Model-Based Multi-Objective Reinforcement Learning

Maikel Withagen, s1867733, m.l.c.withagen@student.rug.nl,
Marco Wiering[1] , Mădălina M Drugan[2]

***Abstract*—This thesis describes a novel multi-objective reinforcement learning algorithm. The proposed algorithm first learns a model of the multi-objective sequential decision making problem, after which this learned model is used by a multi-objective dynamic programming method to compute Pareto optimal policies. The advantage of this model-based multi-objective reinforcement learning method is that once an accurate model has been estimated from the experiences of an agent in some environment, the dynamic programming method will compute all Pareto optimal policies. Therefore it is important that the agent explores the environment in an intelligent way by using a good exploration strategy. In this paper we have supplied the agent with two different exploration strategies and compare their effectiveness in estimating accurate models within a reasonable amount of time. The experimental results show that our method with the best exploration strategy is able to quickly learn all Pareto optimal policies for the Deep Sea Treasure problem.**

I. Introduction

In Reinforcement Learning (RL) [10], an agent is placed in an environment where actions can be made, and those actions are rewarded with an indicator of how good the chosen action in that particular state was. The agent usually has little to no knowledge of the environment and has to learn to choose its actions based on this reward signal. The main goal is to learn a policy which states the best action to take in any possible state and thus reach the highest sum of rewards possible.

This setting has proven to be a very interesting field of research and many approaches to solve this problem have been proposed. In this paper we will use value-based reinforcement learning, which assigns the possible states and state-action pairs a certain value. This value is based not only on the direct reward, but also on possible future rewards. Optimal action selection is then ensured by choosing the action with the highest value.

Where the standard case of reinforcement learning only uses a single objective, the research considering multi-objective reinforcement learning has increased significantly the last decade [9], [12]. As the name suggests, in multi-objective problems there are multiple objectives to be maximized or minimized.

For example, where a cleaning robot only had to focus on cleaning as much as possible in a single-objective case, in the multi-objective case it als has to ensure a minimal battery level, minimize its traveled distance and so on.

An optimal policy is then characterized by the amount of importance each objective is given. This results in a multi-dimensional policy space where each of the axes represents the total obtained reward for an objective, and where all the total obtained rewards by the optimal policies lie on a plane called the Pareto front. Similar to [14], our multi-objective reinforcement learning algorithm considers the Pareto dominance relation to order the policies. The Pareto dominance relation [21] states that two policies are incomparable if one policy is better in one objective and worse in another objective than the second policy. A policy dominates another policy if it is better (or equal) in all objectives, and a policy is dominated by another policy if it is worse (or equal) in all objectives. The set of Pareto optimal policies is the set of policies for which there is no policy that dominates it. This set consists of the policies that lie on the previously mentioned Pareto front.

Value-function RL algorithms can be divided into model-free and model-based algorithms. In model-free algorithms, actions are selected and values are calculated according to live interactions with the environment. In model-based algorithms, the decisions are based on a model that the agent has or creates of the environment. An advantage of model-free algorithms is that they can be applied to any Markov Decision Process (MDP), where model-based algorithms need a good model of the environment or the ability to create one. An advantage of model-based algorithms is that they can process experiences of the agent more effectively, and thus get faster convergence to optimal policies.

[1] M.Wiering is with the Institute of Artificial Intelligence, University of Groningen, The Netherlands
[2] M. Drugan is with the Artificial Intelligence Lab, Vrije Universiteit Brussel, Belgium

Related Work. Most state-of-the-art multi-objective reinforcement learning algorithms are model-free value-based reinforcement learning algorithms [9], [12]. These algorithms often use scalarization functions to combine the different reward or value functions for each objective into a single scalarized value. After this step standard reinforcement learning algorithms are used.
Since a single scalarization function will produce a single unified objective, standard reinforcement learning algorithms will converge to a single Pareto optimal policy. In order to find the complete set of Pareto optimal policies, multiple instances of the scalarization function (with different parameters) or multiple scalarization functions are used.
Scalarized multi-objective reinforcement learning (MORL) algorithms have difficulties in identifying the complete set of Pareto optimal policies for non-convex Pareto fronts [7], [12], but they are efficient for convex Pareto fronts [6], [9].

There are also multi-objective multi-armed bandits algorithms, which are considered reinforcement learning algorithms with a single state used to study the theoretical properties of reinforcement learning algorithms. These methods have also used both the Pareto dominance relation [2] and the scalarization functions [3] to identify the Pareto front. Some multi-objective reinforcement learning algorithms use the lexicographical order relation [5] that assumes that one objective is more important than another objective. Finally, the hypervolume unary indicator has been used [13], [15] which is another function to transform a set of reward vectors into a single reward value.
Novel Contributions. Current MORL methods do not make use of model-based methods that first estimate the model of the environment and then solve this model. Model-based RL methods for a single objective have been researched quite extensively [8], [19] and have been shown to be able to make effective use of the agent's experiences. The transition from standard model-based RL methods to model-based MORL methods involves different dynamic programming-like methods that can solve the problem.
In this thesis we describe a novel model-based reinforcement learning algorithms for solving multi-objective reinforcement learning problems. Our algorithm combines an exploration strategy with the value iteration based multi-objective dynamic programming algorithm CON-MODP [20].
Because in MORL the amount of optimal policies may become very large, far more policies have to be learned at the same time compared with standard RL. This will cause the computational time required for solving a problem to also grow large. In order to keep computational time reasonable, we have considered deterministic multi-objective sequential decision making problems in this thesis.
The performance of CON-MODP in computing Pareto optimal policies depends on the quality of the model. For CON-MODP to find all Pareto optimal policies, all state-action pairs have to be visited at least once because of the deterministic character of the problem. We have therefore used two different exploration strategies which we will compare on their ability to effectively learn a problem. That is, we will compare the number of Pareto optimal policies CON-MODP can calculate by the produced models.
The first exploration strategy is completely random, each action is selected randomly from the possible actions in its current state. We also developed a least-visited exploration strategy, that will keep track of the number of times it has visited each state-action pair. Action selection is then done by choosing the least visited state-action pair out of the possible actions in that state. This ensures a faster exploration of the model, and should produce earlier convergence on the complete Pareto front.
Outline. In Section II we describe reinforcement learning. Section III explains model-based MORL methods, including our new method. In Section IV experimental results are presented for the Deep Sea Treasure problem. Finally, Section V concludes with our main findings and some possible future work directions.

II. Reinforcement Learning

Reinforcement learning algorithms are very useful to let an agent learn from its interaction with an environment. Typically, the RL agent is in some particular state, uses its policy to select an action, and after executing this action, the agent makes a transition to a next state and receives a scalar reward signal. Most often the RL agent is learning without any a-priori information about the environment, and therefore it has to learn from trial-and-error which policy can be used to obtain the highest future cumulative reward intake. In this section we will explain model-based RL methods that estimate a model of the environment while interacting with it and use a dynamic programming-like method to compute the best policy for its current estimated model. We will also briefly explain a model-free model-free RL method called Q-Learning [16].

A. Markov Decision Processes

In reinforcement learning the environment is typically modeled as a Markov Decision Process (MDP). A finite MDP consists of the following information:

- A set of environmental states S, where $s_t \in S$ is the state of the environment at time-step t.
- A set of actions A, where $a_t \in A$ is the action executed by the agent at time-step t.
- A transition function $T(s'|s,a)$ denoting the probability that the agent finds itself in each possible next state s' after executing action a in state s.
- A reward function $R(s,a)$ denoting the expected immediate reward obtained by executing action a in state s.
- A discount factor γ, with $0 \leq \gamma \leq 1$, which gives more importance to immediate rewards compared to rewards obtained in the future.

The goal is to calculate a policy $\pi(\cdot)$ mapping states to actions that will maximize the expected sum of discounted rewards J^π defined by:

$$J^\pi \equiv E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))]$$

B. Dynamic Programming

Each MDP has one or more optimal policies, with optimal being a maximized expected sum of (discounted) rewards. To compute an optimal policy, dynamic programming (DP) methods use value functions to compute the expected utility of a certain action given a certain state. The value of a state $V^\pi(s)$ denotes the expected cumulative discounted future reward when the agent starts in state s and follows policy π:

$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))|s_0 = s, \pi]$$

Another useful function is the Q-function, which stores the expected value of a state-action pair. $Q^\pi(s,a)$ denotes the expected cumulative discounted future reward when the agent starts in state s, takes action a, and then follows policy π:

$$Q^\pi(s,a) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))|s_0 = s, a_0, = a, \pi]$$

If the optimal Q-function Q^* is known, the agent can select optimal actions by always taking the action with the largest Q-value in each state. It can be easily shown that the optimal state value equals the highest state-action value in each state.

Bellman [1] has shown that the optimal state-action value of a state relates to the optimal state values of the states that can be reached in a single step:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a) V^*(s'),$$

where $V^*(s') = \max_{a'} Q^*(s', a')$. A number of efficient DP techniques have been developed that solve this set of non-linear equations. Value iteration for example uses the Bellman equation iteratively on all state-action pairs in order to compute an optimal Q-function:

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} T(s'|s,a) V(s'),$$

where $V(s') = \max_{a'} Q(s', a')$. After a finite amount of such updates for all states and actions, the optimal Q-function Q^* is computed, resulting in an optimal policy for the MDP.

C. Model-based Reinforcement Learning

One of the biggest disadvantages of dynamic programming is that the transition and reward functions should be known a-priori. This means that a human has to design the MDP for a particular problem, which can be time-consuming, erroneous, or infeasible for particular problems.

Model-based RL methods learn to estimate the model from the experiences of the agent interacting with the environment. Although model-learning can be hard for particular environments with continuous or huge state spaces, for moderately sized problems it can be done efficiently. Furthermore, some comparisons have shown model-based RL to be much more effective than model-free methods such as Q-learning [19]. This is why we want to use model-based RL for solving multi-objective MDPs which are usually difficult to solve due to the existence of many Pareto optimal policies.

Model-based RL methods learn the transition and reward models of the environment by making use of counters that are used in a maximum-likelihood way to compute approximate transition probabilities and average rewards. Each time the agent selects action a in state s and makes a transition to state s', the transition model's counter values $C(s,a)$ and $C(s,a,s')$ are increased by one. In a similar fashion, the obtained reward r is added to the value $R^T(s,a)$ which computes the sum of all rewards obtained by selecting action a in state s. Finally, the maximum likelihood model of the MDP is computed as:

$$T'(s'|s,a) = \frac{C(s,a,s')}{C(s,a)} \quad \text{and} \quad R'(s,a) = \frac{R^T(s,a)}{C(s,a)}$$

At each moment in time, the estimated model can be used by a dynamic programming method to compute a new policy. Some algorithms such as prioritized sweeping have been proposed [8] which use a smart strategy to re-compute the Q-function with much less computational effort.

D. Q-Learning

Another approach to avoid the a-priori knowledge is by using a model-free technique. An example of such a technique is Q-learning, which has proven to converge [17] to an optimal policy for every finite MDP.

Q-Learning uses value functions to compute the expected utility of a certain action given a certain state, similar to DP. But where DP uses a-priori knowledge of the environment's transitions and rewards, Q-Learning simply updates its value functions by taking a certain action in a certain state and observing the transition and the reward signal. This 'online' approach eliminates the need for a model. The learning rule itself is a simple update:

$$Q(s,a) \stackrel{+}{=} \alpha[R(s,a) + \gamma \max_a Q(s',a) - Q(s,a)]$$

where α is the learning rate.

III. Model-based MORL

In multi-objective Markov decision processes (MOMDPs), instead of the usual scalar reward function $R(s,a)$, a reward vector $\vec{R}(s,a)$ is used. The vector $\vec{R}(s,a)$ consists of l dimensions or components representing the different objectives. This means that a reward function $\vec{R}(s,a) = (R_1(s,a), \ldots R_l(s,a))$ is given in an MOMDP that returns the expected reward vector for each state-action pair.

The reward vector in MOMDPs has multiple components, and therefore conflicts can arise between them. For example, it may be possible that an agent is able to compute a policy for a very nice sight-seeing tour, but that the cost of this tour is larger than the cost of other possible (less nice) tours. Therefore, the algorithms have to deal with several trade-offs and compute all policies that are not dominated by another policy. This set of policies is called the *Pareto efficient (optimal) set*. When given the set of Pareto optimal policies, we can let a user or autonomous agent select one of them given their preferences at that time.

A relatively simple method to solve MODPs, is the use of a scalarized multi-objective algorithms [7] which basically transform the multi-dimensional solution space to a single dimension, after which single-objective MDP algorithms kan be used.

To solve MOMDPs we can also use particular multi-objective dynamic programming (MODP) algorithms that compute the Pareto front of optimal policies and corresponding value functions. These MODP methods are based on value iteration [18] or policy iteration [4], [11] and extend conventional dynamic programming algorithms by computing sets of policies and value functions.

A. Scalarized Q-Learning

In Scalarized Q-Learning, the multiple Q-values of a state-action pair, one for each objective, are combined into a single scalarized Q-value. This scalarized Q-value can then be used to perform standard Q-Learning, as mentioned in Section II.

A variation of Q-learning that uses scalarization is Linear Scalarized Q-Learning as described in [7]. In this approach each objective is given its own 'weight' value. The scalarized Q-value is than calculated by multiplying each objective's weight value with its Q-value and summing the total. By varying with the weight values, different Pareto optimal values can be found.

B. Multi-objective Dynamic Programming

The solution of most multi-objective DP methods is to keep track of all non-dominated (or Pareto efficient) value functions and policies. The multi-objective value function has different cumulative reward components or values and this is denoted by a value function $V^i(s) = (V_1^i(s), V_2^i(s), \ldots, V_l^i(s))$, where V_x^i denotes the discounted cumulative reward intake of reward component x of policy i.

The dominance function $\succ$ works on two value vectors for a state s as follows:

$$V^i(s) \succ V^j(s) \Leftrightarrow \exists x; V_x^i(s) > V_x^j(s) \wedge \neg \exists y; V_y^i(s) < V_y^j(s)$$

So a value vector of policy i dominates a value vector of policy j if policy i has a higher value on some component x and does not have a lower value on any other component.

We will use V^O to denote the set of value functions (and policies) that are not dominated:

$$V^O(s) = \{V^i(s) | V^i(s) \text{ is not dominated by a policy in } s\}$$

Furthermore, V^D is used for denoting the set of value functions computed at some given moment that may include dominated ones.

The same is done for the Q-function, thus the algorithm keeps track of a set Q^O that denotes the set of non-dominated Q-functions. Here the Q-functions should not be dominated by another Q-vector for the same state-action pair. Thus:

$$Q^O(s,a) = \{Q^i(s,a) | Q^i(s,a) \text{ is not dominated in } s,a\}$$

The non-dominated operator ND tells whether a policy i is not dominated in the state s by any value function vector of the set of value functions $V^D(s)$:

$$ND(V^i(s), V^D(s)) \Leftrightarrow \neg \exists V^j(s) \in V^D(s) \,;\, V^j(s) \succ V^i(s)$$

Analogous definitions hold for Q-vector functions.

C. CON-MODP

In this thesis we will make use of the CON-MODP algorithm [20] that focuses on only computing stationary deterministic Pareto optimal policies. This makes this method more effective than previous dynamic programming methods for solving MOMDPs, although the drawback is that CON-MODP assumes deterministic problems.

With the previous formulations it is now possible to enhance dynamic programming to compute non-dominated value function sets. For simplicity we restrict ourselves to deterministic MDPs. We define the Pareto optimal operator PO as:

$$\begin{aligned} PO(Q^D(s,a)) &= \{Q^i(s,a) | Q^i(s,a) \in Q^D(s,a) \wedge \\ &\quad ND(Q^i(s,a), Q^D(s,a))\} \end{aligned}$$

Furthermore, we construct the dynamic programming operator as follows where $\oplus$ denotes an addition operator working on sets (and vectors) and s' is the next state:

$$DP(Q^D(s,a)) = (\vec{R}(s,a) \oplus \gamma V^O(s') | P(s,a,s') = 1.0)$$

Here $V^O(s)$ is computed as:

$$V^O(s) = PO(\cup_a Q^D(s,a))$$

Note that this dynamic programming operator is defined for deterministic environments (therefore $P(s,a,s') = 1$ for some s'). For stochastic environments, this operator should be changed, but this could lead to a huge increase in the number of Pareto optimal policies.

It is known that in infinite horizon discounted Markov decision processes, there is always a single optimal value function and one or more stationary deterministic policies belonging to it. The CON-MODP algorithm exploits this fact and uses a consistency operator that eliminates most non-stationary policies and reevaluates non-stationary (or inconsistent) policies that are only inconsistent in a single state that is being expanded. Dynamic programming methods usually expand each state and this could sometimes lead to temporally inconsistent policies where different actions are selected in the same state at different time-steps.

CON-MODP detects when a policy is made inconsistent due to the last lookahead update step, and then changes it to a consistent policy by forcing the last action in the current state that is evaluated all the times this state will be visited. In this way, the policy is consistent again.

Because the sequence of actions from which the value function has been computed is changed, policy evaluation is used by CON-MODP to compute the true value of the consistent policy. The CON-MODP algorithm uses the operators CON, PO, and DP. The CON operator may make the policy π consistent (in case it was not) and recomputes its Q-value vector as:

$$\begin{aligned} CON(Q^\pi(s,a)) &= \pi \text{ with } Q^\pi(s,a), \;\; \text{if} \;\; \pi(s) = a \\ &= \pi' \text{ with } Eval(\pi'), \text{ if } \pi(s) \neq a \text{ is the only} \\ &\quad \text{inconsistency, and } ND(Q^\pi(s,a), Q^O(s,a)) \\ &\quad \text{where } \pi'(s) = a \text{ and } \pi'(s') = \pi(s') \; \forall s' \neq s \\ &= \emptyset, \;\; \text{otherwise} \end{aligned}$$

Here $Eval(\pi')$ means that policy π' is evaluated using policy evaluation for the same number of steps as the original policy π is computed. Evaluation is only done if the original inconsistent policy was not already dominated. If the previous policy was dominated, the policy can be discarded anyway, since its values will never be larger than those of the previous policy.

We also let CON work immediately on sets of Q-functions and policies. The CON-MODP algorithm is now defined as:

$$Q^{O*} = (PO(CON(DP(Q_0))))^*$$

Where $X(\cdot)^*$ means that operator X is repeated until convergence. This algorithm is able to compute all stationary Pareto optimal policies and corresponding value functions for deterministic finite MOMDPs.

D. Exploration Strategies

We have developed two different exploration policies to obtain experiences by interacting with the environment from which the model is learned. The simplest exploration strategy is the Random-exploration approach, where actions are taken randomly and the findings are used to update the model. This will finally converge to a perfect environmental model, but faster methods exist. The other exploration strategy we used, is a Least-Visited approach, which chooses the actions to take according to the times it has explored a certain state-action combination.

1) Least-Visited exploration: As stated before, Least-Visited exploration chooses the actions to take according to the times it has taken those actions before. More specifically, it keeps track of the actions in each state and counts the times it has explored them. This gives a value for each possible action in each possible state. Action selection is then done by choosing the action with the lowest counter. After an action is taken, its corresponding state-action counter is incremented. If there are multiple actions with the lowest state-action counter value, the last equal action is chosen.

2) Random-exploration: In Random-exploration, actions are chosen completely randomly. The agent starts at its starting state and takes randomly chosen actions until it lands in a final state, while observing and updating rewards and transition probabilities in its model.

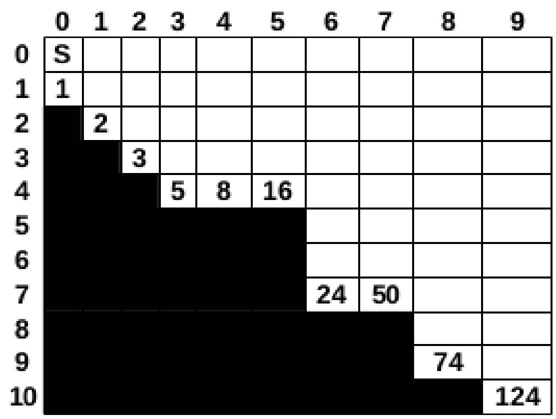

Fig. 1. *A visual representation of the Deep Sea Treasure world. The starting state is shown as S and the numbers in the cells represent the goal state values.*

IV. Experimental Results

In order to test our approach, we have used a multi-objective reinforcement learning problem known as the "Deep Sea Treasure" [12]. The Deep Sea Treasure environment states an episodic task where an agent has to explore the sea bottom for treasures. The world consists of a 10×11 grid with 10 goal states. The goal state value is increased as the location is further away from the starting state. There are four determinstic actions possible; Go Up, Go Down, Go Left, Go Right. The two objectives are to minimize the number of steps taken before reaching a goal state, and maximizing the goal reward. The agent receives a reward of -1 for every step taken (the first objective). If a goal state is reached, the goal reward is the value of the treasure (the second objective). The fastest path to each goal state is part of the Pareto optimal set, so the Pareto front exists of 10 policies. The distribution of the goal state values causes an entirely concave Pareto front. Figure 1 shows an illustration of the problem.

Set-up We will compare our model-based MORL method with the two exploration strategies to Scalarized Q-Learning [7], as mentioned in section III. Because this approach works with linear scalarization functions, it can only find Pareto optimal policies in a convex Pareto front. Therefore, in the Deep Sea Treasure world, it can only find the (-1,1) and the (-19.124) solutions. We have supplied this method with scalarization weights of (1, 0) and (0, 1) for the two objectives. These weights will ensure convergence on the two possible solutions in a good model.

Our results are collected and averaged over 10 trials of each 2000 iterations. For the Scalarized Q-learning algorithm, the learning rate $\alpha = 0.1$, $\gamma = 0.9$ and $\epsilon = 0.1$. For our model-based MORL methods we set $\gamma = 0.9$ as well. The maximum amount of steps to find a solution was 1000.

A. Results

We first present the cardinality results, which are the number of Pareto optimal policies found after a specified number of iterations. For the three methods we use a greedy exploration strategy to compute the Pareto optimal policies after each 10 iterations. As mentioned before, the maximal number of Pareto optimal policies is 10 for the Deep Sea Treasure problem. Table I shows results after 200, 500, 1000, and 2000 iterations. The table shows that our model-based method with the random exploration strategy as well as our least-visited exploration strategy significantly outperforms the scalarized Q-learning algorithm. Furthermore, the table also shows that our model-based MORL method with the least-visited exploration strategy finds all Pareto optimal solutions within 2000 iterations in every run.

TABLE I
Cardinality results on the Deep Sea Treasure

Iterations	Q-learning	Random	Least-visited
200	1.1 ± 0.3	6.6 ± 0.8	7.8 ± 0.8
500	1.2 ± 0.4	8.3 ± 0.8	9.4 ± 0.5
1000	1.8 ± 0.4	9.1 ± 0.7	9.4 ± 0.5
2000	2.0 ± 0.0	9.6 ± 0.5	10.0 ± 0.0

In Figure 2 we also show the entire learning performance of the three methods. The model-based MORL methods improve their results much faster and on final convergence will have learned all ten solutions. The random exploration strategy however needed more then 2000 iterations in some experiments.

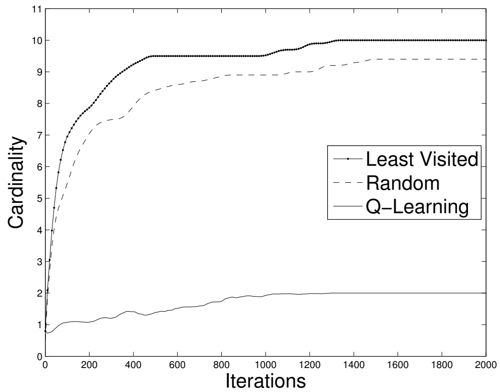

Fig. 2. *Learning performance on the Deep Sea Treasure problem measured in the number of found Pareto optimal policies.*

We also show the results of the three methods using the hypervolume assessment metric. The hypervolume gives the volume between the summed rewards obtained by the Pareto optimal policies and a reference point (-25,0). Table II shows that the hypervolume is maximized after 2000 iterations by the model-based MORL method with the least-visited exploration strategy. It then obtains the maximal value 1155. The model-based MORL method with the random exploration strategy also shows good results, but will not converge as fast as our model-based MORL method with the least-visited exploration strategy and needs more than the given 2000 iterations in a number of experiments. It is also interesting to see that the hypervolume of the least-visited method after 200 iterations is already larger than the one of the scalarized Q-learning algorithm.

TABLE II
Hypervolume results on the Deep Sea Treasure

Iterations	Q-learning	Random	Least-visited
200	98 $\pm$ 2331	686 $\pm$ 264	852 $\pm$ 164
500	172 $\pm$ 311	890 $\pm$ 211	1101 $\pm$ 90
1000	614 $\pm$ 311	971 $\pm$ 183	1101 $\pm$ 90
2000	762 $\pm$ 0	1055 $\pm$ 141	1155 $\pm$ 0

We show the learning performance as measured by the hypervolume assessment function in Figure 3.

We finally test for significant differences on the mean hypervolume performances with a Wilcoxon signed rank test, shown in table III. If we keep the significance threshold on a p-value of 0.05, both our model-based MORL method with the least-visited as well as the random exploration perform significantly better than the scalarized Q-learning algorithm. Between our two methods, no significant difference between the mean hypervolumes exists.

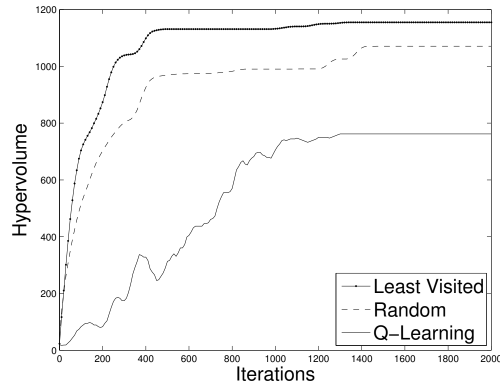

Fig. 3. *Learning performance on the Deep Sea Treasure problem measured using the hypervolume unary indicator.*

TABLE III
Wilcoxon signed rank test of the mean hypervolumes between the approaches

Method 1	**Method 2**	**p-value**
Q-Learning 762.00	*Random* 1055	$4.14e^{-5}$
Q-Learning 762.00	*Least-Visited* 1155	$1.59e^{-5}$
Random 1055.00	*Least-Visited* 1155	$7.76e^{-2}$

B. Discussion

The two policies that the scalarization Q-learning algorithm finds are the two ends of the Pareto front. This confirms the results obtained in [7]. The model-based MORL methods will both converge to learning all Pareto optimal policies. If every state-action pair is visited at least one time, the CON-MODP algorithm is able to compute all Pareto optimal solutions.

The results show that the model-based MORL method with least-visited exploration performs better in terms of speed than the model-based MORL method with the random exploration strategy. This corresponds with the idea that giving priority to less visited state-action pairs leads to a quicker exploration of the complete environment. Fast exploration was useful in this particular setting since only one visit is enough for CON-MODP, but in the case of stochastic environment one visit is not enough to completely learn the environment's transition function. Other exploration methods might then be more preferable.

Some other methods in MORL literature have been used for the Deep Sea Treasure problem, but none of them was as effective as our method. Finally, we want to note that a single call of CON-MODP cost no more than one second.

V. Conclusion and Future Work

In this thesis we have described a novel approach to solve a deterministic multi-objective reinforcement learning problem by using a combination of an exploration strategy and CON-MODP. Our method is able to learn a model of the environment through interaction and to compute the complete Pareto optimal set of policies.

Results on the Deep Sea Treasure have shown that the method is near-optimal for this problem. We are able to compute the complete Pareto front in a shorter learning period compared to other existing approaches.

The deterministic setting was chosen to be able to run test experimental set-ups while keeping the computational time required managable. In future work we intend to handle stochastic environments. This will cause an increase in the computational time required, but with our current experiences and possible exploitation of problem structures faster algorithms can be created.

References

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] M. Drugan and A. Nowe. Designing multi-objective multi-armed bandits: an analysis. In *Proc of International Joint Conference of Neural Networks (IJCNN)*, 2013.

[3] M. M. Drugan and A. Nowe. Scalarization based pareto optimal set of arms identification algorithms. In *Proc of International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2014.

[4] N. Furukawa. Vector valued Markovian decision processes within countable state space. In R. Hartley, L. Thomas, and D. White, editors, *Recent Developments in Markov Decision Processes*, pages 205–223. Academic Press, New York, 1980.

[5] Z. Gabor, Z. Kalmar, and C. Szepesvari. Multi-criteria reinforcement learning. In M. K. P. Inc., editor, *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[6] D. J. Lizotte, M. Bowling, and S. A. Murphy. Efficient reinforcement learning with multiple reward functions for randomized clinical trial analysis. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML)*, 2010.

[7] K. V. Moffaert, M. M. Drugan, and A. Nowé. Scalarized multi-objective reinforcement learning: Novel design techniques. In *ADPRL*, pages 191–199, 2013.

[8] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[9] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Intell. Res. (JAIR)*, 48:67–113, 2013.

[10] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT press, Cambridge MA, A Bradford Book, 1998.

[11] L. Thomas. Constrained Markov decision processes as multi-objective problems. In S. French, L. Thomas, R. Hartley, and D. White, editors, *Multi-Objective Decision Making*, pages 77–94. Academic Press, 1983.

[12] 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, 2010.

[13] K. van Moffaert, M. M. Drugan, and A. Nowe. Hypervolume-based multi-objective reinforcement learning. In *Proc of Evolutionary Multi-objective Optimization (EMO)*. Springer, 2013.

[14] K. van Moffaert, M. M. Drugan, and A. Nowe. Learning sets of pareto optimal policies. In *Thirteenth International Conference on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop (ALA)*, 2014.

[15] W. Wang and M. Sebag. Hypervolume indicator and dominance reward based multi-objective monte-carlo tree search. *Machine Learning*, 92(2-3):403–429, 2013.

[16] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

[17] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[18] D. White. Multi-objective infinite-horizon discounted Markov decision processes. *Journal of Mathematical Analysis and Applications*, 89:639–647, 1982.

[19] M. A. Wiering. *Explorations in Efficient Reinforcement Learning*. PhD thesis, University of Amsterdam, February 1999.

[20] M. A. Wiering and E. D. de Jong. Computing optimal stationary policies for multi-objective markov decision processes. In *Proc of Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 158–165. IEEE, 2007.

[21] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE T. on Evol. Comput.*, 7:117–132, 2003.