Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online
CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 50-1 (2017) 4162–4167

Optimal Control via Reinforcement Learning with Symbolic Policy Approximation

Jiří Kubalík [*] **Eduard Alibekov** [*] **Robert Babuška** [**,*]

[*] *Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Prague, Czech Republic,* {*eduard.alibekov, jiri.kubalik*}*@cvut.cz*
[**] *Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands, r.babuska@tudelft.nl*

Abstract: Model-based reinforcement learning (RL) algorithms can be used to derive optimal control laws for nonlinear dynamic systems. With continuous-valued state and input variables, RL algorithms have to rely on function approximators to represent the value function and policy mappings. This paper addresses the problem of finding a smooth policy based on the value function represented by means of a basis-function approximator. We first show that policies derived directly from the value function or represented explicitly by the same type of approximator lead to inferior control performance, manifested by non-smooth control signals and steady-state errors. We then propose a novel method to construct a smooth policy represented by an analytic equation, obtained by means of symbolic regression. The proposed method is illustrated on a reference-tracking problem of a 1-DOF robot arm operating under the influence of gravity. The results show that the analytic control law performs at least equally well as the original numerically approximated policy, while it leads to much smoother control signals. In addition, the analytic function is readable (as opposed to black-box approximators) and can be used in further analysis and synthesis of the closed loop.

Keywords: reinforcement learning, value iteration, symbolic regression, genetic programming, nonlinear model-based control, optimal control

1. INTRODUCTION

Reinforcement learning (RL) algorithms provide a way to optimally solve dynamic decision-making and control problems. With continuous-valued state and input spaces, RL relies on function approximators to represent the value function and policy mappings. Various types of numerical approximators have been used: expansions with fixed or adaptive basis functions (Munos and Moore, 2002; Buşoniu et al., 2011), regression trees (Ernst et al., 2005), local linear regression (Atkeson et al., 1997; Grondman et al., 2012), and increasingly popular deep neural networks (Lange et al., 2012; Mnih et al., 2013, 2015; Lillicrap et al., 2015; de Bruin et al., 2016).

Function approximators are difficult to tune, so that convergent learning results. In addition, they can negatively affect the control performance, as manifested, for instance, by chattering control signals and steady-state errors. Examples of such a behavior are often found in papers, including the above references, but it is usually disregarded, as the emphasis in RL is on learning a control policy at all, typically from scratch. However, if RL is to be regarded as a viable alternative to other optimal control design methods, close attention must be paid to the actual control performance.

In this paper we present a novel method that uses symbolic regression (SR) to build an analytic representation of the control policy. Symbolic regression is based on genetic programming and it has been used in nonlinear data-driven modeling or data mining, often with quite impressive results (Schmidt and Lipson, 2009; Staelens et al., 2012; Brauer, 2012; Vladislavleva et al., 2013). To our best knowledge, there have been no reports in the literature on the use of symbolic regression for policy approximation in reinforcement learning. We argue that the effective use of symbolic regression is a valuable element missing from current RL schemes and we demonstrate its usefulness.

The specific method employed in this work is a modified version of Single Node Genetic Programming (SNGP) (Jackson, 2012a,b), which is a graph-based genetic programming method. A basic overview on this methods is given in Appendix A. For further details, please, refer to (Kubalík et al., 2016).

The paper is organized as follows. Section 2 outlines the reinforcement learning approach considered in this work. Section 3 gives an overview of standard numerical policy derivation methods and our novel symbolic policy derivation method is described in Section 4. Simulation experiments with 1-DOF robot arm are presented in Section 5, and Section 6 concludes the paper.

2. PRELIMINARIES

Process model and control goal. The system to be controlled is described by the state transition function $x_{k+1} = f(x_k, u_k)$, with $x_k, x_{k+1} \in \mathcal{X} \subset \mathbb{R}^n$ and $u_k \in \mathcal{U} \subset \mathbb{R}^m$. The control goal is defined through a *reward function* which assigns a scalar reward $r_{k+1} \in \mathbb{R}$ to the state transition from x_k to x_{k+1}:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) \\ r_{k+1} &= \rho(x_k, u_k, x_{k+1}) \end{aligned} \tag{1}$$

The reward function is defined by the user and typically calculates each individual reward based on the distance of the current state from a given constant reference state x_r that should be attained. Note that the process model does not have to be described by explicit equations; one can use a generative model, such as a numerical simulation of differential equations.

The goal of RL is find the optimal control policy $\pi : \mathcal{X} \rightarrow \mathcal{U}$, which in each state selects a control action such that the expected cumulative discounted reward over time, called the return, is maximized:

$$R^\pi = E\Big\{\sum_{k=0}^{\infty} \gamma^k \rho(x_k, \pi(x_k), x_{k+1})\Big\} \tag{2}$$

Here $\gamma \in [0, 1)$ is a discount factor and the initial state x_0 is drawn uniformly from the state space domain $\mathcal{X}$. The return is approximated by the value function $V^\pi : X \rightarrow \mathbb{R}$ defined as:

$$V^\pi(x) = E\Big\{\sum_{i=0}^{\infty} \gamma^i \rho(x_i, \pi(x_i), x_{i+1}) \Big| x_0 = x, \pi\Big\} \tag{3}$$

An approximation of the optimal V-function $\hat{V}^*(x)$ can be computed by solving the Bellman optimality equation

$$\hat{V}^*(x) = \max_{u \in \mathcal{U}} \Big[\rho\big(x, \pi(x), f(x, u)\big) + \gamma \hat{V}^*\big(f(x, u)\big)\Big] \tag{4}$$

To simplify the notation, in the sequel, we drop the hat and the star superscript: $V(x)$ will therefore denote the approximated optimal V-function. Based on $V(x)$, the corresponding optimal policy can be derived in several ways, as detailed in Section 3.

Fuzzy V-iteration. To compute $V(x)$, the fuzzy V-iteration algorithm (Buşoniu et al., 2010) is used. Given is the process model and the reward function (1). Define the set $C = \{c_1, \ldots, c_N\}$ of points distributed over a regular grid in the state space. Further define a vector of triangular membership functions (MF) $\phi = [\phi_1(x), \ldots, \phi_N(x)]^T$ so that each $\phi_i(x)$ is centered at c_i, i.e., $\phi_i(c_i) = 1$ and $\phi_j(c_i) = 0$, $\forall j \neq i$. The membership functions are normalized so that $\sum_{j=1}^{N} \phi_j(x) = 1, \forall x \in \mathcal{X}$. For a single state variable x_j these functions are defined as follows:

$$\begin{aligned} \phi_1(x_j) &= \max\left(0, \min\left(1, \frac{c'_2 - x_j}{c'_2 - c'_1}\right)\right), \\ \phi_i(x_j) &= \max\left(0, \min\left(\frac{x_j - c_{i-1}}{c_i - c_{i-1}}, \frac{c_{i+1} - x_j}{c_{i+1} - c_i}\right)\right), \\ &\quad i = 2, \ldots, N_j - 1, \\ \phi_{N_j}(x_j) &= \max\left(0, \min\left(\frac{x_j - c_{N_j-1}}{c_{N_j} - c_{N_j-1}}, 1\right)\right). \end{aligned}$$

An extension to more dimensions is realized in a straightforward way by using the Cartesian product of the membership functions in the individual dimensions. Finally, define a finite set of discrete control input values $U = \big\{u^1, u^2, \ldots, u^M\big\} \subset \mathcal{U}$. The value function is approximated by the following basis-function expansion

$$V(x) = \theta^T \phi\,(x)$$

where $\theta = [\theta_1, \ldots, \theta_N]^T \in \mathbb{R}^N$ is a parameter vector found through the following iteration:

$$\theta_i \leftarrow \max_{u \in U} \Big[\rho\big(c_i, u, f(c_i, u)\big) + \gamma \theta^T \phi\big(f(c_i, u)\big)\Big] \tag{5}$$

for $i = 1, 2, \ldots, N$. This value iteration is guaranteed to converge (Buşoniu et al., 2010) and terminates when the following condition is satisfied:

$$||\theta - \theta^-||_\infty \leq \epsilon \tag{6}$$

with θ^- the parameter vector calculated in the previous iteration and ϵ a user-defined convergence threshold. Fuzzy value iteration is very effective for second-order systems – computing the optimal value function is a matter of seconds. However, the computational and memory requirements grow exponentially and the method is not practical for systems above order four. Other methods (Ernst et al., 2005) can be used for higher-order systems.

3. NUMERICAL POLICY DERIVATION METHODS

There are two principal ways to derive the control policy from the value function. The first one is based on an online maximization of the Bellman optimality equation's right-hand side, while the second one applies the Bellman equation offline and uses basis functions to interpolate online. We term the first method the *hill-climbing policy* (H-policy) and the second one the *interpolated policy* (I-policy).

3.1 Hill-climbing policy

The optimal control action in any given state x is found as the one that maximizes the right-hand side of the Bellman optimality equation (4):

$$u = \underset{u' \in U}{\operatorname{argmax}} \Big[\rho\big(x, u', f(x, u')\big) + \gamma V\big(f(x, u')\big)\Big] \tag{7}$$

An advantage of this control law is its inherent stability – the value function is analogous to the control Lyapunov function (Lewis et al., 2012) and the above control law boils down to hill-climbing the Lyapunov function. However, two drawbacks of this method are immediately clear:

(1) The process model must be available for on-line use in the controller. If the model is computationally involved, so will be the computation of the control action.
(2) The maximization is a computationally expensive procedure. The most straightforward way is to enumerate all discrete actions in U and choose the one that maximizes the argument. This obviously leads to discrete-valued control action and the associated drawbacks. For more details and methods to alleviate these drawbacks refer to (Alibekov et al., 2016a).

Additional properties of the policy given by (7) stem from the fact that $V(x)$ is approximate and its smoothness is influenced by the choice of basis functions. This affects the hill-climbing process and may result in artifacts like chattering of the control action, whose influence on the control performance is difficult to estimate a priori.

3.2 Interpolated policy

This method is based on calculating (7) off-line for all states in set C:

$$p_i = \underset{u \in U}{\operatorname{argmax}} \left[\rho\big(c_i, u, f(c_i, u)\big) + \gamma \theta^T \phi\big(f(c_i, u)\big) \right] \qquad (8)$$

where p_i is the optimal control action in state c_i, for $i = 1, 2, \ldots, N$. These control actions are collected in vector $p = [p_1, \ldots, p_N]^T \in U^N$ and the control action in an arbitrary state x is then obtained by interpolation:

$$u = p^T \phi(x) \qquad (9)$$

where $\phi(x)$ are the same basis functions[1] as for $V(x)$. An obvious advantage of this method is its computational simplicity: most computations are done offline (vector p is actually obtained for free as a byproduct of the fuzzy value iteration algorithm) and the online interpolation is computationally cheap. Another advantage is that (9) directly produces continuous control actions. However, as we will see in Section 5, the control signal is not necessarily smooth and the interpolation can also result in a steady-state error. Therefore, in the next section, we propose a symbolic approximation method which is computationally effective and also yields a smooth control law.

4. SYMBOLIC POLICY APPROXIMATION

We build an analytic approximation of the policy by using symbolic regression. This general technique is based on genetic programming and its goal is to find an analytic equation describing some given data. Here, the specific objective is to find an analytic equation for the policy function that closely approximates data sampled from the interpolated policy. The policy must produce smooth control and also return as precise value u_r of the control signal at the reference point $\mathbf{x}_r$ as possible, rendering the reference state x_r an equilibrium of the closed-loop system:

$$x_r = f(x_r, u_r)\,.$$

Symbolic regression based on genetic programming is a suitable technique for this task, as we generally do not have any prior knowledge on the symbolic policy function sought. We use a variant of *Single Node Genetic Programming*, which is described in Appendix A.

When applying genetic programming to a particular symbolic regression problem, one has to define a set of elementary functions whose combination is sufficient to produce a precise approximation model. We use the basic arithmetic operators plus, minus, multiply and three nonlinear functions – sine, square and hyperbolic tangent. To avoid over-fitting, we impose a limit on the maximal size of the evolved symbolic expressions. If no size limits were used, the genetic programming could produce overly complex models which would be useless for our purpose. At the same time, we need the symbolic model to be very precise at the reference point $\mathbf{x}_r$ in order to attain a minimal steady-state error. To achieve this, we highly penalize (weight 10) the error produced by the model at the point $\mathbf{x}_r$.

[1] Other interpolation methods can be employed, such as cubic splines.

For a typical optimal control problem as stated in Section 2, the policy surface can be split into saturated parts where the control signal attains the minimal or maximal possible value, and a (often rather steep) transition between the two parts. The transition is generally nonlinear and its shape follows in a non-trivial way from the model and from the control goal, stated via the reward function. Thus, the final policy can be approximated with a combined approximation composed of two constant functions for the saturated parts and the nonlinear function for the transition. In this paper, the SR method is used to evolve the symbolic policy (S-policy, for short) for the transition only. The training data consist of samples of the transition itself plus samples on the boundaries between the transition and the saturated parts of the policy.

5. SIMULATED ROBOT ARM

The policy approximation methods are evaluated and compared in simulation of a 1-DOF robot arm operating under the influence of gravity. The equation of motion is:

$$J\ddot{\alpha} = Mgl\sin(\alpha) - \left(b + \frac{K^2}{R_a}\right)\dot{\alpha} + \frac{K}{R_a}u \qquad (10)$$

where α is the arm angle measured clockwise from the upright position and $u \in [-10, 10]$ V is the control voltage. The model parameters are given in Table 1.

Table 1. Robot arm parameters

Model parameter	**Symbol**	**Value**	**Units**
Arm inertia	J	$1.91 \cdot 10^{-4}$	kgm^2
Arm mass	M	$5.50 \cdot 10^{-2}$	kg
Arm length	l	$4.20 \cdot 10^{-2}$	m
Gravity acceleration	g	9.81	m/s^2
Damping	b	$3 \cdot 10^{-6}$	Nms
Torque constant	K	$5.36 \cdot 10^{-2}$	Nm/A
Armature resistance	R_a	9.50	Ω

The fully measurable state $\mathbf{x}$ consists of the angle α and the angular velocity $\dot{\alpha}$. The reference state $\mathbf{x}_r$ is given by the desired reference angle α_r and zero velocity:

$$\mathbf{x} = \begin{bmatrix} \alpha \\ \dot{\alpha} \end{bmatrix} \qquad \mathbf{x}_r = \begin{bmatrix} \alpha_r \\ 0 \end{bmatrix}$$

The reward function is defined as a quadratic function with a steeper exponential peak superimposed in the neighborhood of the reference angle:

$$\rho(x_k, u_k, x_{k+1}) = -(x_{r,1} - x_{k,1})^2 - 8(1 - e^{-10(x_{r,1} - x_{k,1})^2})$$

The parameters of the fuzzy value iteration algorithm are listed in Table 2. The number of membership functions was chosen quite large (19×19) in order to get a dense coverage of the state space domain of interest. To obtain the one-step-ahead state transitions, the model (10) is simulated by using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.01$ s. The discount factor $\gamma = 0.99$ is selected close to one, so that virtually no discounting takes place within a typical closed-loop transient which lasts for about 30 samples ($\gamma^{30} \approx 0.74$).

The fuzzy value iteration algorithm was applied to the system for a set of predefined reference states. It converges

Table 2. Value iteration parameters

Parameter	Symbol	Value	Units
State domain	$\mathcal{X}$	$[-\pi, \pi] \times [-40, 40]$	rad×rad/s
Number of MF	N	$361 = 19 \times 19$	–
Discount factor	γ	0.99	–
Convergence threshold	ϵ	0.1	–
Sampling period	T_s	0.01	s

in about 70 iterations, yielding value functions such as the one shown in Figure 1 for the reference angle $\alpha_r = -0.7$ rad.

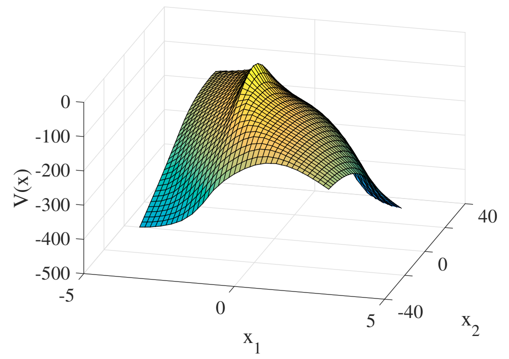

Fig. 1. The value function computed for $\alpha_r = -0.7$ rad.

Figure 2 shows a typical closed-loop response obtained with the I-policy (9) from the initial state $x_0 = [-2.5 \;\; 0]^T$. While the state trajectory is smooth and resembles a time optimal (bang-bang) response, the control input trajectory shows transient oscillations. These are caused by the interpolation artifacts on the steep part of the I-policy surface visualized in Figure 3.

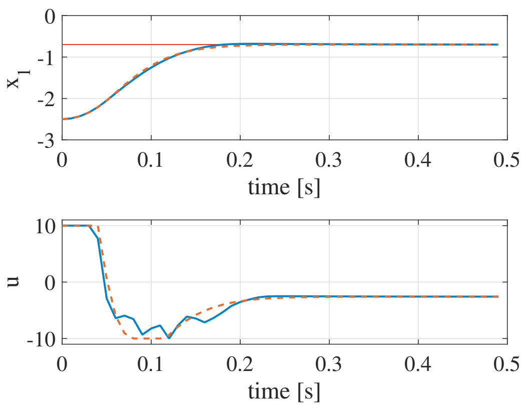

Fig. 2. I-policy: a typical time-domain response suffers from control input chattering (solid blue curve). S-policy: control input chattering is absent (dashed red curve).

Evolved symbolic policies were experimentally evaluated and compared to H-policy and I-policy. One hundred simulations were carried out with each policy for three different reference angles $\alpha_r^1 = -1.37$ rad, $\alpha_r^2 = -1.05$ rad and $\alpha_r^3 = -0.70$ rad. The simulations were started from initial states sampled over a regular grid in the state space. Each simulation resulted in a trajectory for which the return (2) and the mean absolute angle error (MAE) were

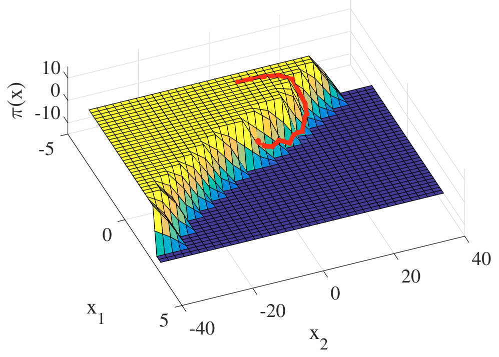

Fig. 3. The state–input trajectory of Figure 2 superimposed on the I-policy surface. The chattering is caused by the interpolation artifacts clearly visible on the steep part of the control policy surface.

calculated. The MAE is calculated as the mean absolute difference between observed angle and the reference angle over the states visited along the simulation trajectory. The mean values of R and MAE over the set of one hundred simulations were used as the performance measures to compare the three policy variants. The results are in Table 3. The H-policy performs worse, due to chattering and steady-state errors, while the I-policy and S-policy yield similar performance. The main advantage of the S-policy is the smoothness of the control action, as shown in Figure 2.

Table 3. Return and mean-absolute error obtained with H-policy, I-policy and S-policy. Numbers are rounded to two decimal digits.

	H-policy		I-policy		S-policy	
α_r	R	MAE	R	MAE	R	MAE
α_r^1	-368.24	0.27	-366.98	0.25	-367.04	0.25
α_r^2	-343.00	0.23	-341.97	0.20	-342.14	0.19
α_r^3	-320.08	0.22	-318.04	0.17	-317.92	0.17
mean	-343.77	0.24	-342.33	0.20	-342.37	0.20

Examples of well-performing S-policies evolved for the three desired reference angles are:

$$u^{(\alpha_r^1)} = \mathrm{sat}\big(a_1 x_1 + a_2 x_2 + a_3 \tanh(0.5 x_1^2) + a_4\big)$$
$$u^{(\alpha_r^2)} = \mathrm{sat}\big(a_1 x_1^2 + a_2 x_1 + a_3 \sin(\sin(x_1) + a_4) + a_5 x_2 + a_6\big)$$
$$u^{(\alpha_r^3)} = \mathrm{sat}\big(a_1 x_1 + a_2 x_2 + a_3 \sin(x_1 + a_4) + a_5\big)$$

with the function $\mathrm{sat}(\cdot)$ defined as follows:

$$\mathrm{sat}(z) = \max\left(-10, \min\left(10, z\right)\right)$$

and the coefficients shown in Table 4. These policies were selected manually from a set of results obtained by repeatedly running the SNGP algorithm (see the Appendix). It is interesting to note that all the policies involve a linear PD control law (the linear terms involving x_1 and x_2) and one or more smooth nonlinear terms.

Table 4. S-policy coefficients

α_r	Parameter vector **a**
α_r^1	$[-25.15 \quad -2.06 \quad 21.27 \quad -54.89]$
α_r^2	$[\;\; 1.82 \quad -21.83 \quad -18.25 \quad 51.60 \quad -1.82 \quad -20.19]$
α_r^3	$[-21.53 \quad -2.05 \quad -12.86 \quad 0.58 \quad -18.97]$

Figure 4 shows a plot of the S-policy evolved for reference angle α_r^3. Notice that thanks to the smoothness, no chattering occurs on the state–input trajectory which is superimposed on the S-policy surface.

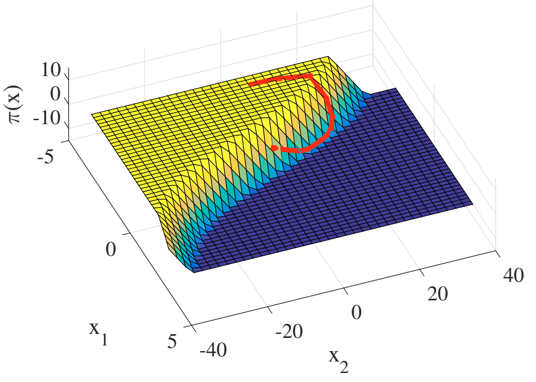

Fig. 4. The state–input trajectory of Figure 2 superimposed on the smooth S-policy surface. No chattering occurs.

6. CONCLUSION

The proposed symbolic method offers an alternative way to policy approximation. An inherent advantage of this approach is the possibility to interpret and analyze the resulting policy, which is described analytically. Simulation results on a 1-DOF robot arm show the evolved symbolic policy exhibits smooth control, contrary to the hill-climbing and interpolated policies. Moreover, the symbolic policy outperforms the hill-climbing policy and performs equally well to the interpolated policy with respect to the return and the mean absolute angle error. Our future research will focus on automatic selection of the final symbolic model based on an accuracy-complexity tradeoff and on formal analysis of closed-loop stability with the obtained policies.

7. ACKNOWLEDGMENT

This research was supported by the Grant Agency of the Czech Republic (GAČR) with the grant no. 15-22731S titled "Symbolic Regression for Reinforcement Learning in Continuous Spaces" and by the Grant Agency of the Czech Technical University in Prague, grant no. SGS16/228/OHK3/3T/13 titled "Knowledge extraction from reinforcement learners in continuous spaces". The authors thank Erik Derner for his suggestions to improve the paper.

Appendix A. GENETIC PROGRAMMING

Genetic programming (GP) belongs to methods frequently used to solve the symbolic regression problem. Besides the standard Koza's tree-based GP (Koza, 1992), many other variants have been proposed such as Grammatical Evolution (Ryan et al., 1998) which evolves programs whose syntax is defined by a user-specified grammar, Gene Expression Programming (Ferreira, 2001) that evolves linear chromosomes that express as tree structures through a genotype-phenotype mapping or graph-based Cartesian GP which represents programs in the form of a directed graph (Miller and Thomson, 2000).

A Single Node Genetic Programming (SNGP) (Jackson, 2012a,b), used in this work, is a graph-based GP method that evolves a population of individuals, each representing a single program node. The node can be either terminal, i.e. a constant or a variable in case of the symbolic regression problem, or a function chosen from a set of functions defined for the given problem. Importantly, the individuals are not entirely distinct, they are interlinked in a graph structure so some individuals act as input operands of other individuals.

Formally, an SNGP population is a set of L individuals $M = \{m_0, m_1, \ldots, m_{L-1}\}$, with each individual m_i being a single node represented by the tuple $m_i = \langle e_i, f_i, Succ_i, Pred_i, O_i \rangle$, where

- $e_i \in T \cup F$ is either an element chosen from a function set F or a terminal set T defined for the problem;
- f_i is the fitness of the node;
- $Succ_i$ is a set of successors of node i, i.e. the nodes whose output serves as the input to the node i;
- $Pred_i$ is a set of predecessors of node i, i.e. the nodes that use output of node i as their input;
- O_i is a vector of outputs produced by the node.

The population is organized so that the left-most nodes are terminals followed by function nodes. Every function node can use as its successor (i.e. the operand) only nodes that are positioned lower down in the population. This means that for each $m_j \in Succ_i$ it holds $0 \leq j < i$. Note that each function node is in fact a root of a program tree that can be constructed by recursively traversing its successors towards the leaf terminal nodes.

A single evolutionary operator called *successor mutation* (*smut*) is used to modify the population. It randomly picks one individual of the population and replaces one of its successors by a reference to another individual of the population making sure that the constraint imposed on successors is satisfied. Output values of the mutated node and all nodes higher up in the population affected by the mutation operation are recalculated.

Finally, the population evolves through a local search-like procedure. In each iteration, a new population is produced by the *smut* operator which is then accepted or rejected for the next iteration according to a chosen acceptance rule.

The SNGP implementation used in this work to solve the symbolic regression differs from the one described in (Jackson, 2012a,b) in the following aspects

- Organization of the population. We use a population with function nodes divided into head and tail partitions as introduced by Alibekov et al. (2016b). The head partition nodes can use only other head function nodes and constant nodes as its input. The tail nodes can use head nodes, tail nodes, constants and variables as their input. The head partition therefore represents a pool of constants that can be used in expressions rooted in the tail partition nodes.
- Form of the regression model. A hybrid SNGP denoted as the Single-Run SNGP with LASSO pro-

posed by Kubalík et al. (2016) is used, which evolves generalized linear regression models. The tail partition nodes represent possibly nonlinear features from which the generalized linear regression model is built using the Least Absolute Shrinkage and Selection (LASSO) regression technique (Tibshirani, 1994). In this way, a precise, linear-in-parameters nonlinear regression models can be produced.
- Fitness function. The generalized regression models are optimized with respect to the mean squared error calculated over the set of training samples.
- Selection strategy used to choose the nodes to be mutated. We use the so-called depth-wise selection introduced by Kubalík et al. (2016). This selection method is biased toward deeper nodes of well-performing expressions. The idea behind this strategy is that changes made to the nodes at deeper levels are more likely to bring an improvement than changes made to the nodes close to the root of the expression.
- Evolutionary model. The process of evolving the population is carried out in epochs. In each epoch, multiple independent parallel threads are run for a predefined number of generations, all of them starting from the same population – the best final population out of the previous epoch threads. In this way the chance of getting stuck in a local optimum is reduced.

REFERENCES

Alibekov, E., Kubalík, J., and Babuška, R. (2016a). Policy derivation methods for critic-only reinforcement learning in continuous action spaces. In *Proceedings 4th IFAC Conference on Intelligent Control and Automation Sciences (ICONS)*, 285–290. Reims, France.

Alibekov, E., Kubalík, J., and Babuška, R. (2016b). Symbolic method for deriving policy in reinforcement learning. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, 2789–2795. IEEE.

Atkeson, C.G., Moore, A.W., and Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review*, 11(1-5), 11–73.

Brauer, C. (2012). Using Eureqa in a Stock Day-Trading Application. Cypress Point Technologies, LLC.

Buşoniu, L., Ernst, D., Babuška, R., and De Schutter, B. (2010). Approximate dynamic programming with a fuzzy parameterization. *Automatica*, 46(5), 804–814.

Buşoniu, L., Ernst, D., De Schutter, B., and Babuška, R. (2011). Cross-entropy optimization of control policies with adaptive basis functions. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 41(1), 196–209.

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. (2016). Off-policy experience retention for deep actor-critic learning. In *Deep Reinforcement Learning Workshop, Advances in Neural Information Processing Systems (NIPS)*.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 503–556.

Ferreira, C. (2001). Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13(2), 87–129.

Grondman, I., Vaandrager, M., Buşoniu, L., Babuška, R., and Schuitema, E. (2012). Efficient model learning methods for actor–critic control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 42(3), 591–602.

Jackson, D. (2012a). *A New, Node-Focused Model for Genetic Programming*, 49–60. Springer, Berlin, Heidelberg.

Jackson, D. (2012b). *Single Node Genetic Programming on Problems with Side Effects*, 327–336. Springer, Berlin, Heidelberg.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. MIT Press Ltd.

Kubalík, J., Alibekov, E., Žegklitz, J., and Babuška, R. (2016). *Hybrid Single Node Genetic Programming for Symbolic Regression*, 61–82. Springer, Berlin, Heidelberg.

Lange, S., Riedmiller, M., and Voigtlander, A. (2012). Autonomous reinforcement learning on raw visual input data in a real world application. In *Proceedings 2012 International Joint Conference on Neural Networks (IJCNN)*, 1–8. Brisbane, Australia.

Lewis, F., Vrabie, D., and Vamvoudakis, K. (2012). Reinforcement Learning and Feedback Control. *IEEE Control Systems Magazine*, 32(6), 76–105.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. ArXiv:1509.02971 [cs.LG].

Miller, J.F. and Thomson, P. (2000). *Cartesian Genetic Programming*, 121–132. Springer, Berlin, Heidelberg.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arxiv.org/abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

Munos, R. and Moore, A. (2002). Variable resolution discretization in optimal control. *Machine learning*, 49(2), 291–323.

Ryan, C., Collins, J., and Neill, M.O. (1998). *Grammatical evolution: Evolving programs for an arbitrary language*, 83–96. Springer, Berlin, Heidelberg.

Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923), 81–85.

Staelens, N., Deschrijver, D., Vladislavleva, E., Vermeulen, B., Dhaene, T., and Demeester, P. (2012). Constructing a No-Reference H. 264/AVC Bitstream-based Video Quality Metric using Genetic Programming-based Symbolic Regression. *Circuits and Systems for Video Technology, IEEE Transactions on*, 99, 1–12.

Tibshirani, R. (1994). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58, 267–288.

Vladislavleva, E., Friedrich, T., Neumann, F., and Wagner, M. (2013). Predicting the energy output of wind farms based on weather data: Important variables and their correlation. *Renewable Energy*, 50, 236–243.