ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 22-24 April 2015, i6doc.com publ., ISBN 978-287587014-8.
Available from http://www.i6doc.com/en/.

Pareto Local Policy Search
for MOMDP Planning

Chiel Kooijman, Maarten de Waard, Maarten Inja,
Diederik M. Roijers, and Shimon Whiteson

University of Amsterdam
Science Park 904, 1098 XH Amsterdam
The Netherlands

Abstract. Standard single-objective methods such as value iteration are not applicable to multi-objective Markov decision processes (MOMDPs) because they depend on a maximization, which is not defined if the rewards are multi-dimensional. As a result, special multi-objective algorithms are needed to find a set of policies that contains all optimal trade-offs between objectives, i.e. a set of Pareto optimal policies. In this paper, we propose Pareto local policy search (PLoPS), a new planning method for MOMDPs based on Pareto local search (PLS) [3]. This method produces a good set of policies by iteratively scanning the neighbourhood of locally non-dominated policies for improvements. It is fast because neighbouring policies can be quickly identified as improvements, and their values can be computed incrementally. We test the performance of PLoPS on several MOMDP benchmarks, and compare it to popular decision-theoretic and evolutionary alternatives. The results show that PLoPS outperforms the alternatives.

1 Introduction

Many real-world planning problems require reasoning about future states and rewards, while considering multiple possibly conflicting objectives [5]. An example is maintenance scheduling on a traffic network while minimising both the cost of operations and the hindrance for traffic. These problems can naturally be expressed as a *multi-objective Markov decision process (MOMDP)*. In MOMDPs, values of policies are vectors, making it impossible to maximise over them. Instead, the values are compared based on a *dominance* relationship, that states that the *scalarised utility* of a value vector is always better than the value of another with respect to a family of utility functions or *scalarization functions*. The most common such relationship is the Pareto dominance relationship defined with respect to the family of *monotonically increasing scalarization functions*. However, for arbitrary monotonically increasing scalarization functions, traditional dynamic programming methods for solving single-objective MDPs are not applicable because the Bellman optimality equation that such methods solve relies on the assumption of *additive returns*, which does not hold for the scalarised values [5]. Therefore, alternative approaches are needed to solve MOMDPs.

We propose a new algorithm, called *Pareto local policy search (PLoPS)*, that is based on *Pareto local search* [3] and quickly approximates the *Pareto front* of deterministic stationary policies. To do so, PLoPS performs an iterated neighbourhood search for policies with known values. PLoPS is fast because it

ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 22-24 April 2015, i6doc.com publ., ISBN 978-287587014-8. Available from http://www.i6doc.com/en/.

exploits the fact that neighbouring policies usually have similar values, and that it is easy to check whether a policy π with a known value is dominated by a neighbour π'.

We show empirically that PLoPS quickly finds good approximations on two known MOMDP benchmarks. We compare the performance of PLoPS to evolutionary and decision-theoretic alternatives, and show PLoPS outperforms them.

2 Related Work

Several planning methods exist for calculating the Pareto front of deterministic stationary policies in MOMDPs. CON-MODP [9, 10] is a multi-objective planning algorithm based on value iteration for multi-objective dynamic programming, that can only be used for MOMDPs with a deterministic transition function. CON-MODP can calculate the Pareto front of deterministic stationary policies by defining a CON operator that forces policies to be stationary after Bellman backups.

Multi-objective Monte Carlo tree search (MO-MCTS) [8] extends *Monte-Carlo tree search (MCTS)* to multi-objective sequential decision making, and can be used for both planning and learning. It uses a hypervolume indicator instead of the *upper confidence bound* often used in single-objective MCTS.

3 Background

In this paper, we aim to find the Pareto front of deterministic stationary policies in MOMDPs. An MOMDP is a tuple $\langle S, A, \mathcal{P}, \boldsymbol{\mathcal{R}}, \mu, \gamma \rangle$, where S is a finite set of states, A is a finite set of actions, $\mathcal{P} : S \times A \times S \to [0, 1]$ is a transition function that specifies for each state and action the probability of reaching the next state, and $\boldsymbol{\mathcal{R}} : S \times A \times S \to \mathbb{R}^D$ is the D-dimensional reward that specifies the immediate reward for each state, action and next state. μ denotes the probability distribution over the initial states and γ is a discount factor that specifies the relative importance of immediate rewards and future rewards.

A deterministic stationary policy π is a mapping from states to actions. Because we only consider this type of policy, we refer to these simply as policies. The value of a policy π can be calculated using *policy evaluation (PE)* [6][1]. PE repeatedly applies Bellman backups to the value $\mathbf{V}^\pi(s)$ of each state s until convergence. This update is defined by:

$$\mathbf{V}^\pi_{t+1}(s) \leftarrow \sum_{s'} \mathcal{P}^{\pi(s)}_{ss'} \left[\boldsymbol{\mathcal{R}}^{\pi(s)}_{ss'} + \gamma \mathbf{V}^\pi_t(s') \right], \tag{1}$$

where t indicates the number of iterations in PE. How many iterations it takes for PE to converge depends on the initialisation of the values at iteration 0. If the initial estimates $\mathbf{V}^\pi_0(s)$ are close to the true values, fewer iterations are required. PLoPS exploits this by initialising these values to known values of similar policies.

The stateless value for a policy π is defined as $\mathbf{V}^\pi = \sum_{s \in S} \mu(s) \mathbf{V}^\pi(s)$. A policy π *Pareto dominates* a policy π' when its value is at least as big in all objectives, and

[1] Note that [6] defines PE for single-objective MDPs, but the algorithm is identical for MOMDPs.

ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 22-24 April 2015, i6doc.com publ., ISBN 978-287587014-8. Available from http://www.i6doc.com/en/.

bigger in at least one objective: $\mathbf{V}^{\pi} \succ \mathbf{V}^{\pi'} = \forall i \left[\mathbf{V}_i^{\pi} \geq \mathbf{V}_i^{\pi'}\right] \wedge \exists j \left[\mathbf{V}_j^{\pi} > \mathbf{V}_j^{\pi'}\right]$. If π neither dominates π', nor is dominated by π', i.e., $\neg(\mathbf{V}^{\pi'} \succ \mathbf{V}^{\pi}) \wedge \neg(\mathbf{V}^{\pi} \succ \mathbf{V}^{\pi'})$, π and π' are *incomparable*. A solution to an MOMDP is a Pareto set P, a set of policies that are not *dominated* by another policy in P. The *Pareto front* P^* is defined as: $P^* = \left\{\pi \in \Pi : \neg\exists(\pi' \in \Pi)\ \mathbf{V}^{\pi'} \succ \mathbf{V}^{\pi}\right\}$, where Π is the set of all possible policies. Two sets of policies can be *merged* into a Pareto set by taking the union of the sets and removing all dominated solutions.

3.1 Pareto Local Search

Pareto local search (PLS) methods [4] are fast multi-objective optimisation algorithms that aim to find the Pareto front of undominated solutions and have been applied to a wide variety of optimisation problems. Planning in MOMDPs can be seen as a multi-objective optimisation problem, in which a solution is a policy, the fitness of a policy π is its multi-objective stateless value $\mathbf{V}^{\pi}$, and fitness evaluations are done by policy evaluation.

PLS algorithms use the Pareto dominance criterion to explore the neighbourhood of policies. We define the neighbourhood of a policy π, $\mathcal{N}(\pi)$, as the set of policies that can be made by changing one action for a single reachable state in π:

$$\mathcal{N}(\pi) = \left\{\pi' \in \Pi : \exists s \in S^{\pi}\left[\pi(s) \neq \pi'(s)\ \wedge \forall(s' \neq s)\left[\pi(s') = \pi'(s')\right]\right]\right\}, \quad (2)$$

where S^{π} is the set of states that can be reached by following policy π. PLS algorithms can be fast if it is computationally cheap to evaluate solutions that are similar to (or in the neighbourhood of) an already evaluated solution. PLS algorithms iteratively search the neighbourhood of Pareto undominated solutions for improvements in order to improve upon an intermediate approximate Pareto front. In this paper, we use *queued Pareto local search (QPLS)* [3], which is a state-of-the-art PLS algorithm, as the basis for PLoPS. QPLS can also be deployed within a genetic scheme to escape local optima [3], yielding *genetic QPLS (GQPLS)*. Note that in general any PLS algorithm can be used by performing the adjustments proposed in the next section.

4 Method

In this section, we describe how PLS algorithms, specifically GQPLS, can be adapted to be more efficient in MOMDP settings, yielding our main contribution: *Pareto local policy search (PLoPS)*.

Following the standard approach for PLS algorithms, PLoPS starts with N random policies, which are evaluated using PE. Following QPLS, these random policies and values are put in a queue Q of candidate solutions. Policies are popped off this queue one by one. Each popped policy π is then *improved*, by changing it to a policy π' from $\mathcal{N}(\pi)$ that Pareto dominates it. These improvements are performed iteratively, until no more improvements are possible. We thus need to scan $\mathcal{N}(\pi)$ for policies that Pareto dominate π. Fortunately, in MOMDPs, we can quickly check whether a neighbouring policy π' dominates π, before completely evaluating it.

ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 22-24 April 2015, i6doc.com publ., ISBN 978-287587014-8.
Available from http://www.i6doc.com/en/.

Theorem 1 *A policy $\pi' \in \mathcal{N}(\pi)$ that is different from π only in performing a' rather than a in state s_c, Pareto dominates π if and only if*

$$\sum_{s'} \mathcal{P}_{s_c s'}^{\pi'(s_c)} \left[\boldsymbol{\mathcal{R}}_{s_c s'}^{\pi'(s_c)} + \gamma \mathbf{V}^{\pi}(s') \right] \succ \mathbf{V}^{\pi}(s_c). \tag{3}$$

Proof sketch. The value of a π' can be calculated using PE, using any initialization of the values for the states. When we take $\mathbf{V}_{t=0}^{\pi'}(s) = \mathbf{V}^{\pi}(s)$ for all states, we observe that in the first iteration of PE for all states $\forall s \neq s_c \mathbf{V}_{t=0}^{\pi'}(s) = \mathbf{V}_{t=1}^{\pi'}(s)$ (due to Equation 1). For s_c, $\mathbf{V}_{t=1}^{\pi'}(s_c)$ is exactly the left-hand side of Equation 3. The stateless value after one iteration in PE can thus increase iff Equation 3 holds. In subsequent iterations, the value of s_c is propagated to states that can lead to s_c under policy π', but these can only increase in objectives i for which $V_{i,t=1}^{\pi'}(s_c)$ is an improvement over $V_{i,t=0}^{\pi'}(s_c)$, and only decrease for objectives for which $V_{i,t=1}^{\pi'}(s_c) < V_{i,t=0}^{\pi'}(s_c)$, as the transition probabilities (due to actions) for all states but s_c do not change, and the only modification can thus come via change in the values of s_c. □

After identifying a dominating policy π' from the neighbourhood of π, $\mathcal{N}(\pi)$, we initialise the values $\mathbf{V}_{t=0}^{\pi'}(s) = \mathbf{V}^{\pi}(s)$, and continue our search from π' by running PE fully to calculate $\mathbf{V}^{\pi'}$. Note that this typically requires many fewer iterations than, e.g., a uniform initialisation for $\mathbf{V}_{t=0}^{\pi'}(s) = c$, with some constant c, as the values of π and π' are typically similar.

PLoPS improves the policy π until no further improvements can be made. At that point, the policy is merged into an intermediate Pareto set P, which is initially empty. New candidates are created from π by taking k incomparable policies from $\mathcal{N}(\pi)$, which are added to the queue Q. Then, a new policy is popped off Q and improved. This procedure repeats until the queue is empty, at which point all solutions from the intermediate P are merged into the global $\overline{P}$. Finally, a new queue is filled with n (cross-)mutations sampled from $\overline{P}$ with a probability inversely proportional to the number of occurrences of its value, and the process is restarted. Contrary to standard GQPLS we limit the number of solutions, n, with which to restart QPLS, because $\bar{P}$ can grow unbounded and restarting QPLS with a large queue can be slow.

5 Experiments

In this section we give an overview of the benchmark problems used and then discuss the results.

5.1 Problem Setting

We evaluated the algorithms on two problem settings. The stochastic (in terms of reward) *Buridan's ass (BA)*, with an infinite horizon, and the episodic deterministic *deep sea treasure (DST)*.

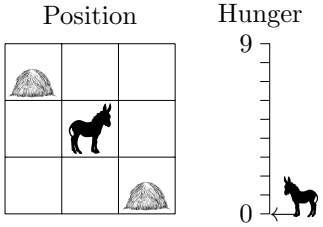

Fig. 1: BA state space

5.1.1 Buridan's Ass

BA [2], depicted in figure 1, is an infinite horizon MDP consisting of a 3×3 grid, with the agent starting in the middle. The reward vector is three-dimensional. Food is located in the two opposing corners. The agent can move into bordering

ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 22-24 April 2015, i6doc.com publ., ISBN 978-287587014-8.
Available from http://www.i6doc.com/en/.

squares, yielding a -1 reward in the first dimension. Every time step there is a 0.9 probability for each food cache to be stolen if the agent is not adjacent to or on that square. This does not make the food disappear, but gives a -1 reward in the second dimension. Hunger is a state feature which increases every time step and resets to zero when the agent is at a food square. When hunger is maximal (nine), it stops increasing and results in a -1 reward in the third dimension.

5.1.2 Deep Sea Treasure

DST [7] is an MDP that consists of a 10×11 grid, shown in figure 2. The agent always starts at the top left square, as indicated by the submarine. It can move deterministically in every direction, but only into grey and white squares. Each move results into a -1 reward in the first dimension. The maximum number of steps is 100. The grey squares are terminal states containing treasure whose worth is the reward in the second dimension.

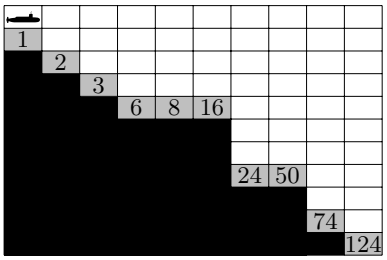

Fig. 2: DST state space

5.2 Results

PLoPS was tested on these problems, using MO-MCTS[2] and NSGA-II as baselines. NSGA-II [1] is a multi-objective evolutionary algorithm that employs elitism, uses a fast sorting algorithm, and focuses on maintaining diversity within the population. It uses the same fitness function as PLoPS. All parameters were empirically optimised, with MO-MCTS on DST as an exception, since the authors propose the optimal parameters for this problem in their paper [8].

For both problems, the best settings for NSGA-II were: a population size of 100, a mutation rate of 0.1, and no crossover. For PLoPS the parameters were: $N = 5$ for BA and $N = 50$ for DST, mutation rate 0.1, $k = 2$, and $n = 10$. For BA, MO-MCTS was configured to have a progressive widening parameter of 2 and an exploration vs. exploitation parameter of $(100, 150, 150)$. The expected value of the stochastic reward was used as a reward, in order to prevent MO-MCTS from creating an optimistic Pareto front.

For BA, γ was set to 0.99. The reference point was set to $(-200, -200, -200)$ for BA and to $(-100, 0)$ for DST. Since this MDP is deterministic, PE was replaced with a single roll-out from the starting position.

Figures 3 and 4 show the mean and standard deviation of the hypervolume measure averaged over 20 runs for BA and DST respectively. For BA, PLoPS finds a significantly better approximation than all other methods (T-test: $p < 0.001$), although NSGA-II almost immediately finds a good solution and has a smaller standard deviation. MO-MCTS is outperformed as well. For DST, it can be seen that MO-MCTS finds a bigger hypervolume on average at the start, but PLoPS consistently finds the true hypervolume after less than a minute, whereas

[2]We compared two versions of MO-MCTS. The original, which computes non-stationary policies, and a version in which it only explores stationary policies. The original was better.

ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 22-24 April 2015, i6doc.com publ., ISBN 978-287587014-8.
Available from http://www.i6doc.com/en/.

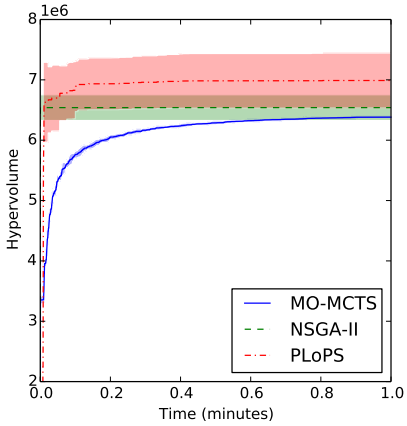

Fig. 3: Mean and standard deviation of the hypervolume over time on BA

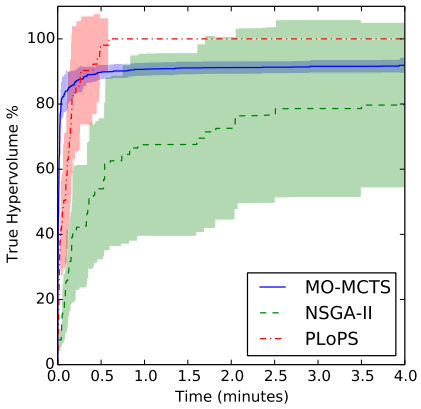

Fig. 4: Mean and standard deviation of the hypervolume over time on DST

MO-MCTS shows very little improvement after the first half minute. NSGA-II is outperformed by PLoPS on all time steps.

We also compared against CON-MODP, which was not able to converge for BA within 1 hour. For DST, it was considerably faster. However, while CON-MODP finds the entire Pareto front for MOMDPs with a deterministic transition function, it cannot handle MOMDPs with stochastic transitions [10]. Our approach (and other evolutionary methods) do not have this limitation.

6 Conclusions

This article showed that it is possible to define a neighbourhood around MOMDP policies whose values can be evaluated quickly. We exploited this property in a new algorithm called PLoPS that is based on QPLS. We also showed empirically that PLoPS performs well on benchmarks compared to the state-of-the-art.

References

[1] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *LNCS*, 1917:849–858, 2000.

[2] Z. Gábor, Z. Kalmár, and C. Szepesvári. Multi-criteria reinforcement learning. In *ICML*, volume 98, pages 197–205, 1998.

[3] M. Inja, C. Kooijman, M. de Waard, D. M. Roijers, and S. Whiteson. Queued pareto local search for multi-objective optimization. In *PPSN XIII*, pages 589–599. Springer, 2014.

[4] L. Paquete, M. Chiarandini, and T. Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for Multiobjective Optimisation*, pages 177–199. Springer, Heidelberg, 2004.

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

[6] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[7] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84(1-2):51–80, 2011.

[8] W. Wang, M. Sebag, et al. Multi-objective monte-carlo tree search. In *ACML*, 2012.

[9] M. A. Wiering and E. D. De Jong. Computing optimal stationary policies for multi-objective markov decision processes. In *ADPRL 2007*, pages 158–165, 2007.

[10] M. A. Wiering, M. Withagen, and M. M. Drugan. Model-based multi-objective reinforcement learning. In *ADPRL 2014*, pages 1–6. IEEE, 2014.