Learning Optimal Policies in MDPs with Value Function Discovery*

Paper #35, 3 pages

ABSTRACT

In this paper we describe recent progress in our work on Value Function Discovery (VFD), a novel method for discovery of value functions for Markov Decision Processes (MDPs). In a previous paper we described how VFD discovers algebraic descriptions of value functions (and the corresponding policies) using ideas from the Evolutionary Algorithm field. A special feature of VFD is that the descriptions include the model parameters of the MDP. We extend that work and show how additional information about the structure of the MDP can be included in VFD. This alternative use of VFD still yields near-optimal policies, and is much faster. Besides increased performance and improved run times, this approach illustrates that VFD is not restricted to learning value functions and can be applied more generally.

Categories and Subject Descriptors

G.3 [**Probability and statistics**]: Markov processes; I.2.8 [**Problem Solving, Control Methods, and Search**]: Dynamic programming

General Terms

Algorithms, Performance

Keywords

Markov Decision Processes, Evolutionary Algorithms, Value Function, Genetic Programming.

1. INTRODUCTION

In this paper we extend our work [2] on VFD, a novel method that discovers value functions for Markov Decision Processes (MDPs). VFD is based on Genetic Programming (a type of Evolutionary Algorithm), and has as its key feature that the discovered descriptions of value functions are algebraic in nature. In particular, these description include the model parameters of the MDP.

We give a short description of VFD and introduce the MDP from [2], which we reuse as an example. Contrary to [2], in this paper we do not use VFD to discover a near-optimal value function. Instead, we use the fact that, for our example MDP, the optimal policy is a threshold policy, and we let VFD learn a description of this threshold in terms of the model parameters. The learned threshold corresponds to a policy, which we then compare to the optimal policy.

Numerical experiments show that this alternative use of VFD yields an expression for the threshold that captures the threshold of the optimal policy well. The run time of VFD in this paper is, however, several orders of magnitude shorter compared to the application of VFD in [2]. Additionally, the work in the current paper shows that VFD is not restricted to learning value functions and can be applied more generally.

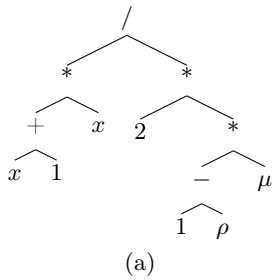
*A full version of this paper is available at [2]. Authors are anonymized for double-blind review.

IFIP WG 7.3 Performance 2015, Sydney, Australia

2. GENETIC PROGRAMMING

VFD is based on Genetic Programming (GP) and the standard application of GP is discovering algebraic descriptions of functions from samples of this function at various points. To be precise, suppose a function $V(x)$ is unknown, but that we do have samples $V(s_i)$ at various points s_i. Applying GP allows discovery of an approximate algebraic expression $\widetilde{V}(x)$ for $V(x)$ such that $V(s_i) \approx \widetilde{V}(s_i)$ for all sample points $(s_i, V(s_i))$.

GP uses trees to represent a function, and several of these trees together form the population. Fig. 1a illustrates a tree representation of the function $V(x) = \frac{x(x+1)}{2\mu(1-\rho)}$. The operators $(/, *, +, -)$ from this expression are in the internal nodes of the tree, whereas the leafs contain the variables (x), parameters (ρ, μ), and constants $(1, 2)$. Here we only use the operators $(/, *, +, -)$ shown in the example, but the representation is flexible and also allows for, e.g., exponents, square roots, logarithms, and rounding.



(a)

Figure 1: Function $V(x) = \frac{x(x+1)}{2\mu(1-\rho)}$ as a tree

By mutating single trees and combining multiple trees, GP

Algorithm 1 Value function discovery (VFD)

```
1: function VFD( )
2:     samplePointSets ← readSamplePointSets()
3:     population ← initPopulation()
4:     bestTrees ← List()
5:     while not isConverged() do
6:         repeat
7:             if apply mutation then
8:                 children ← mutate(selectParent());
9:             else
10:                children ←recombine(selectParent(),
11:                                    selectParent());
12:            end if
13:        until MU children generated
14:        setError(children)
15:        population ← population + children
16:        sort(population)
17:        survivorSelection()
18:    end while
19:    sort(bestTrees)
20:    return bestTrees[0]
21: end function
```

creates new functions based on the trees in the population. Following the paradigm of an Evolutionary Algorithm, mutating and combining 'good' trees will, over time, result in even 'better' trees. This process is repeated until a function is discovered that fits sufficiently well.

3. VALUE FUNCTION DISCOVERY

VFD applies GP to sample points of the optimal value function of an MDP, provided by, for instance, value iteration. By including sample points obtained with different values for the parameters of the MDP, these parameters can also be included in the discovered expression.

A pseudo code listing of VFD is shown in Algorithm 1. The algorithm starts at line 2 by loading the sets of sample points of the MDP. Next, the population is initialized by filling it with LAMBDA randomly generated trees. Lines 5–18 describe the steps taken by GP: first, MU children are generated using mutation and recombination (lines 7–12). Then, their error is calculated, they are added to the population, and the population is sorted from smallest error to largest (lines 14-16). Survivor selection removes LAMBDA trees from the population, leaving MU individuals (line 17). This procedure is repeated until convergence (line 5). Upon convergence VFD returns the best tree found during the search (line 20). A detailed description of the various functions can be found in [2].

4. EXAMPLE MDP

We use the same MDP as in [2], and shortly repeat it here. Fig. 2 shows a queue with Poisson arrivals (rate λ) and two servers with exponential service rates μ_1 and μ_2 ($\mu_1 > \mu_2$). The jobs in the queue have to be assigned non-preemptively to either the fast server (S_1) or the slower server (S_2), assuming one is available. This decision is taken after a job completion, as well as after a job arrival.

We model this scenario as an MDP, with state $(x, i) \in \mathcal{X} = \mathbb{N} \times \{0, 1\}$. Here, x denotes the number of jobs in the queue and S_1, and i the number of jobs in S_2. Our aim is to minimize the average number of jobs in the system. From [1] we have the optimality equation

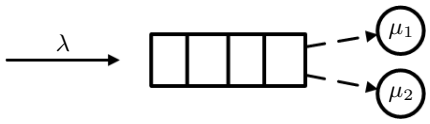


Figure 2: An M/M/2 system with control, where jobs (arriving with rate λ) from the queue have to be assigned to either a fast server S_1 (with service rate μ_1) or to a slow server S_2 (with service rate μ_2)

$$g + V(x,i) = (x+i) + \lambda W(x+1,i) + \mu_1 W((x-1)^+, i) + \mu_2 W(x,0), \quad (1)$$

with

$$\begin{aligned} W(x,0) &= \min\{V(x,0); V(x-1,1)\} \quad \text{if } x > 0, \\ W(0,i) &= V(0,i), \\ W(x,1) &= V(x,1). \end{aligned} \quad (2)$$

The function $W(x,i)$ reflects the decision to be taken after the occurrence of an event. In particular, if S_2 is empty the decision is between leaving the job in the queue ($V(x,0)$) or moving one job from the queue to S_2 ($V(x-1,1)$), as shown in Eq. (2). If the queue and S_1 are empty then moving a job is not possible and the state of the system does not change ($W(0,i) = V(0,i)$). Also, if the second server is busy the state does not change ($W(x,1) = V(x,1)$). In Eq. (1), the second, third, and fourth term on the right-hand side correspond to the decision upon a job arrival, a job completion at S_1, and a job completion at S_2, respectively. Finally, the constant g is the time-average costs of the system.

5. LEARNING THRESHOLD WITH VFD

In [2] we apply VFD to sample points of the optimal value function, obtained via value iteration. In this paper, we use prior knowledge that the optimal policy of Eq. (1) is a threshold policy (see [1] for a proof). To be precise, for given model parameters λ, μ_1, μ_2 the optimal policy states that server S_2 should be used whenever $x > T$, i.e., when x exceeds a threshold T. By viewing this threshold as a function of the model parameters, we can apply VFD and discover an algebraic expression for T in terms of the model parameters (λ, μ_1, μ_2).

Using this alternative approach, each sample point set contains just one point: the threshold T. In [2], each sample point set would contain several points (x, i) on the value function $V(x,i)$. Note that by making this change we only affect the input to VFD, and not the algorithm itself.

6. NUMERICAL RESULTS

We reuse the parameters settings for VFD that we listed in [2, Table 1]. The sample point sets, which form the input to VFD, are slightly different. First, we focus on systems with a high load, since systems with a low load have a large threshold which is unlikely to be reached. Therefore, we keep only sample point sets $3-6$, corresponding to

Table 1: Model parameters per sample point set

Set	ρ_1	λ	μ_1	μ_2
0	0.650	0.375	0.578	0.047
1	0.775	0.429	0.554	0.017
2	0.900	0.464	0.515	0.021
3	0.950	0.459	0.483	0.058

Table 2: Costs $\tilde{g}$ and threshold $\widetilde{T}$ of the threshold policy discovered by VFD, compared to costs g and threshold T of the optimal policy

Set	ρ_1	T	$\widetilde{T}$	g	$\tilde{g}$
0	0.650	5	4.254	1.771	1.790
1	0.775	10	9.195	3.338	3.340
2	0.900	6	7.044	6.914	6.923
3	0.950	3	2.874	6.094	6.133

Table 3: Costs $\tilde{g}$ and threshold $\widetilde{T}$ of the policy discovered by VFD, compared to costs g and threshold T of the optimal policy. The model parameters are different from the ones VFD was given as input

ρ_1	λ	μ_1	μ_2	T	$\widetilde{T}$	g	$\tilde{g}$
0.600	0.364	0.606	0.030	9	6.207	1.490	1.510
0.700	0.389	0.556	0.055	4	3.643	2.083	2.117
0.825	0.443	0.537	0.021	7	7.451	4.271	4.271
0.875	0.433	0.494	0.073	3	2.559	3.679	3.699
0.925	0.473	0.511	0.016	7	9.064	9.310	9.347

loads $\rho_1 = 0.650, \rho_1 = 0.775, \rho_1 = 0.900$, and $\rho_1 = 0.950$. We renumber them as $0-3$. Second, we change the model parameters λ, μ_1, μ_2 of each sample point set so that the resulting thresholds of the optimal policy differ substantially (in [2] the thresholds ranged from 3 to 5 and these are too close together for the current experiment). The resulting model parameters are shown in Table 1.

For each of these 4 parameter combinations we determine the threshold T of the optimal policy using value iteration, and together these thresholds form the input to VFD. Running VFD yields an algebraic description of the threshold in terms of the model parameters:

$$\widetilde{T}(\lambda,\mu_1,\mu_2) = \mu_1(\mu_1+\mu_2)\Big(1+\mu_1(\frac{1}{\lambda}+\lambda+2(1-\mu_2)) + \frac{\lambda}{\mu_2}-(1+\lambda)\mu_2+\mu_1^2\mu_2\Big).$$

This expression implies a policy for which we determine the time-average costs (denoted by $\tilde{g}$) with policy evaluation. In Table 2 we list the costs g and threshold T of the optimal policy, and the costs $\tilde{g}$ and threshold $\widetilde{T}$ from VFD. The table shows that our alternative application of VFD also yields near-optimal policies, and that the discovered thresholds closely resemble the optimal thresholds. Note that, because we set MIN_ERROR to 0.2, the discovered thresholds deviate at most 20% from the optimal values.

Next, we inspect the performance of the thresholds discovered by VFD on model parameters that it was not trained on. Therefore, we select five values for ρ_1, different from those in Table 2, and randomly generate new parameters λ, μ_1, μ_2. The resulting values are shown in Table 3 in the first four columns. Then, we calculate the thresholds $\widetilde{T}$ using the function discovered by VFD, and evaluate the corresponding policy to find $\tilde{g}$. Finally, we run value iteration for each combination of model parameters to get the optimal T and g. Table 3 shows that the function discovered by VFD yields thresholds that closely resemble the optimal thresholds, as well as near-optimal policies.

Finally, we repeat the investigation of the run time of VFD from [2, Sec. V.D]. There, we recorded a mean run time of 2 minutes and 21 seconds over 25 runs. In the current setup, running VFD 25 times gives a median run time of 95.6 milliseconds, several orders of magnitude faster than in [2]. This improvement in run time is because the sample point sets now contain just one point and are thus much smaller than before.

7. CONCLUSIONS AND FUTURE WORK

In this paper we continued our work on VFD, a novel algorithm for discovering value functions for Markov Decision Processes. We applied VFD to the same example MDP as in [2], but included extra knowledge that the optimal policy of the MDP is of threshold type. Instead of discovering the value function with VFD, we learned an algebraic expression for the threshold in terms of the model parameters. We numerically inspected the resulting threshold and the corresponding policy to the optimal policy. The results show that his alternative use of VFD still yields near-optimal policies and thresholds that closely resemble the optimal value.

The work presented in this paper is currently in progress, and we intend to include additional features in the near future. Firstly, we can include the fact that thresholds are integers, whereas VFD currently yields decimal values. We expect that this increases the speed of VFD on this problem. Secondly, we plan to find more accurate thresholds by implementing what we called the 'exploitation phase' in [2]. This phase essentially gives VFD more time to improve a learned threshold function.

8. REFERENCES

[1] G. Koole. A simple proof of the optimality of a threshold policy in a two-server queueing system. *Systems & Control Letters*, 26(5):301–303, 1995.

[2] Author 1, Author 2, and Author 3. Value Function Discovery in MDPs with Evolutionary Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics Systems*, 2014. [under review].