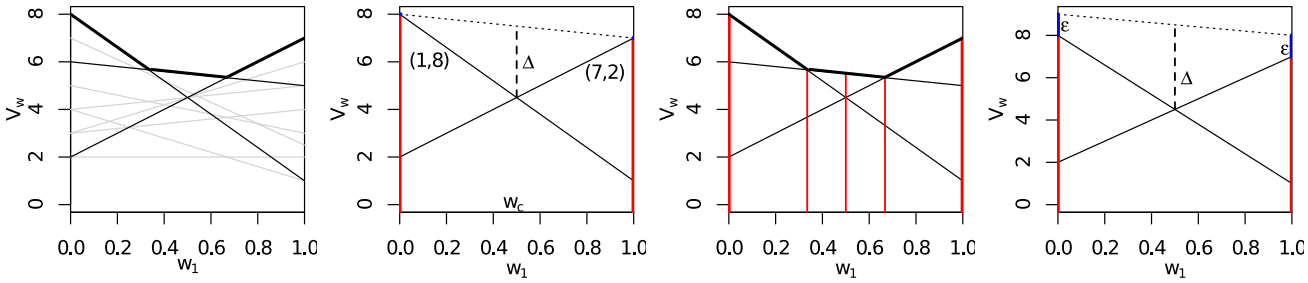


Figure 2: (a) All possible value vectors for a 2-objective MOMDP. (b) OLS finds two payoff vectors at the extrema $\mathbf{w} = (0, 1)$ and $\mathbf{w} = (1, 0)$, a new corner weight $\mathbf{w}_c = (0.5, 0.5)$ is found, with maximal possible improvement Δ. $\overline{CCS}$ is shown as the dotted line. (c) OLS finds a new vector at $(0.5, 0.5)$, resulting in two new corner points which do not result in new vectors, making $S = \overline{CCS} = CCS$. (d) AOLS computes the priorities using an adjusted optimistic CCS.

3 Approximate Optimistic Linear Support

OLS can find the exact CCS only when we can find exact solutions for all weights. This is an important limitation, because it is often computationally infeasible to compute exact solutions for the (often large) MDPs used to model stochastic planning problems. In this section, we propose *approximate OLS* (AOLS), which does not require an exact MDP solver. Because AOLS employs approximate MDP solvers, the so-called corner weights need to be recomputed everytime we find a new value vector, which is not necessary in OLS. Furthermore, the computation of maximal possible error and the next weight for which to call the MDP solver are different from OLS. We establish both a convergence guarantee and show that AOLS provides a bounded approximation *over all weights*.

First, we assume that instead of an exact algorithm for `solveMDP` we have an ϵ- approximate algorithm.

Definition 1. *An* ϵ-approximate MDP solver *is an algorithm that produces a policy, whose expected value is at least* $(1 - \epsilon)V^*$*, where* V^* *is the optimal value for the MDP and* $\epsilon \geq 0$.

Given an ϵ-approximate MDP solver we can compute a set of policies for which the scalarised value for each possible $\mathbf{w}$ is at most a factor ϵ away from the optimum (ϵ-CCS). In Section 3.2 we show that AOLS can still be applied even without this assumption, though with weaker guarantees.

AOLS, shown in Algorithm 1, takes as input an MOMDP M, and a single objective MDP solver `solveMDP` that is ϵ-approximate. AOLS maintains the set of value vectors and associated policies in a set S (initialised on line 1). It keeps track of the tuples of weights already searched with their corresponding scalarised values (initialised on line 2). AOLS starts by selecting the first extremum (the unit vector $\mathbf{e}_1$, e.g. $(0, 1)$ or $(1, 0)$ in the two-objective case) of the weight simplex with an infinite maximal possible improvement (line 3 and 4). Until there is no corner weight with a maximal relative improvement of at least ϵ left (line 5), AOLS scalarises the MOMDP M with weight $\mathbf{w}_{max}$ and computes an ϵ-optimal policy π (line 6). AOLS then uses policy evaluation to determine the expected value of said policy (line 7) (Bellman 1957b). This evaluation step can be expensive, but is usually much less expensive than finding an

Algorithm 1: `AOLS`$(M,$ `solveMDP`$, \epsilon)$

```
1  S ← ∅ //partial CCS
2  WV_old ← ∅ //searched weights and scalarised values
3  Δ_max ← ∞
4  w_max ← e_1
5  while Δ_max > ε ∧ ¬timeOut do
6      π ← solveMDP(scalarise(M, w_max))
7      V^π ← evaluatePolicy(M,π)
8      WV_old = WV_old ∪ {(w_max, w_max · V^π)}
9      if V^π ∉ S then
10         S ← S ∪ {V^π}
11         W ← recompute corner weights V*_S(w)
12         (w_max, Δ_max) ← calcOCCS(S, WV_old, W, ε)
13     end
14 end
15 return S, Δ_max
```

(ϵ-)optimal policy. After computing the value, the searched weight and corresponding scalarised value is added to the set WV_{old} (line 8), which is necessary to compute the optimistic CCS.

If the vector $\mathbf{V}^\pi$ found at $\mathbf{w}_{max}$ is new, i.e., not in S, it is added to this solution set (line 10). However, because these values are approximate values, it is not certain that they are optimal anywhere. Therefore, contrary to OLS, we need to test whether any of the vectors in $S \cup \{\mathbf{V}^\pi\}$ is dominated, and recompute all the corner weights accordingly (line 11). This recomputation can be done by solving a system of linear equations. For all corner weights, we compute the maximal possible improvement $\Delta(\mathbf{w}) = V^*_{\overline{CCS}}(\mathbf{w}) - V^*_S(\mathbf{w})$. This happens in Algorithm 2, in which for each corner weight, $\Delta(\mathbf{w})$ is evaluated, using a linear program.

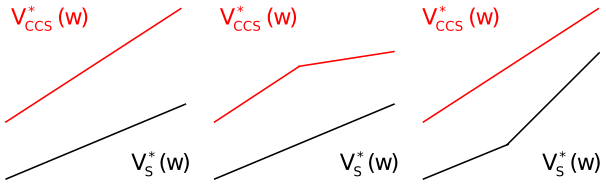
The definition of the optimistic CCS is different than the CCS used in OLS because it incorporates the possibility of being ϵ-wrong, shifting the scalarised value $V^*_{\overline{CCS}}(\mathbf{w})$ upwards (as shown in Figure 2d). The maximal scalarised value of $V^*_{\overline{CCS}}(\mathbf{w})$ for a given $\mathbf{w}$ can be found by solving the linear program given on line 6 in Algorithm 2. This linear program is different than that of OLS, because of the ϵ term in the constraints. By prioritising weights according to their maximal possible improvement, AOLS becomes an

Algorithm 2: `calcOCCS`(S, WV, W, ϵ)

for $i \in 1...length(\mathbf{w})$ **do**
  if $\mathbf{e}_i \notin WV$ **then**
    return $(\mathbf{e}_i, \infty)$ //extrema should be tested first
  end
end
$V_{CCS}[\cdot] \leftarrow$ for all weights in $W[\cdot]$ calculate:

$$\max \ \mathbf{w} \cdot \mathbf{v}$$
$$\text{subject to} \ \ \forall (\mathbf{w}, u) \in WV : \ \mathbf{w} \cdot \mathbf{v} \leq u + \epsilon$$

$i \leftarrow \arg\max_i V_{CCS}[i] - V_S^*(W[i])$
return $(W[i], \ V_{CCS}[i] - V_S^*(W[i]))$



(a) (b) (c)

Figure 3: Possible cases for the maximum possible improvement of S with respect to the CCS.

anytime algorithm, i.e., the highest value of $\Delta(\mathbf{w})$ is the current ϵ of an ϵ-CCS.

3.1 Correctness

We now establish the correctness of AOLS. Because the scalarised value of $V^*_{\overline{CCS}}(\mathbf{w})$ (as computed by Algorithm 2) obtained through using the ϵ of the approximate `solveMDP` is no longer identical to $V_S^*(\mathbf{w})$, we need to make an adjustment to Cheng's theorem.

Theorem 1. *There is a corner weight of $V_S^*(\mathbf{w})$ that maximises:*

$$\Delta(\mathbf{w}) = V^*_{CCS}(\mathbf{w}) - V_S^*(\mathbf{w}),$$

where S is an intermediate set of value vectors computed by AOLS.

This theorem is identical in form to Cheng's, but, because S is no longer a subset of CCS, the proof is different.

Proof. $\Delta(\mathbf{w})$ is a PWLC function because it is constructed as the difference between two PWLC functions. To maximise this function, we have three possible cases, shown in Figure 3: (a) the maximum is at a weight that is neither a corner point of $V^*_{CCS}(\mathbf{w})$, nor of $V_S^*(\mathbf{w})$; (b) it is at a corner point of $V^*_{CCS}(\mathbf{w})$ but not of $V_S^*(\mathbf{w})$; or (c) it is at a corner point of $V_S^*(\mathbf{w})$.

Case (a) only applies if the slope of $\Delta(\mathbf{w})$ is 0, and therefore the value is never higher than the value at the corner points where this slope changes. Case (b) can never occur, because if $\mathbf{w}$ is a corner point of CCS, and not of S, and $V^*_{CCS}(\mathbf{w}) - V_S^*(\mathbf{w})$ is at a maximum, then $V^*_{CCS}(\mathbf{w})$ must be at a maximum at $\mathbf{w}$. If $V^*_{CCS}(\mathbf{w})$ is at a maximum at a weight $\mathbf{w}$ then it cannot be a PWLC function, leading to a contradiction. Therefore, only case (c) remains. □

Because the maximum possible improvement is still at the corner points of S, even though S now contains ϵ-approximate solutions, the original scheme of calling `solveMDP` for the corner weights still applies.

AOLS terminates when the maximal possible improvement is smaller than or equal to $\epsilon \cdot V^*_{\overline{CCS}}(\mathbf{w})$, i.e., corner weights with a possible improvement less than that are no longer considered.

Theorem 2. *AOLS terminates after a finite number of calls to an ϵ-approximate implementation of* `solveMDP` *and produces an ϵ-CCS.*

Proof. AOLS runs until there are no corner points left in the priority queue to check, and returns S. Once a corner point is evaluated, it is never considered again because the established value lies within $(1-\epsilon)V^*_{\overline{CCS}}(\mathbf{w})$. AOLS thus terminates after checking a finite number of corner weights. All other corner weights have a possible improvement less than $\epsilon V^*_{\overline{CCS}}(\mathbf{w})$. Therefore S must be an ϵ-CCS. □

3.2 Unbounded Approximations

When the ϵ that `solveMDP` guarantees is known, Theorem 1 guarantees an ϵ-CCS. When `solveMDP` uses an approximate implementation where ϵ is not known beforehand, AOLS still converges on some approximation of the CCS. but, since ϵ is unknown, it cannot prioritise which corner weight to check first. Therefore, Δ_{max} cannot be computed, and AOLS selects a random yet unchecked corner weight. In this case. AOLS cannot guarantee a certain ϵ. In Section 5, we use such an implementation (PROST) and determine the ϵ empirically by comparing against an exact implementation of `solveMDP` (SPUDD).

4 Scalarised Sample-based Incremental Improvement

If no a priori information about the weights is known, the AOLS algorithm can successfully approximate the CCS for all possible weights and, when the single-objective subroutine is bounded, AOLS provides an ϵ-CCS. However, decision makers can often provide an indication as to the preferences they might have. We can expression this information using a prior distribution over the weights that we can exploit, e.g., if they cannot exactly specify the economic impact of delay but instead know a distribution over its potential impact cost.

In such scenarios, AOLS might not be the best approach since it distributes its planning effort equally over the corner weights (which might not be in the right region), i.e., the allowed time budget for the single objective planner for each corner weight is the same. Approximate single objective methods such as UCT* provide better approximations when given more time. If we can focus on important weights we may be able to establish a better ϵ in the regions *where it matters most*, and as a result discover more relevant available trade-offs than possible without a prior.

We propose the *scalarised sample-based incremental improvement* (SSII) algorithm (Algorithm 3) that uses the prior to sample weights, thereby focusing more attention on the

Algorithm 3: SSII(M, solveMDP, pr, N, $timeOut$)

1 $weightsList \leftarrow$ sample N weights from pr
2 **for** $i \in [1...N]$ **do**
3 $\quad t_i \leftarrow$ minimum runtime solveMDP
4 $\quad vectorList[i] \leftarrow$ run solveMDP$(m, weightsList[i], t_i)$
5 **end**
6 **while** $\neg$*timeOut* **do**
7 $\quad (\mathbf{w}_{max}, \Delta_{max}) \leftarrow$ calcOCCS$(vectorList, weightsList, \epsilon)$
8 $\quad$ get i for which $\mathbf{w}_{max} = weightsList[i]$ and increase t_i
9 $\quad vectorList[i] \leftarrow$cont. solveMDP$(M, weightsList[i], t_i)$
10 **end**
11 **return** $vectorList$

most interesting regions. The algorithm takes as input the MOMDP M, an anytime single objective solver solveMDP, a prior distribution over weights pr, the number of samples N, and a timeout $timeOut$; it produces an optimistic CCS.

For each weight $\mathbf{w}$ sampled from the prior distribution pr, the algorithm quickly computes an initial approximate policy and associated value vector $\hat{\mathbf{V}}^{\pi}_{\mathbf{w}}$ (lines 1-5). After determining this initial set of value vectors, it improves on the expected reward by incrementally allowing the anytime solver more runtime to improve the solution for one sample at a time (lines 6-10). Note that we abuse the notation of the call to calcOCCS, the argument WV is composed of $vectorList$ and $weightsList$, and S is just $vectorList$ and W is just $weightsList$.

The incremental improvement per sample is guided by a procedure that prioritises these weight samples. At each iteration, one sample is selected, and SSII will continue to improve until a specified time limit has been reached for the runtime of the single objective MDP solver (line 9). Improving on one of vectors by running solveMDP does not require a full rerun; instead solveMDP saves its state and resumes from there. In UCT*, we can preserve the search tree and continue from there if an improvement of the same sample weight is required.

Selecting which sampled weight $\mathbf{w}$ to improve can be done in various ways, yielding different anytime and final solutions. Our implementation of SSII prioritises the samples based on their improvement $\Delta(\mathbf{w})$ as defined in the previous section. Intuitively, this will focus the SSII algorithm on samples with the highest potential improvement.

5 Experiments

In this section, we present an empirical analysis of the performance of the AOLS and SSII algorithms in terms of both runtime and quality of the CCS. In order to illustrate the runtime difference with exact methods, we compare against the exact OLS/SPUDD algorithm. Both AOLS and SSII use the UCT* algorithm to approximate policies. Because we cannot let UCT* run until all states have been evaluated, we can only obtain a partial policy. In our experiments we 'repair' partial policies, i.e., supplement these policies by taking a conservative action; in the problems we consider this means inserting no-ops and corresponding result states (see next section), thereby not changing the expected value of the policy. Note that one could use any heuristic to supplement the partial policies.

Our experiments are performed on the *maintenance planning problem* (MPP) (Scharpff et al. 2013). Using generated instances of this problem, we measure the error with respect to the exact CCS. The experiments use a JAVA implementation that calls the SPUDD and PROST packages.[4] All experiments were conducted on an Intel i7 1.60 GHz computer with 6 GB memory.

5.1 Maintenance Planning

The *maintenance planning problem* consists of a set of contractors, i.e. the agents, N and a period of discrete time steps $T = [t_1, t_2, \ldots, t_h]$ where h is the horizon length. Every agent $i \in N$ is responsible for a set A_i of maintenance activities. The main goal in this problem is to find a contingent maintenance schedule that minimises both the cost as well as the traffic hindrance.

The maintenance activities $a \in A_i$ are defined by tuples $\langle w, d, p, \hat{d} \rangle$ where $w \in \mathbb{R}^+$ is the payment upon completion of the task, $d \in \mathbb{Z}^+$ is the number of consecutive time steps required to complete the task, $p \in [0, 1]$ the probability that a task will delay and $\hat{d} \in \mathbb{Z}^+$ the additional duration if the task is delayed. Each agent also has a cost function $c_i(a, t)$ that captures the cost of maintenance (e.g. personnel, materials, etc.) for all of its activities $a \in A_i$ and every time step $t \in T$. Agents are restricted to executing at most a single maintenance activity and they can only plan activities if they are guaranteed to finish them within the plan horizon. The utility of an agent is defined by the sum of payments minus the costs for every completed activity and therefore depends on the executed maintenance schedule P_i. When we assume that P_i contains for each time step $t \in T$ the activity that was executed we can write this utility as $u_i = \sum_{a_i \in P_i} w_{a_i} - \sum_{t \in T} c(P_i(t), t)$.

The impact of maintenance on the network throughput is given by the function $\ell(A^t, t)$, expressed in traffic time lost (TTL). In this function, A^t is used to denote any combination of maintenance activities being performed at time t, by one or more agents. This function allows us to employ arbitrary models, e.g., extrapolated from past results, obtained through (micro-)simulation or computed using heuristic rules.

We model the MPP as an MOMDP in which the state is defined as the activities yet to be performed, the time left to perform them, and the availability of the individual agents. Because the agents are *cooperative* we can treat the set of agents as a single agent: the actions at each time step are the possible combinations of actions that could be started at that time step, i.e., the Cartesian product of the possible actions for each agent. The activities have an uncertain completion time, as maintenance activities can run into delays, resulting in a stochastic transition function. The reward is of the form $\sum_{i \in N} u_i - \sum_t \ell(A^t, t)$

[4] These can be found at www.computing.dundee.ac.uk/staff/jessehoey/spudd/ and prost.informatik.uni-freiburg.de/, respectively.

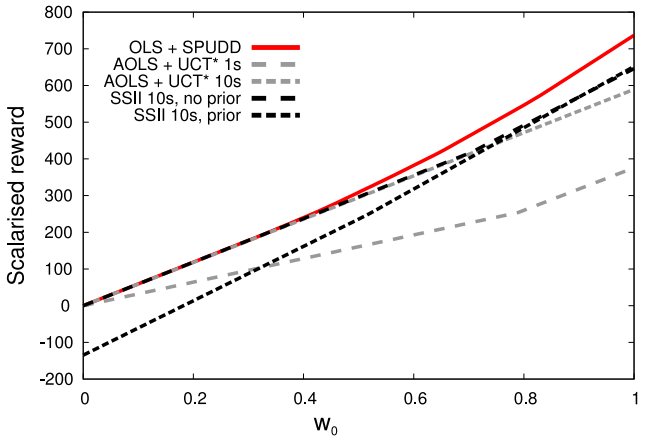


Figure 4: Typical example of solution sets found by AOLS, SSII and OLS/SPUDD, using different runtimes.

For our experiments, we generated many instances of the problem with various sizes. All of them are *fully coupled*, i.e., for each pair $a_i \in A_i, a_j \in A_j$ there exists a network cost $\ell(a_i, a_j) > 0$.

5.2 CCS Quality Metrics

The error of the coverage sets obtained through AOLS and SSII is defined with respect to the actual CCS (obtained through OLS/SPUDD). We describe the output sets of AOLS or SSII as S, and the actual CCS as CCS. We define two error metrics: maximal and expected error.

The maximal error ε_{max} is a conservative measure that specifies the maximal scalarised error over all possible $\mathbf{w}$. This measure does not include any prior information over the weights:

$$\varepsilon_{max}(S, V^*_{CCS}) = \max_{\mathbf{w}} V^*_{CCS}(\mathbf{w}) - V^*_S(\mathbf{w}).$$

Note that this metric is absolute rather than relative.

The expected error incorporates the prior distribution over $\mathbf{w}$, pr in order to give a larger penalty to errors in more important regions:

$$\varepsilon_{exp}(S, V^*_{CCS}) = \int_{\mathbf{w}} pr(\mathbf{w})(V^*_{CCS}(\mathbf{w}) - V^*_S(\mathbf{w}))d\mathbf{w}.$$

5.3 Comparison of Algorithms

In order to test the performance of AOLS and SSII in terms of runtime and solution quality, we apply these algorithms to the MPP domain. We take moderately sized instances so that it is possible to find the exact CCS for comparison. To compute the exact CCS, we apply OLS with exact solutions found by SPUDD (Hoey et al. 1999) as the exact single-objective MDP solver subroutine. The approximation of policies for both AOLS and SSII is done using the UCT* algorithm from the PROST toolbox (Keller and Eyerich 2012). UCT* is an anytime algorithm which is expected to perform better when it is given more time to compute approximate solutions. Therefore, we test AOLS and SSII with various settings for the allowed runtime for UCT*. For SSII we ran both experiments with and without prior knowledge about weights. We use limited information of the form "costs are more important than TTLs" leading to a uniform prior in a limited range of weights, $[0.5, 1]$ for the cost weight (and hence $[0, 0.5]$ for the TTL weight).

To develop an intuition on the quality of the solution sets found by AOLS and SSII for varying runtimes, we first test them on a single test problem, with two maintenance companies and three activities each, and visualise the resulting solution sets S by plotting $V^*_S(\mathbf{w})$, as shown in Figure 4. SSII was given a uniform prior on the interval [0.5,1]. For low runtimes we observe that AOLS is better than SSII where the prior has probability mass 0, i.e., $w_1 < 0.5$, but SSII is better in the region it focuses on. For higher runtimes, AOLS and SSII both happen to find a good vector for the left side of the weight space, and both methods find the same value for $w_1 = 0.5$.

Neither AOLS nor SSII achieves the value of the exact CCS with their solution sets. However, much less runtime was used compared to finding the optimum. OLS/SPUDD used around 3 hours, while AOLS used close to 6 minutes, and SSII just over 7 minutes, both with $10s$ allowed for the single objective solver.

In order to test the error with respect to the CCS numerically, we averaged over 12 MPP instances with 2 maintenance companies with 3 activities each with randomly (uniformly) drawn horizons between 5 and 15. Table 1 shows the average performance per algorithm, expressed in terms of ε_{max}, ε_{exp} and the fraction of instances solved optimally. The exact solver was run once for each problem, whereas the AOLS and SSII algorithms were run 5 times on the same problem to compute the average performances and runtimes, because UCT* is a randomised algorithm.

These results show a striking difference between the exact and approximation runtimes. Whereas the average runtime required to find the optimum CCS is in the range of hours (the longest took 3.5 hours), SSII and AOLS require time in the order of seconds to a few minutes to find an optimistic CCS. Moreover, our experiments confirm that the maximal and expected errors decrease when more runtime is given. Also very interesting is that both AOLS and SSII are able to find a (near-)optimal CCS in many cases, where OLS takes hours to do the same, and that SSII is competitive with AOLS.

In order to demonstrate that prior information on the distribution over the weight can be exploited, we ran SSII with two different sample ranges. 'SSII no prior' used 5 evenly distributed samples over the entire region, i.e., assuming no prior information is available. The 'SSII prior' algorithm, also distributed its samples evenly, but only in the range $[0.5, 1]$. Note that any CCS found with samples in this range is also a CCS for the full weight range, although not likely to be close to optimal outside the sample range. The results in the last three columns show the errors for only the range of the prior, illustrating that both the maximal error, expected error and even the percentage of optimal solutions found *within this range* is slightly better when exploiting the prior.

			$[0, 1]$			$[0.5, 1]$		
Algorithm	Runtime	$\|CCS\|$	ε_{exp}	ε_{max}	%OPT	ε_{exp}	ε_{max}	%OPT
OLS + SPUDD	2390.819	9.250	-	-	-	-	-	-
AOLS + UCT* 0.01s	8.612	3.389	0.701	325.354	0.000	0.692	325.025	0.000
AOLS + UCT* 1s	19.940	4.111	0.119	65.668	0.167	0.117	65.426	0.167
AOLS + UCT* 10s	65.478	4.528	0.084	56.439	0.333	0.091	56.381	0.333
AOLS + UCT* 20s	165.873	5.694	0.044	38.667	0.417	0.048	38.627	0.417
SSII 1s, no prior	18.795	4.306	0.118	70.244	0.167	0.116	70.195	0.167
SSII 10s, no prior	59.336	3.889	0.061	51.800	0.333	0.068	51.747	0.333
SSII 1s, prior	17.892	3.944	0.221	95.189	0.000	0.125	61.667	0.167
SSII 10s, prior	59.154	4.083	0.141	71.290	0.083	0.057	43.006	0.333

Table 1: Comparison of averaged performance of the algorithms presented in this paper for various parameters, shown for two regions of the scalarised reward space. Runtimes are in seconds, the expected error ε_{exp} and maximum error ε_{max} are relative to the optimum CCS and %OPT denotes the fraction of instances that were solved optimally.

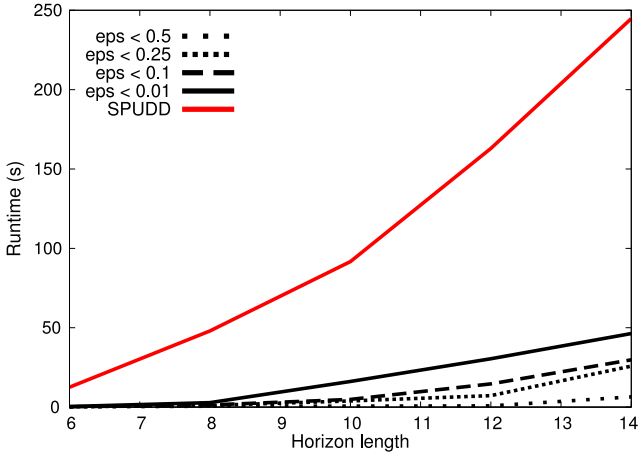


Figure 5: Average runtime for UCT* to attain various ϵ levels and the runtime required to compute the optimal policy using SPUDD.

5.4 UCT* approximation quality

Although the UCT* algorithm does not provide a guarantee on the resulting policy value, we establish empirically that, given sufficient runtime, it can produce ϵ-approximations for various values of ϵ. Using the same instances as before, we determine the average runtime that is required to obtain such approximation levels. For each instance, we run the UCT* algorithm 10 times, where we iteratively increase the allowed runtime in each run until a solution of the required quality is obtained. Figure 5 shows the average runtimes in seconds required to obtain various ϵ compared to the runtime of SPUDD, for different horizon lengths. From the figure we can see that UCT* obtains at least half of the optimal value within mere seconds. Interestingly, UCT* is also able to produce near-optimal solutions within a minute for these problem instances (sometimes the optimum is found, see Table 1). Unfortunately, we do not know whether these results translate to larger problem sizes, as a comparison with exact solutions is computationally infeasible for bigger problems.

6 Related Work

Several other methods exist for computing the CCS MOMDPs. Viswanathan, Aggarwal, and Nair (1977) devise a linear programming method for episodic MOMDPs; Wakuta and Togawa (1998) use a policy iteration based method and Barrett and Narayanan (2008) extend value iteration with CCS computations in the back-up phase to form convex hull value iteration (CHVI). However, unlike our approach, all these methods assume the state space is small enough to allow exact solution. Moreover, they deal with the multiple objectives on a low level, deep inside the algorithms. In contrast, we deal with the multiple objectives in a modular fashion, i.e., both AOLS and SSII have an outer loop to deal with the multiple objectives and call approximate single-objective MDP solvers as a subroutine. We can thus easily replace these subroutines with any future method.

The closest MOMDP method to our work is Lizotte, Bowling, and Murphy (2010)'s finite-horizon multi-objective value iteration algorithm for continuous state-spaces, and its extended version (Lizotte, Bowling, and Murphy 2012). This method plans backwards from the planning horizon, computing a point-based (i.e., using sampled weights) value backup for each timestep until $t = 0$ is reached. However, their approach is infeasible in our setting because of the large number of reachable end states. Instead, UCT* performs a forward Monte-Carlo tree search starting at $t = 0$, reducing the number of end states visited.

7 Conclusions and Future Work

Solving MDPs with multiple objectives is challenging when the relative importance of objectives is unclear or impossible to determine a priori. When MOMDPs can be linearly scalarised, we should find the set of alternatives – the convex coverage set – which for each choice of weight span the maximal achievable expected rewards. Finding the exact convex coverage set is however often not feasible, because finding exact solution of a (single-objective) MDP is already polynomial in the number of states, and the number of states in many problems is huge. In this work we therefore focused on methods to approximate the CCS for MOMDPs.

Our main contributions are the approximate optimistic linear support (AOLS) and scalarised sample-based incremental improvement (SSII) algorithms. For AOLS, we proved a modified version of Cheng's theorem (Cheng 1988) such that the main results for OLS also hold when using

an approximation method to compute the multi-objective value at corner weights (Theorem 1). Moreover, when an ϵ-approximation algorithm is used, we can bound the loss in scalarised value of the resulting approximate CCS.

Although AOLS is generally applicable, it cannot exploit prior information on the distribution of weights. When such a prior is available we want to focus most of our attention to the regions of the CCS that correspond to this distribution. Instead, SSII samples weights from this prior for which it iteratively tries to improve the policy, prioritising the samples that are the furthest away from the best known CCS.

We compared the performance of the approximate algorithms proposed in this paper with the optimal algorithm in terms of runtime, expected CCS error and maximum CCS error, on problems from the maintenance planning domain (Scharpff et al. 2013). These experiments show that both algorithms can compete with the exact algorithm in terms of quality, but require several orders of magnitude less runtime. When a prior is known, our experiments show that SSII has a slight advantage over AOLS.

In this paper, we assume that it is feasible to perform exact evaluation of $\mathbf{V}^{\pi}$, i.e., the multi-objective value. In MPP this assumption appears valid. In general, however, this also becomes an expensive operation for large problems. An important direction of future research, therefore, is to investigate if it is possible to extend our results to the case of approximate evaluation of $\mathbf{V}^{\pi}$.

Another direction for future work is to combine the two main ideas presented in this work for cases when no single objective solver with a known ϵ bound is available. We are interested in improving the performance of AOLS by applying a heuristic exploiting a prior on the weights to decide on which corner point to evaluate.

This work assumes that the agents are fully cooperative, but many problems exists where this is not the case. For instance in the MPP domain, when agents are self-interested they will not likely give up profits in favour of traffic hindrance reduction unless they are explicitly motivated to do so. A mechanism design approach, similar to (Scharpff et al. 2013), could help to extend this work to the wider class of non-cooperative planning problems.

Acknowledgements

This research is supported by the NWO DTC-NCAP (#612.001.109), Next Generation Infrastructures/Almende BV and NWO VENI (#639.021.336) projects.

References

Alarcon-Rodriguez, A.; Haesen, E.; Ault, G.; Driesen, J.; and Belmans, R. 2009. Multi-objective Planning Framework for Stochastic and Controllable Distributed Energy Resources. *IET Renewable Power Generation* 3(2):227–238.

Barrett, L., and Narayanan, S. 2008. Learning All Optimal Policies with Multiple Criteria. In *Proc. of the International Conference on Machine Learning*, 41–47. New York, NY, USA: ACM.

Bellman, R. E. 1957a. A Markov Decision Process. *Journal of Mathematical Mechanics* 6:679–684.

Bellman, R. E. 1957b. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bertsimas, D., and Tsitsiklis, J. 1997. *Introduction to Linear Optimization*. Athena Scientific.

Calisi, D.; Farinelli, A.; Iocchi, L.; and Nardi, D. 2007. Multi-objective Exploration and Search for Autonomous Rescue Robots. *Journal of Field Robotics* 24(8-9):763–777.

Cheng, H.-T. 1988. *Algorithms for Partially Observable Markov Decision Processes*. Ph.D. Dissertation, UBC.

Clemen, R. T. 1997. *Making Hard Decisions: An Introduction to Decision Analysis*. South-Western College Pub, 2 edition.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic Planning Using Decision Diagrams. In *Proc. of the Fifteenth conference on Uncertainty in artificial intelligence*, 279–288.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *Proc. of the International Conference on Automated Planning and Scheduling*.

Keller, T., and Helmert, M. 2013. Trial-based Heuristic Tree Search for Finite Horizon MDPs. In *Proc. of the International Conference on Automated Planning and Scheduling*.

Lizotte, D. J.; Bowling, M.; and Murphy, S. A. 2010. Efficient Reinforcement Learning with Multiple Reward Functions for Randomized Clinical Trial Analysis. In *27th International Conference on Machine Learning*, 695–702.

Lizotte, D. J.; Bowling, M.; and Murphy, S. A. 2012. Linear Fitted-Q Iteration with Multiple Reward Functions. *Journal of Machine Learning Research* 13:3253–3295.

Mouaddib, A.-I. 2004. Multi-objective Decision-theoretic Path Planning. In *Proc. of the International Conference on Robotics and Automation*, volume 3, 2814–2819. IEEE.

Roijers, D. M.; Vamplew, P.; Whiteson, S.; and Dazeley, R. 2013. A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research* 47:67–113.

Roijers, D. M.; Whiteson, S.; and Oliehoek, F. 2014. Linear Support for Multi-Objective Coordination Graphs. In *Proc. of the Autonomous Agents and Multi-Agent Systems conference*.

Scharpff, J.; Spaan, M. T. J.; Volker, L.; and de Weerdt, M. 2013. Planning Under Uncertainty for Coordinating Infrastructural Maintenance. *Proc. of the International Conference on Automated Planning and Scheduling*.

Vamplew, P.; Dazeley, R.; Berry, A.; Issabekov, R.; and Dekker, E. 2011. Empirical Evaluation Methods for Multiobjective Reinforcement Learning Algorithms. *Machine Learning* 84(1-2):51–80.

Viswanathan, B.; Aggarwal, V. V.; and Nair, K. P. K. 1977. Multiple Criteria Markov Decision Processes. *TIMS Studies Management Science* 6:263–272.

Wakuta, K., and Togawa, K. 1998. Solution Procedures for Markov Decision Processes. *Optimization: A Journal of Mathematical Programming and Operations Research* 43(1):29–46.