

An Empirical Comparison of Two Common Multiobjective Reinforcement Learning Algorithms

Rustam Issabekov and Peter Vamplew

School of Science, Information Technology and Engineering,
University of Ballarat, Ballarat VIC 3353, Australia
rus.issabekov@gmail.com,
p.vamplew@ballarat.edu.au

Abstract. In this paper we provide empirical data of the performance of the two most commonly used multiobjective reinforcement learning algorithms against a set of benchmarks. First, we describe a methodology that was used in this paper. Then, we carefully describe the details and properties of the proposed problems and how those properties influence the behavior of tested algorithms. We also introduce a testing framework that will significantly improve future empirical comparisons of multiobjective reinforcement learning algorithms. We hope this testing environment eventually becomes a central repository of test problems and algorithms. The empirical results clearly identify features of the test problems which impact on the performance of each algorithm, demonstrating the utility of empirical testing of algorithms on problems with known characteristics.

1 Introduction

Reinforcement learning (RL) has primarily been limited in its applicability to solving only single objective problems. However, many industrial and scientific problems are inherently complex and cannot be expressed in terms of just a single objective. This has motivated a spike in multiobjective optimization research which in turn gave birth to multiobjective reinforcement learning (MORL). MORL combines advances in multiobjective optimization and techniques from reinforcement learning, thus extending RL techniques into the realms of multiobjective problems. A number of MORL algorithms were proposed in [1, 3, 5, and 6].

However, as was outlined in [8] currently there is a need for empirical comparison of these proposed algorithms to provide detailed guidance about each algorithm's strengths and weaknesses. [8] motivated this need of guidance with the following arguments:

- Most of the proposed algorithms were tested in isolation, with only a small number of tested problems. This leads to the uncertainty about algorithm's performance under a variety of problems with different natures.
- Different testing methodologies and test problems have been used in different papers. This leads to the inability to perform direct comparison of algorithms.

This paper will provide an empirical comparison of two of the most commonly used ways to provide preferences over objectives. One way, called weighted scalarization (WS), assigns weight to each objective and forms a scalar reward based on weighted sum of individual objectives. The WS essentially turns the problem into a single objective RL task. This approach was used in [2,6]. Another way, called threshold lexicographic ordering (TLO), assigns thresholds to objectives, i.e. minimum acceptable value reward. This approach was used in [3,4].

2 Testing Framework

The introduction section showed why we are motivated to provide much needed guidance based on empirical analysis. This section will briefly explain why it was difficult to provide such guidance before.

One of the features of the reinforcement learning process is the dynamic nature of the test problem – it is a multistep decision making process, where the next state and reward depend on the currently applied action. This dynamic nature implies an interaction of the test problem and an action provider (learning algorithm). So rather than being able to use common test datasets as in other research fields such as supervised learning, RL researchers must rely on implementations of dynamic test environments. Issues then arise in terms of ensuring consistency between different implementations of the same test, and ensuring compatibility between the interfaces used for test and the varying implementations of learning algorithms created by different researchers.

In single objective reinforcement learning these issues were tackled in [7], with their software called RL-Glue. RL-Glue consists of standard application programming interfaces for problems, learning algorithms, experiments and server software that connects all them together. RL-Glue lays a solid foundation for empirical analysis in reinforcement learning. RL-Glue is very good and very useful software but is restricted to single objective reinforcement learning. In this paper we present a modified version of RL-Glue (which we call MORL-Glue) that was extended to cope with multiple number of objectives. First application programming interfaces were changed, and then server software and Java codec were altered to reflect changes in those programming interfaces.

To support the evaluation methodology proposed in [8] we also integrated hypervolume calculation code into the framework. Currently the leading algorithm for calculation of hypervolume is the one provided by the Walking Fish Group from the University of Western Australia [9]. Walking Fish Group distributes their algorithm as program written in C language. As MORL-Glue is implemented in the Java programming language, as part of its development we have ported their algorithm to Java. The MORL-Glue server's source code is available through SVN repository at <http://subversion.assembla.com/svn/mo-rl-glue/>. All the benchmarks and the algorithms that were used in this paper along with compiled Java code are available through SVN repository at <https://subversion.assembla.com/svn/rl-glue-plugins/>. Use phrase “mo-rl-glue” without quotes as a username and password, if required.

3 Methodology

Because we deal with multiple numbers of objectives we need to define the notion of dominant policy. We use the notion of Pareto dominance, which allows us to compare a pair of policies when a number of objectives is greater than one. According to the Pareto dominance, a policy A strongly dominates a policy B if it is superior on all objectives. If the policy A is equal to the policy B on at least one objective and superior on all other objectives then we say that the policy A weakly dominates the policy B. We say that the policy A is not comparable to the policy B if they dominate each other on at least one objective.

In this paper the hypervolume indicator [11] is used to measure the quality of a Pareto front approximation produced by a learning algorithm. Given a frontal set and a reference point (which is dominated by all members of the frontal set) we can calculate the hypervolume of the objective space region which is formed by the frontal set and the reference point, as shown in Figure 1. Each black dot on the figure represents a single non-dominated solution and all black dots represent an approximation to the Pareto front. Reference point is represented as a red dot. The hypervolume is the area of the shaded region, bounded by the black dots and the red dot. Improvement in any of the main characteristics of a Pareto front (accuracy, extent, diversity) translates into a higher value of the hypervolume. The meaningful interpretation of an experiment's results requires a consistency of a reference point between those experiments.

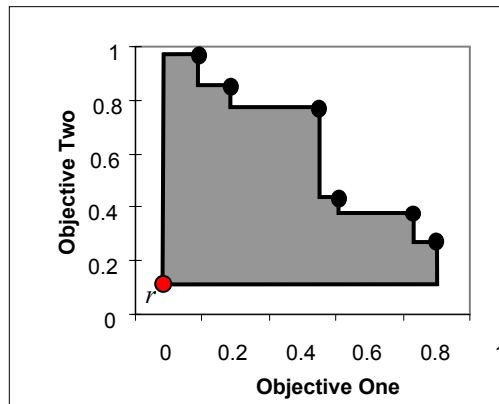


Fig. 1. An example hypervolume: The shaded area, bounded by the prevailing front (black dots) and the reference point r (red dot), represents the region from which the hypervolume is derived

Our goal is to measure the hypervolume of the Pareto front produced by given algorithms: Weighted Scalarization (WS) and Threshold Lexicographic ordering (TLO), on a variety of problems. Both of these algorithms, given a weight for each objective or a threshold for the first $n-1$ objectives, will converge to a single point in an objective space. Thus any given run of the algorithm will produce only one member of the Pareto front. To overcome this limitation the following approach was used.

3.1 Weighted Scalarization

It is well known that using the WS algorithm we can point the algorithm to a specific part of the objective space by favoring one objective over another. Thus we can target points in all regions of the objective space by gradually changing weights on all objectives. This method leads to a set of policies found using different weight combinations - each of them is only a member of the Pareto front, but all of them together approximate the Pareto front.

To summarize, first we prepare a set of weight combinations that will target points all over objective space, then we run WS algorithm on each weight combination in that set and observe a point to which algorithm converged. Eventually we will have a set of optimal points, which will be a suitable approximation of the Pareto front. The following pseudo code describes an action selection mechanism for the WS algorithm.

```
bestAction = argmax_a getWeightedSum(state, a)
```

We assume that information about number of objectives, weights assigned to these objectives and value function for each objective is provided in `getWeightedSum()` function. For each action we calculate a weighted sum and we pick an action with maximum value of the weighted sum. In this paper we use Q-learning version of the WS algorithm across all tested problems.

3.2 Threshold Lexicographic Ordering

TLO, due to its nature, requires some knowledge about the problem under consideration. Namely, thresholds should be provided for the first $n-1$ objectives, which clearly require some prior knowledge of the range of values expected for each objective. One way to obtain the required knowledge is to observe the results produced by the WS algorithm. The WS produced approximation to the Pareto front and by observing the actual points from that approximated front we can identify the region of objective space in which all points fell. Thus we can find min and max value for first $n-1$ objectives. Then we can use that information to tell TLO to look in the specific objective space region. Effectively we bound first $n-1$ objectives and leave the last objective unbound to see whether TLO can produce better results than WS. The following pseudo code sketches an action selection mechanism for the TLO algorithm.

```
chooseGreedyAction(state) {
    actions = { all available actions };
    bestActions = { };
    for(int i = 1; i <= numOfObjectives ; i++) {
        if( i < numOfObjectives ) {
            bestActions=getAboveThreshold(state,i,actions);
            if(bestActions is empty) {
                bestActions = getMaximum(state,actions,i);
                break;
            } else {
```

```

        actions = bestActions;
    }
    } else {
        bestActions = getMaximum(state, actions, i);
    }
}
return bestActions;
}

```

We assume that an array of thresholds is provided and is accessible in the `getAboveOrEqualThreshold()` function. Also we assume that the first $n-1$ objectives are to be thresholded and the last objective is to be maximized. As we run through the first $n-1$ objectives we leave only the actions that are in compliance with provided thresholds and when we reach our final objective we just pick a greedy action. In this paper we use Q-learning version of the TLO algorithm across all tested problems.

3.3 Hypervolume Sampling

To better understand the behavior of a learning algorithm we measure the hypervolume not only after convergence but also periodically during the exploration phase. To make that hypervolume sample we turn off exploration and make one run through the environment picking only greedy actions in every state, after that we turn back exploration and proceed normally.

This intermediate sampling allows us to understand a lifetime performance of a learning algorithm and compare it with other algorithms. This can be important if we are interested in good performance from the start of the problem.

4 Benchmarks

As was mentioned in first two chapters of this paper MORL-Glue facilitates the creation of a central repository of test problems. This central repository should have a wide variety of test problems each of which subjects a learning algorithm to a particular property which is found in real life problems. All these properties should be well documented to facilitate exchange of the test problems.

An example of such property is the type of rewards being used in a test problem. There are two types of rewards: extrinsic and intrinsic [10]. Intrinsic rewards are non-zero most of the time (like time penalty applied every time step), extrinsic rewards are non-zero only at special moments (like reaching a goal state). This is an important property as the algorithms may behave differently depending on the type of rewards. Another important property is the shape of the Pareto front. Research in multiobjective optimization has shown that frontal properties such as the presence of concave regions, discontinuities or uneven distribution of frontal points can significantly affect the performance of some algorithms.

In this paper we will examine three benchmarks that were presented in [8]. Further details of these benchmarks are available at <http://uob-community.ballarat.edu.au/~pvamplew/MORL.html>

4.1 Deep Sea Treasure

Deep Sea Treasure is a two-dimensional environment. The environment represents an undersea world with multiple numbers of treasure locations with varying values. The agent controls a submarine, four actions are available: left, right, up, down. The agent faces two objectives: maximizing found treasure value and minimizing time penalty.

Deep Sea Treasure is an interesting problem because of the fact that one of the objectives, treasure, is extrinsic and the second one, time, is intrinsic. In addition the Pareto front for this problem is globally concave. This structure of rewards is really suitable for showcasing advantages of TLO algorithm over WS.

Table 1 shows the results of WS and TLO runs on the Deep Sea Treasure. As was expected the WS algorithm was not able to find concave Pareto front members. Contrary to the WS, the TLO algorithm was able to locate concave Pareto front members. So the TLO was able to find extreme members of the Pareto front as well as intermediate, this leads to higher hypervolume values.

Table 1. TLO and WS algorithms tested on Deap Sea Treasure. Reference point is (0,-20). Epsilon is 0.17, alpha is 0.9, gamma is 1.0. Step limit is 200 per episode.

	10k	20k	30k	40k	50k
WS	388	292	462	288	142
TLO	460	510	510	510	510

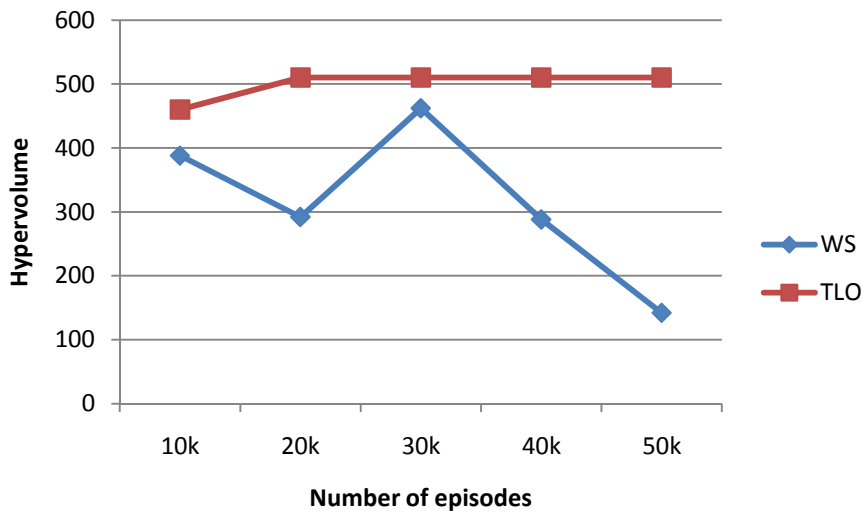


Fig. 2. WS vs TLO tested on Deep Sea Treasure. Reference point is (0;-20). Epsilon is 0.17, alpha is 0.9, gamma is 1.0.

Figure 2 summarizes the difference in performance between the TLO and the WS. The WS line on the graph is erratic due to the fact that between 10k and 40k episodes the WS algorithm was able to temporarily find concave members of the Pareto front but these members are merely a milestone and the algorithm has not yet converged. As the state-action values converge to their true values, the weighted sum approach to action selection is unable to produce any policies other than those corresponding to the extremes of the front.

4.2 MO-PuddleWorld

MO-PuddleWorld is a two-dimensional environment with puddles located at different places. The agent starts at a random location and must reach a goal state at the top-right corner. A time penalty is applied at every state except the goal state and another penalty is applied when the agent steps into the puddles.

The reward structure for MO-PuddleWorld has one intrinsic reward, namely the time penalty, which is -1 on all steps except goal state, when penalty is 0. The second reward, namely puddle penalty, is extrinsic. The MO-PuddleWorld test problem represents a state as a combination of two continuous variables: x position and y position. A 20 by 20 discretization was used in case of both the TLO and the WS algorithms. Tables 2 and 3 provide details of WS and TLO algorithms ran from 5 different starting positions. Fixed positions were used to remove the effects of noise due to random starting positions.

The results of the two algorithms are very similar. This can be explained by the nature of the problem itself. In the deep sea treasure problem the shape of the Pareto front provides a number of concave solutions and the TLO algorithm can converge to any point in that front, which increases the hypervolume. Contrary, the MO-PuddleWorld’s Pareto front from most of the starting points is primarily convex in shape, and when concave solutions are available, a number of those solutions is very small and they contribute only slightly to the overall hypervolume attainable. So the TLO algorithm never receives a chance to showcase its benefits over the WS algorithm.

Table 2. WS algorithm tested on MO-PuddleWorld problem with 5 different starting positions. Reference point is (-100,-100). Epsilon is 0.15, alpha is 0.9, gamma is 1.0. Step limit is 100 steps per episode.

	(0.25;0.6)	(0.35;0.55)	(0.3;0.55)	(0.3;0.7)	(0.2;0.7)
0	1084	988	99	99	99
500	7998	8084	7984	8287	8087
1000	7998	8084	7998	8297	8098

Table 3. TLO algorithm tested on MO-PuddleWorld problem with 5 different starting positions. Reference point is (-100,-100). Epsilon is 0.15, alpha is 0.9, gamma is 1.0. Step limit is 100 steps per episode.

	(0.25;0.6)	(0.35;0.55)	(0.3;0.55)	(0.3;0.7)	(0.2;0.7)
0	99	100	99	99	99
500	7984	8083	7984	8287	8087
1000	7984	8083	7984	8287	8087

4.3 MO-MountainCar

Mountain car is a well known reinforcement learning benchmark. Original version of the benchmark is single objective. To create the MO-MountainCar test the reward structure was modified from one objective to three.

The reward structure for mountain car problem consists of 3 intrinsic rewards. The first is a penalty of -1 applied each time step, the second is a penalty of -1 applied at every backward acceleration and the third one is a penalty of -1 applied at every forward acceleration. The MO-MountainCar test problem represents a state as a combination of two continuous variables: a position and a velocity. A 170(position) by 140(velocity) discretization was used in case of both the TLO and the WS algorithms.

Table 4. WS algorithm results on MO-MountainCar problem over 5 runs. Starting position is always fixed and is -0.5. Reference point is (-300;-300;300). Epsilon is 0.0, alpha is 0.9, gamma is 1.0.

	5k	15k	25k	35k	40k
1	6,065,247	11,819,348	15,175,815	15,322,349	15,322,597
2	5,357,800	10,926,832	15,137,425	15,417,314	15,416,766
3	5,785,239	11,629,691	14,865,468	15,302,728	15,303,667
4	6,786,636	10,713,965	15,068,736	15,442,287	15,443,135
5	7,870,923	10,963,501	14,941,997	15,270,617	15,326,463
AVG	6,373,169	11,210,667	15,037,888	15,351,059	15,362,525

Tables 4 and 5 show results of the WS and the TLO algorithms. As you can see the WS algorithm outperformed the TLO algorithm. This can be explained by the intrinsic rewards. The linear combination of objectives used to perform action-selection in the WS algorithm is compatible with both intrinsic and extrinsic rewards. However TLO's non-linear action selection mechanism performs poorly when thresholding is applied to intrinsic rewards, as it fails to account for the rewards already received earlier in the episode. Thus TLO was heavily impacted by the intrinsic rewards, which

resulted in poor figures of hypervolume. This observation is compatible with the preliminary results reported in [8], who noted TLO performs poorly on the Deep Sea Treasure task if the ordering of the objectives is changed such that the intrinsic reward is being thresholded.

Table 5. TLO algorithm results on MO-MountainCar problem over 5 runs. Starting position is always fixed and is -0.5. Reference point is (-300;-300;-300). Epsilon is 0.0, alpha is 0.9, gamma is 1.0.

	5k	25k	40k	55k	65k
1	85,349	9,315,462	11,798,739	11,925,448	11,942,691
2	85,591	9,970,806	12,590,952	12,883,731	12,879,446
3	85,528	8,483,264	12,282,405	12,386,070	12,465,344
4	85,460	10,951,818	12,525,832	12,549,660	12,549,715
5	87,309	9,206,131	12,509,929	12,617,007	12,623,177
AVG	85,847	9,585,496	12,341,571	12,472,383	12,492,074

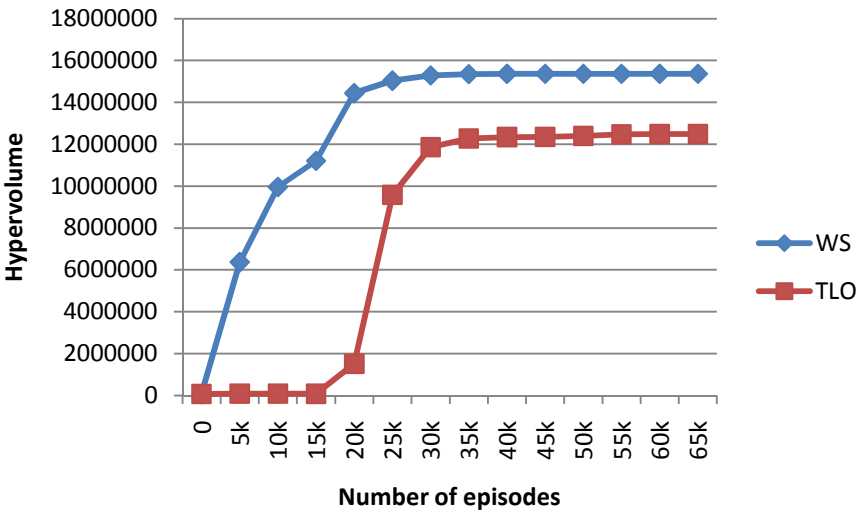


Fig. 3. WS vs TLO hypervolume growth on MO-MountainCar problem. Starting position is always fixed and is -0.5. Reference point is (-300;-300;-300). Epsilon is 0.0, alpha is 0.9, gamma is 1.0.

5 Conclusion and Future Work

Three different problems were presented. Deep Sea Treasure and MO-PuddleWorld has similar reward structure. In that one of the objectives is extrinsic and the other one is intrinsic. Meanwhile the MO-MountainCar has all of its objectives being intrinsic.

Deep Sea Treasure has a number of concavities in Pareto front. The TLO algorithm clearly benefited from its ability to locate those concavities. The WS algorithm doesn't have this ability and this leads to a situation where the WS algorithm was dominated by the TLO algorithm. The MO-PuddleWorld's reward structure is similar to the one of Deep Sea Treasure but this doesn't lead to similar dominance. This can be explained by the nature of the MO-PuddleWorld problem. The MO-PuddleWorld's Pareto front from any starting position has concavities, but the number of those concavities is not comparable to Deep Sea Treasure and for some starting positions there are no concavities at all. This lead to a situation where the TLO algorithm was not able to benefit from its main strength and showed similar results as the WS algorithm. The MO-MountainCar benchmark highlighted the dominance of the WS algorithm over the TLO in problems with intrinsic rewards.

In summary this paper demonstrates the importance of empirical studies on benchmark problems with known characteristics in establishing the conditions under which different MORL algorithms will work effectively. Clearly the TLO algorithm can only be used reliably on problems with no more than one intrinsic reward. However where the reward structure of an environment does meet this criteria, TLO is likely to outperform WS due to its capacity to discover policies which lie in concave regions of the Pareto front which cannot be found by the WS algorithm. In the future we will expand upon these results by extending the number and range of benchmarks to include other important characteristics such as larger numbers of objectives, partial-observability and non-episodic environments. We will also extend the number of MORL algorithms included in the comparison, to provide the first comprehensive comparison of a wide assortment of MORL methods.

References

1. Barrett, L., Narayanan, S.: Learning all optimal policies with multiple criteria. In: Proceedings of the International Conference on Machine Learning (2008)
2. Castelletti, A., Corani, G., Rizzolli, A., Soncinie-Sessa, R., Weber, E.: Reinforcement learning in the operational management of a water system. In: IFAC Workshop on Modeling and Control in Environmental Issues, pp. 325–330. Keio University, Yokohama (2002)
3. Gabor, Z., Kalmar, Z., Szepesvari, C.: Multi-criteria reinforcement learning. In: The Proceedings of the 15th International Conference on Machine Learning, pp. 197–205 (1998)
4. Geibel, P.: Reinforcement Learning for MDPs with Constraints. In: Fürnkranz, J., Schaffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 646–653. Springer, Heidelberg (2006)
5. Mannor, S., Shimkin, N.: The steering approach for multi-criteria reinforcement learning. In: Neural Information Processing Systems, Vancouver, Canada, pp. 1563–1570 (2001)

6. Natarajan, S., Tadepalli, P.: Dynamic preferences in multi-criteria reinforcement learning. In: The Proceedings of the International Conference on Machine Learning, Bonn, Germany, pp. 601–608 (2005)
7. Tanner, B., White, A.: RL-Glue: Language-Independent Software for Reinforcement-Learning Experiments. *Journal of Machine Learning Research* 10, 2133–2136 (2009)
8. Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., Dekker, E.: Empirical Evaluation Methods for Multiobjective Reinforcement Learning Algorithms. *Machine Learning, Special Issue on Empirical Evaluation of Reinforcement Learning* 84(1-2), 51–80 (2011)
9. While, L., Bradstreet, L., Barone, L.: A Fast Way of Calculating Exact Hypervolumes. *IEEE Transactions on Evolutionary Computation* (2010)
10. Uchibe, E., Doya, K.: Finding intrinsic rewards by embodied evolution and constrained reinforcement learning. *Neural Networks* 21(10), 1447–1455 (2008)
11. Zitzler, E.: Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland (November 1999)