Applied Soft Computing 27 (2015) 457–467

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Continuous probabilistic model building genetic network programming using reinforcement learning

Xianneng Li[a,b], Kotaro Hirasawa[a,b,*]

[a] Graduate School of Information, Production and Systems, Waseda University, Hibikino 2-7, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan
[b] Information, Production and Systems Research Center, Waseda University, Hibikino 2-7, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan

ARTICLE INFO

Article history:
Received 17 April 2013
Received in revised form
13 September 2014
Accepted 18 October 2014
Available online 6 November 2014

Keywords:
Estimation of distribution algorithm
Probabilistic model building genetic network programming
Continuous optimization
Reinforcement learning

ABSTRACT

Recently, a novel probabilistic model-building evolutionary algorithm (so called estimation of distribution algorithm, or EDA), named probabilistic model building genetic network programming (PMBGNP), has been proposed. PMBGNP uses graph structures for its individual representation, which shows higher expression ability than the classical EDAs. Hence, it extends EDAs to solve a range of problems, such as data mining and agent control. This paper is dedicated to propose a continuous version of PMBGNP for continuous optimization in agent control problems. Different from the other continuous EDAs, the proposed algorithm evolves the continuous variables by reinforcement learning (RL). We compare the performance with several state-of-the-art algorithms on a real mobile robot control problem. The results show that the proposed algorithm outperforms the others with statistically significant differences.

1. Introduction

Despite the selection operator based on the concept of "Survival-of-the-fittest", classical Evolutionary Algorithms (EAs) generally evolve the population of candidate solutions by the random variation derived from biological evolution, such as crossover and mutation. However, numerous studies report that the results of EAs strongly rely on the configurations of the parameters associated with the stochastic genetic operators, such as crossover/mutation rate. For concrete problems, the parameter settings generally vary. Hence, the parameter tuning itself becomes an optimization problem. Meantime, the stochastic genetic operators sometimes may not identify and recombine the building blocks (BBs, defined by high-quality partial solutions) correctly and efficiently due to the implicit adaptation of the *building block hypothesis* (BBH) [1,2], which causes the problems of premature convergence and poor evolution ability. These reasons have motivated the proposal of a new class of EAs named estimation of distribution algorithm (EDA) [3], which has received much attention in recent years [4,5]. As the name implies, EDA focuses on estimating the probability distribution of the population using statistic/machine learning to construct a probabilistic model. Despite the selection operator which is also used to select the set of promising samples for the estimation of probability distribution, EDA replaces the crossover and mutation operators by sampling the model to generate new population. By explicitly identifying and recombining the BBs using probabilistic modeling, EDA has drawn its success to outperform the conventional EAs with fixed, problem-independent genetic operators in various optimization problems.

Numerous EDAs has been proposed, where there are mainly three ways to classify the existing EDAs. (1) From the model complexity viewpoint, EDAs can be mainly classified into three groups [6]: univariate model, pairwise model and multivariate model, which identify the BBs of different orders. Univariate model assumes there is no interactions between the elements,[1] hence constructing the probabilistic model by marginal probabilities to identify BBs of order one. Similarly, pairwise and multivariate models use more complex methods to model BBs of order two and more. One can easily observe that estimating the distribution is not an easy task and modeling more accurate model generally requires higher computational cost [4,7]. (2) From the perspective of individual structures, EDA can mainly be classified into two groups,

* Corresponding author at: Graduate School of Information, Production and Systems, Waseda University, Hibikino 2-7, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan. Tel.: +81 93 692 5261; fax: +81 93 692 5261.
E-mail addresses: sennou@asagi.waseda.jp (X. Li), hirasawa@waseda.jp (K. Hirasawa).

[1] The elements refer to the variables/alleles in genetic algorithm (GA), or nodes in genetic programming (GP).

http://dx.doi.org/10.1016/j.asoc.2014.10.023
1568-4946/© 2014 Elsevier B.V. All rights reserved.

which are probabilistic model building genetic algorithm (PMBGA) [8] and PMB genetic programming (PMBGP) [9]. PMBGA studies the probabilistic modeling using GA's bit-string individual structures. PMBGP explores EDA to tree structures which provide more complex ways to represent solutions for program evolution. (3) For different problem domains, EDA can be grouped into discrete EDAs and continuous EDAs, which solve the optimization problems of discrete domain [4,8] and continuous domain [10–13].

A novel EDA, called probabilistic model building genetic network programming (PMBGNP), was recently proposed [14–16]. PMBGNP is inspired by the classical EDAs, however, a distinguished directed graph (network) structure [17–21] is used to represent its individual. Hence, it can be viewed as a graph EDA that extends conventional EDAs like bit-string structure based PMBGA and tree-structure based PMBGP. The fundamental points of PMBGNP are:

1. PMBGNP allows higher expression ability by means of graph structures than conventional EDAs.
2. Due to the unique features of its graph structures, PMBGNP explores the applicability of EDAs to wider range of problems, such as data mining [22,14,23] and the problems of controlling the agents' behavior (agent control problems) [16,24–26].

In the previous research, it has been demonstrated that PMBGNP can successfully outperform classical EAs with the above problems.

However, PMBGNP is mainly designed for discrete optimization problems. In other words, it cannot deal with (or directly handle) continuous variables which are widely existed in many real-world control problems. To solve this problem, the simplest way is to employ discretization process to transfer the continuous variables into discrete ones, however, which will cause the loss of solution precision.

This paper is dedicated to an extension of PMBGNP to continuous optimization in agent control problems. Different from most of the existing continuous EDAs developed by incremental learning [10], maximum likelihood estimation [11], histogram [27] or some other sorts of machine learning techniques [28–32], the proposed algorithm employs the techniques of reinforcement learning (RL) [33], such as actor critic (AC), as the mechanism to estimate the probability density functions (PDFs) of the continuous variables. Although most of the classical continuous EDAs formulate the PDFs of continuous variables by Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, the proposed algorithm applies AC to calculate the temporal-difference (TD) error to evaluate whether the selection (sampling) of continuous values is better or worse than expected. Based on the idea of trial-and-error, a scalar reinforcement signal which can decide whether the tendency to select the sampled continuous value should be strengthened or weakened is formulated by the gradient learning for the evolution of Gaussian distribution (μ and σ).

Most importantly, as an extension of PMBGNP, the proposed algorithm mainly possesses the ability to solve the agent control problems, rather than the conventional continuous EDAs only for function optimization problems. Accordingly, the applicability of continuous EDAs is explored in certain degrees.

In this paper, the proposed algorithm is applied to control the behavior of a real autonomous robot, Khepera robot [34,35], in which the robot's wheel speeds and sensor values are continuous variables. To evaluate the performance of this work, various classical algorithms are selected from the literature of standard EAs, EDA and RL for comparison.

The rest of this paper is organized as follows. Section 2 briefly introduces the original framework of PMBGNP in the discrete domain. In Section 3, extending PMBGNP to continuous domain is explained in details. The experimental study is shown in Section 4. Finally we conclude this paper in Section 5.

2. Probabilistic model building genetic network programming

2.1. Directed graph (network) structure

From the explicit viewpoint, PMBGNP distinguishes itself from the classical EDAs by using a unique directed graph (network) structure to represent its individual, depicted in Fig. 1. The directed graph structure is originally proposed in a newly graph-based EA named Genetic Network Programming (GNP) [17,18,36]. Three types of nodes are created to form the program (individual) of GNP:

- *Start node*: it has no function and conditional branch.
- *Judgment node*: it has its own judgment function and multiple conditional branches.
- *Processing node*: it has its own processing function but no conditional branch.

Each program is composed of one start node, multiple judgment and processing nodes. Start node only plays the role on deciding the first node to be executed. Judgment nodes imitate the "if-then" decision-making functions to deal with the specific inputs of the problems, such as the sensor values of the robot. Processing nodes enforce the action functions for task solving, such as determining the wheel speeds of the robot. By separating judgment and processing functions, the distinguished directed graph can deal with various combinations of judgments and processing to efficiently evolve the compact programs by only selecting the necessary judgments and processing. Such separation and selection by necessity can efficiently generate partially observable markov decision process (POMDP) [36] and ensure high generalization ability. The number of judgment and processing nodes is defined in advance and problem specific. As a result, the directed graph never causes the bloat problem of GP. Although a small number of nodes is prepared, such a structure can obtain good performance by well realizing the repetitive process by the frequent reuse of nodes.

More empirically, the directed graph can be encoded into bit-strings as shown in Fig. 1, which is defined by a tuple

$$G = (N_{\text{node}}, B, \text{LIBRARY}),$$

where N_{node} and B are the sets of nodes and branches in an individual, respectively; *LIBRARY* is a set of judgment and processing functions given by the tasks. Each node $i \in N_{\text{node}}$ is defined by a tuple [2]

$$i = (NT_i, NF_i, B(i), C_i).$$

NT_i defines the node type, where 0, 1 or 2 for start, judgment or processing node, respectively. $NF_i \in$ LIBRARY represents its function. $B(i)$ represents its set of branches. C_i consists of a set of C_{ik} indicating the node connected from the k_{th} branch of node i.

In standard GNP, NT_i, NF_i and $B(i)$ are generally unchanged, while evolution is carried out to change C_i, which means that the task of evolution is to find the optimal solution $g^* \in G$ by evolving the node connections.

In GNP, since the predefined and unchanged start node without function is only used to determine the first node to be executed, the start node and its branch are not considered in the formulation of G for simplicity.

[2] V_i and x_i denote the state-value and continuous variables of node i, which will be described in the next section. They are not included in the discrete PMBGNP.

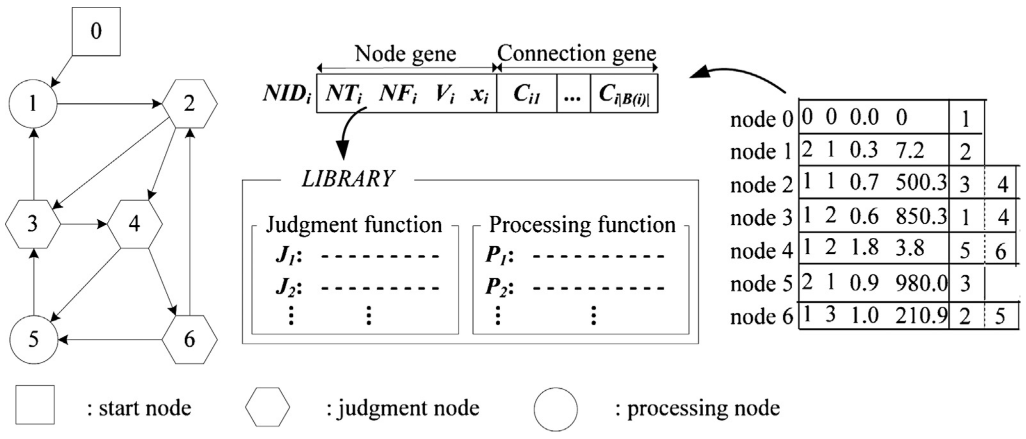


Fig. 1. Illustration of the directed graph structure.

In a certain respect, GNP can be considered as an extension of GP to graph structure, which naturally allows more flexibility and higher expression ability for some complex problems. In addition to a class of related EAs, including parallel algorithm discovery and orchestration (PADO) [37], parallel distributed GP (PDGP) [38], cartesian GP (CGP) [39] and evolutionary programming (EP) [40], GNP does not require the terminal node and the nodes can be connected arbitrarily. Most importantly, by separating judgment and processing functions, GNP can efficiently generate POMDP by selecting only the necessary judgments for the current state of the problems.

2.2. Probabilistic modeling

As indicated above, the role of evolution in GNP is to evolve the node connections to determine the optimal solutions. Different from standard GNP using stochastic crossover and mutation for evolution, PMBGNP enforces a probabilistic model from a set of selected individuals (i.e., top N individuals by truncation selection), and uses the model to generate new population.

The probabilistic modeling of PMBGNP is inspired by the univariate EDAs [8,41]. The probabilistic model P_{nc} is composed of a set of probabilities $P_{nc}(b(i), j)$, which indicates the connection probability from branch $b(i)$ of node i to node j, as shown by:

$$P_{nc} = \{P_{nc}(b(i), j) | \forall i, j \in N_{\text{node}}; \forall b(i) \in B(i)\}. \quad (1)$$

To calculate the probabilities of node connections, various approaches proposed in PMBGNP [14,16,24] can be applied. In this paper, we use a RL [33] based method called reinforced PMBGNP (RPMBGNP) [16] to construct P_{nc}. In this method, the episodes of RL can be obtained during the execution of individuals:

Definition 1 *((Episode).).* Given an individual $g \in G$, an episode is defined by the sequence of node transitions obtained during the execution of g.

Each node connection is defined as a state-action pair[3] of RL. By factorizing the individuals to the sequences of state-action pairs, we can generate and maintain the state-action Q table efficiently. The Q values can measure the qualities of their corresponding node connections, which are used to calculate the probabilities of node connections.

In each generation, the episodes from the top N individuals are selected, in which the Q values of the state-action pairs (substituted by the set of node connections in PMBGNP) are updated by RL. Since the aim of applying RL is to accumulate the actual knowledge of the individuals, an on-policy method called Sarsa Learning (Sarsa) [33] which uses the true experience of the agents is selected to update the Q values. Suppose the current state-action pair at time step t is node connection $(b(i), j)$, and the state-action pair at time step $t+1$ is node connection $(b(j), k)$. Then, the Q value is updated by:

$$Q(b(i), j) \leftarrow Q(b(i), j) + \alpha_s[r_j + \gamma_s Q(b(j), k) - Q(b(i), j)], \quad (2)$$

where, α_s and γ_s: learning rate and discount factor of Sarsa. r_j: reward of choosing node j at branch $b(i)$ of node i, and,

1. If j is a judgment node, $r_j = 0$.
2. If j is a processing node, r_j is given after processing node j.

The procedure of updating Q values in each generation is shown in Fig. 2. With this procedure, the good state-action pairs will be rewarded with higher Q values, and vice-versa, accordingly, which explicitly shows the quality of the substituted node connection. The probabilistic modeling is hence derived by incorporating such learnt knowledge:

$$P_{nc}(b(i), j) = \frac{\exp((Q(b(i), j))/(T))}{Z(b(i))}, \quad (3)$$

and $Z(b(i))$ is the normalization function calculated by:

$$Z(b(i)) = \sum_{j' \in A(b(i))} \exp\left(\frac{Q(b(i), j')}{T}\right), \quad (4)$$

where, $A(b(i))$ is the set of possible nodes connected from branch $b(i)$ of node i.

Boltzmann distribution is used to relax the sample pressure, due to the sensitive diversity loss of PMBGNP (The proof of its diversity loss can be found in [16]). The temperature parameter T is defined adaptively as follows:

$$T = \frac{\tau}{t+1}, \quad (5)$$

where, τ is a coefficient and t is the current number of generations.

3. Extending PMBGNP to continuous domain

PMBGNP is dedicated to solve the discrete optimization problems, since its search space is formulated by the node connections C_i for all $i \in N_{\text{node}}$ while the function of each node is fixed and unevolvable. In other words, for the problems including functions with continuous variables that take any real numbers within the given intervals, discretization should be carried out to transform

[3] In RPMBGNP, a state is defined as a branch of a node in PMBGNP, and an action is defined as a node. Therefore, node connection $(b(i), j)$ is equivalent to state-action pair $(b(i), j)$.

```
1: $g = 1$
2: while $g \leq N$ do
3:     execute individual $g$, obtain the sequence of state-action pairs of Sarsa
4:     update the $Q$ values using Eq. (2)
5:     $g \leftarrow g + 1$
6: end while
```

Fig. 2. Pseudocode of updating Q values by Sarsa.

the problems to discrete cases in PMBGNP. By discretization the continuous variables are substituted by a set of constant values or segmentations of sub-intervals, denoted as distinct functions in *LIBRARY*. However, this will cause the problem of losing the solution precision. Proposing a continuous version of PMBGNP would be a meaningful work by making it more general in order to adapt to wider range of problems.

3.1. Individual representation

In order to make the directed graph structure G suitable for expressing continuous search space, each node $i \in N_{\text{node}}$ is defined by a tuple $i = (NT_i, NF_i, x_i, B(i), C_i)$. Comparing with discrete PMBGNP, an additional variable x_i is added, which is a continuous variable in a range of $[\texttt{lower}_i, \texttt{upper}_i]$, denoting the lower and upper bounds of variable x_i, as an example shown in Fig. 3.

Most attempts on extending EDA to continuous domains are to use Gaussian distribution [10,11,42]. The advantages of utilizing Gaussian distribution is its simplicity of implementation and without loss of generality for solutions which ensures the performance in continuous domains. As a result, this paper straightforwardly applies Gaussian distribution for the representation of continuous variables in each nodes, denoted by $\mathcal{N}_i(\mu, \sigma^2)$ for $\forall i \in N_{\text{node}}$. Thus, the evolution of each x_i is transformed to the updating of the mean μ and standard deviation σ of its Gaussian distribution.

3.2. Structure of the probabilistic model

Let P_{cv} be the probabilistic model of continuous variables, which consists of a set of probabilities $P^i_{cv}(x; \mu, \sigma)$, denoting the probability density function (pdf) of continuous variable x_i of node i. Thus, we have:

$$P_{cv} = \{P^i_{cv}(x_i; \mu, \sigma) | \forall i \in N_{\text{node}}\}. \tag{6}$$

and $P^i_{cv}(x_i; \mu, \sigma)$ is described by:

$$P^i_{cv}(x_i; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x_i - \mu)^2}{2\sigma^2}\right], \tag{7}$$

where x_i is the continuous variable of node i.

3.3. Probabilistic modeling

In order to evolve the mean μ and standard deviation σ, one may concern with the incremental learning [10] or maximum likelihood estimation [11]. In this paper, we propose a novel method to update the Gaussian distribution, which contains the following parts.

3.3.1. Gradient calculation

First, we calculate the partial derivative of parameter μ and σ for each variable as follows:

$$\frac{\partial P_{cv}(x; \mu, \sigma)}{\partial \mu} = \underbrace{\frac{2}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x - \mu)^2}{2\sigma^2}\right]}_{>0} \times (x - \mu), \tag{8}$$

$$\frac{\partial P_{cv}(x; \mu, \sigma)}{\partial \sigma} = \underbrace{\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x - \mu)^2}{2\sigma^2}\right]}_{>0} \times \left[\frac{(x - \mu)^2}{\sigma^2} - 1\right] \tag{9}$$

As indicated, the first terms in the right side of Eqs. (8) and (9) are always greater than 0. Therefore, inspired by the idea of gradient descent we can obtain the updating directions of μ and σ as follows given a sampled value x of the Gaussian distribution:

$$\nabla(\mu; x) = x - \mu, \tag{10}$$

$$\nabla(\sigma; x) = \frac{(x - \mu)^2}{\sigma^2} - 1. \tag{11}$$

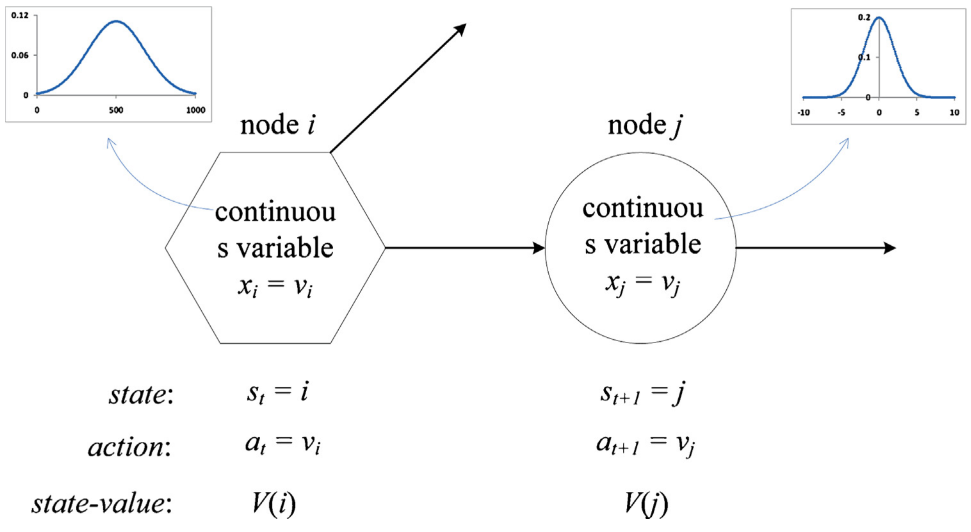

0.12
0.06
0
0
500
1000
node i
continuou s variable $x_i = v_i$
0.2
0.1
0
-10
-5
0
5
10
node j
continuou s variable $x_j = v_j$
state: $s_t = i$ $s_{t+1} = j$
action: $a_t = v_i$ $a_{t+1} = v_j$
state-value: $V(i)$ $V(j)$

Fig. 3. Representation of the continuous variables in PMBGNP-AC.

These two equations are the simplified versions of Eqs. (8) and (9). Then, we have the following updating rules of μ and σ:

$$\mu \leftarrow \mu + \alpha_\mu \nabla(\mu; x), \tag{12}$$

$$\sigma \leftarrow \sigma + \alpha_\sigma \nabla(\sigma; x), \tag{13}$$

where α_μ and α_σ are the learning rates (step size) of the corresponding parameters, respectively.

3.3.2. *Integrating actor critic (AC)*

Second, based on the calculated gradients, a novel algorithm is proposed by integrating a RL technique – Actor critic (AC) [33].

In RL, an agent is interacted with its environment through observations and actions. At every step, the agent observes the current state of the environment, then chooses an action to change the state of the environment. At every step of choosing actions, a scalar reinforcement value is formulated according to the reward the agent obtains. This value is backwardly sent to the agent which allows the modification of its actions to maximize the reinforcement value.

Based on this idea, the continuous version named PMBGNP-AC is proposed to update the Gaussian distribution. To incorporate AC, we define the state and action according to the structure of PMBGNP as follows:

Definition 2 *((State).).* State *s* is defined as a node in the directed graph.

Definition 3 *((Action).).* Action *a* is defined as the selection of the values in the continuous variable of each node.

Fig. 3 shows an example of such definitions. Note that these two definitions are different from that of Sarsa used in P_{nc} which shows the form of node connections. In AC, the set of states are denoted by N_{node} and the action space is infinite and bounded according to the range of each continuous variable. With such appropriate definitions, AC can be easily incorporated, where the Gaussian distribution can be regarded as the *actor* since it is used to select the actions (sample continuous variables), and the *critic* is formulated as the state-value function *V* to criticize the actions made by the actor.

At each action selection, the critic evaluates the new state to determine whether this selection is better or worse than expected. The evaluation is formulated by the temporal-difference (TD) error δ as follows:

$$\delta_t = r_t + \gamma_{ac} V(s_{t+1}) - V(s_t), \tag{14}$$

where, r_t: reward obtained by the agent at time step *t*. $V(s_t)$: value function of state *s* at time step *t*. γ_{ac}: discounted factor of AC.

After obtaining the TD error of each time step, it is sent back to update the state-value function *V* by:

$$V(s_t) \leftarrow V(s_t) + \alpha_{ac}\delta_t, \tag{15}$$

where, α_{ac}: learning rate of AC.

This TD error can evaluate the action of each time step. If δ_t is positive, it suggests that the tendency to select a_t should be strengthened, and vice-versa. Accordingly, we formulate a scalar reinforcement signal θ_t to indicate whether the tendency to select this action should be strengthened or weakened.

$$\theta_t = \begin{cases} -1, & \text{for} \quad \delta_t < 0 \\ 0, & \text{for} \quad \delta_t = 0 \\ 1, & \text{for} \quad \delta_t > 0 \end{cases} \tag{16}$$

```
1:  $t \leftarrow 0$;
    Initialize population $Pop(t)$ with the size of $M$;
    Initialize the probabilistic models $P_{nc}$ and $P_{cv}$;
    Initialize $Q(S, A)$ and $V(S)$;
2:  Evaluate the fitness of $Pop(t)$;
3:  Select top $N$ individuals;
4:  for $g = 1, 2, ..., N$ do
5:      Update $Q$ values by Eq. (2);                /* Sarsa */
6:      Update TD error $\delta$ by Eq. (14);        /* Actor Critic */
        Update $V$ values by Eq. (15);
7:      Construct $P_{cv}$ by updating $\mu$ and $\sigma$ using Eq. (17) and (18);
8:  end for
9:  Construct $P_{nc}$ by Eq. (3);
10: Generate $Pop(t+1)$ by sampling $P_{nc}$ and $P_{cv}$ according to Eq. (19);
    $t \leftarrow t + 1$;
11: Go back to step 2 until the terminal condition is met.
```

Fig. 4. Pseudocode of PMBGNP-AC.

3.3.3. *Updating the probabilistic model*

Inserting the scalar reinforcement signal of Eq. (16) into Eqs. (12) and (13), we get the final updating rules of the Gaussian distribution of PMBGNP-AC as follows:

$$\mu \leftarrow \mu + \alpha_\mu \nabla(\mu; x)\theta_t, \tag{17}$$

$$\sigma \leftarrow \sigma + \alpha_\sigma \nabla(\sigma; x)\theta_t. \tag{18}$$

3.4. *Algorithm*

The algorithm of PMBGNP-AC is a combination of the probabilistic model P_{nc} of node connections and P_{cv} of continuous variables of each node. As a result, the final probability of generating individual *g* by PMBGNP-AC is:

$$P(g) = \prod_{i \in N_{\text{node}}} \left[P^i_{cv}(x_i; \mu_i, \sigma_i) \prod_{b(i) \in B(i)} P_{nc}(b(i), j) \right]. \tag{19}$$

The detailed pseudocode of the proposed algorithm PMBGNP-AC is described in Fig. 4. The initial *Q* values and *V* values are prepared in advance, which are set to zero in this paper. P_{nc} is initialized to uniform distribution, while initial values of μ and σ in P_{cv} is determined problem-specifically.

3.5. *Discussion*

It is easily observed that Eqs. (12) and (13) if using the sampled values *x* from the promising individuals would be almost identical to the classical continuous EDAs, such as continuous population-based incremental learning(PBILc) [10] and continuous univariate marginal distribution algorithm (UMDAc) [11]. In other words, classical continuous EDAs tend to update μ and σ of each variable toward the values obtained from the promising individuals.[4] Thus, they hold the assumption that the values of variables from the promising individuals would be the correct and target ones. However, this arises problems: First, all the sampled values from the promising individuals are treated equally, regardless of the significance of the corresponding quality. Second and most importantly, this assumption may not be true since the fitness evaluation is defined for the entire individual structures rather than the sampled value of each variable.

[4] Also away from the worst individual in the case of PBILc, since it also uses the sampled value of the *worst* individual to update the Gaussian distribution.

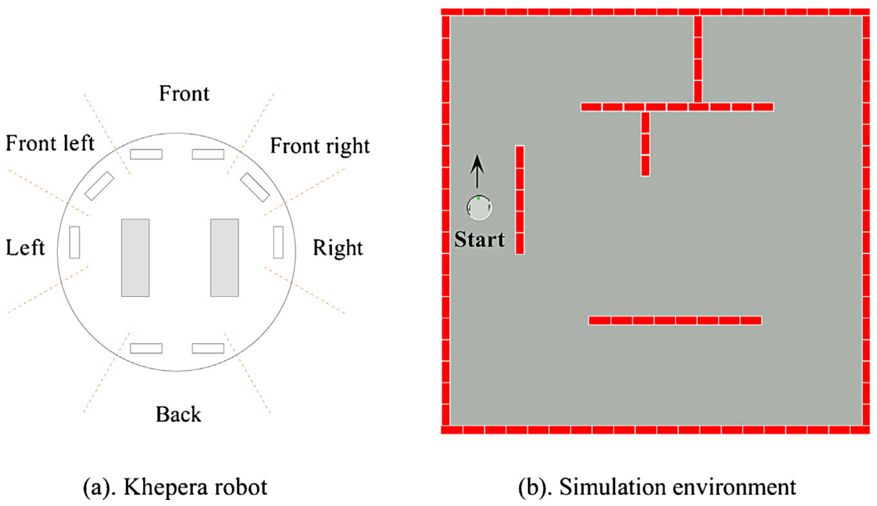


Fig. 5. Khepera robot.

Nonetheless, the proposed method drives the updating directions from another angle to overcome the above problems. That is, it first factorizes each individual to a sequence of state-action pairs, and applies AC to measure the quality of each sampled value by calculating TD error δ. After calculating δ for each promising individual, it is used to formulate the scalar reinforcement signal θ which is capable of measuring the quality of the sampled value. If it is denoted as a good value, Gaussian distribution will be updated toward this value by setting $\theta = 1$. Otherwise, Gaussian distribution will be updated away from the sampled value by setting $\theta = -1$. Consequently, each sampled value is evaluated more precisely by AC rather than the fitness of the entire individual.

Moreover, although the probabilistic model itself is represented by the univariate Gaussian distribution as that of classical PBILc and UMDAc, the updating of each variable is influenced by the other variables implicitly, since at each time t, the reinforcement signal θ_t is formulated with the consideration of the future state-values, i.e., $V(s_{t+1})$, by viewing Eqs. (14)–(16) as a whole. This provides an attempt to construct a more accurate probabilistic model without increasing its complexity in EDA community, which differs from the traditional works on multivariate EDAs [43,44].

4. Experiments

Different from most of the conventional EDAs doing function optimization problems, PMBGNP-AC is applied to controlling the behaviors of an autonomous robot – Khepera robot [34,35], which can be classified to a kind of Reinforcement Learning (RL) problems [33].

4.1. Problem description

Khepera robot [34,35] is a 5.5 cm wheeled mobile robot including 8 sensors. Each sensor can return a continuous value ranging from 0 (no object in front) to 1023 (an object is almost touching). Two wheel motors can take speeds ranging in [−10,10]. Different combinations of the two speeds can control the robots for different moving behaviors.

The experimental environment is shown in Fig. 5. A benchmark problem – Wall-Following problem – is selected for evaluation, in which the robot should find a strategy to move 1) along the wall, 2) as fast as possible and 3) as straight as possible. Based on the above three goals, the reward of each step is defined by:

$$Reward = \underbrace{C}_{1)} \times \underbrace{\frac{v_R + v_L}{20}}_{2)} \times \underbrace{\left(1 - \sqrt{\frac{|v_R - v_L|}{20}}\right)}_{3)}, \tag{20}$$

where, v_R, v_L: the speed of right and left wheels. C: value defined by

$$C = \begin{cases} 1, & \text{allthesensorvaluesarelessthan1000,} \\ & \text{andatleastoneofthemismorethan100,} \\ 0, & \text{otherwise.} \end{cases}$$

The final fitness is the average reward over all ST steps:

$$Fitness = \frac{\sum_{step=1}^{ST} Reward}{ST}. \tag{21}$$

where, ST: user-defined limited steps.

In this paper, ST is set to $\{100, 200, 300, 400, 500\}$ to simulate the problems from simple cases to complex ones, which can be viewed as the problem size in function optimization problems [4].

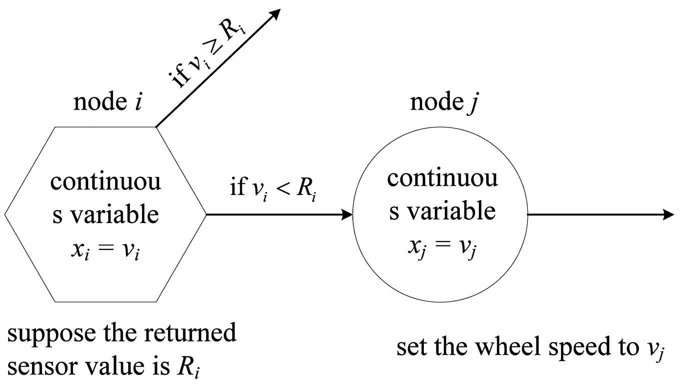
4.2. Node functions and continuous variables

The node functions are shown in Table 1. There are total 8 judgment functions for judging the sensors' values of the robot (as shown in Fig. 5). In this paper, the number of branches of each judgment node is set at 2 to efficiently implement the IFLTE(a, b, c, d) function. The roles of continuous variables in judgment/processing nodes to be optimized are as follows:

- **Judgment node** (i.e., node i): continuous variable x_i samples a value v_i to divide the domain of its sensor value into two intervals

Table 1
Node functions used for Khepera robot.

Node	*NF*	Function	Domain
$J_1, J_2, \ldots, J_8$	1, 2,..., 8	Judge the value of the sensor of 1, 2,..., 8	[0, 1023]
P_1, P_2	1, 2	Determine the speed of the right/left wheel	[−10, 10]



node i
if $v_i \geq R_i$
node j
continuous variable $x_i = v_i$
if $v_i < R_i$
continuous variable $x_j = v_j$
suppose the returned sensor value is R_i
set the wheel speed to v_j

Fig. 6. Roles of continuous variables in judgment/processing nodes.

([0, v_i) and [v_i, 1023]), where the selection of branches is determined by the comparison of real returned value and v_i.

- **Processing node** (i.e., node j): continuous variable x_j ([−10, 10]) formulates the speed of its corresponding wheel motor by sampling value v_j.

These two kinds of continuous variables are to be evolved to determine the final solutions, as an example shown in Fig. 6.

4.3. Compared algorithms and parameter settings

To evaluate the performance of the proposed algorithm, this paper selects several state-of-the-art algorithms from the conventional EAs, EDAs and RL for comparison, which are shown as follows:

(1) Classical EAs: genetic programming (GP) [45] and GNP [17] are selected as the representative algorithms for comparing this study with conventional EAs. GP uses complete 2-ary trees to represent its individual, while FULL method is used for initialization. GNP uses the directed graph structure to express its solutions as described in Section 2.1. Standard crossover and mutation are used to evolve the individuals of EAs.

(2) Discrete PMBGNP: RPMBGNP [16] is selected to confirm the superiority of the proposed continuous algorithm over the discrete one. In RPMBGNP, the continuous variables are discretized in advance. For judgment nodes, the domain is divided into two fixed intervals, [0, 1000) and [1000, 1023] based on the definition of C in Eq. (20). In processing nodes, the domain is discretized into the set of $\{-10, -5, 0, 5, 10\}$ to determine the robot's speed. Such a discretization is based on the previous study [16] which has shown to be sufficient for solving the problems.

(3) Classical EDAs: PBILc [10] – one of the most famous continuous EDAs – is selected to compare the proposed algorithm with the classical continuous EDAs with univariate interaction. In PBILc, the mean μ is updated based on the best two and worst one individuals:

$$\mu \leftarrow (1-\alpha_\mu)\mu + \alpha_\mu (x^{best,1} + x^{best,2} - x^{worst}), \quad (22)$$

and σ is updated by memorizing the variance of the N promising individuals:

$$\sigma \leftarrow (1-\alpha_\sigma)\sigma + \alpha_\sigma \sqrt{\frac{\sum_{i=1}^{N}(x^i-\mu)^2}{N}}. \quad (23)$$

4) Classical RL: Sarsa [33] is selected as a state-of-the-art RL algorithm for comprehensive study. The discretization of Sarsa is done similarly as PMBGNP, where the state space is defined by the combinations of sensor values and the action space is defined by different settings of the robot's speed. ϵ-greedy policy is used to balance the exploitation-exploration.

The detailed experimental settings are reported in Table 2, where all the values are determined by trial-and-error to perform the best of each algorithm [16].

In addition to the original RPMBGNP, four additional parameters should be carefully configured to control the effect of the proposed PMBGNP-AC. Three of them correspond to the learning rates, that is, α_{ac} of AC, α_μ of the mean value μ and α_σ of the standard deviation σ. In general, too small values of learning rates will lead to the low learning efficiency, while if too large values are set, the result may diverge. The discount factor γ_{ac} of AC determines how much future reward is taken into account for the updating of state values. Small γ_{ac} denotes that the agent only cares about the immediate reward obtained by the action taken, while high γ_{ac} causes the state values to be updated by more strongly considering future rewards. In order to tune these parameters, relatively small learning rates and large discount factor are recommended, as the ones shown in Table 2.

4.4. Experimental results and analysis

Five experiments of $ST \in \{100, 200, 300, 400, 500\}$ are carried out to testify the effectiveness and scalability of the proposed algorithm. All the experiments were done by a PC with Intel Core i5 processor running at 2.70 GHz with 8 GB of RAM. The OS used was Ubuntu 10.10 release. The final experimental results are the average over 30 independent runs.

4.4.1. Fitness values

The fitness curves of the compared algorithms are shown in Fig. 7. Each detailed fitness value is indicated in Table 3, together with the associated standard deviation. The bold values denote the

Table 2
Simulation conditions.

	GP	GNP	RPMBGNP	PBILc	PMBGNP-AC
Population size M	300	300	2000	2000	2000
– Elite ind.	1	1	100	100	100
– Crossover ind.	120	120	–	–	–
– Mutation ind.	179	179	–	–	–
– Promising ind. N	–	–	1000	1000	1000
Program size $\lvert N_{node}\rvert$	127	60	60	60	60
Crossover rate	0.9	0.1	–	–	–
Mutation rate	0.02	0.02	–	–	–
Other parameters	(— Sarsa-Learning) α_s = 0.1, γ_s = 0.9 (— Actor critic) α_{ac} = 0.1, γ_{ac} = 0.9 α_μ = 0.05, α_σ = 0.05 (— ϵ-greedy policy) ϵ = 0.1				
Terminal condition	300,000 fitness evaluations				

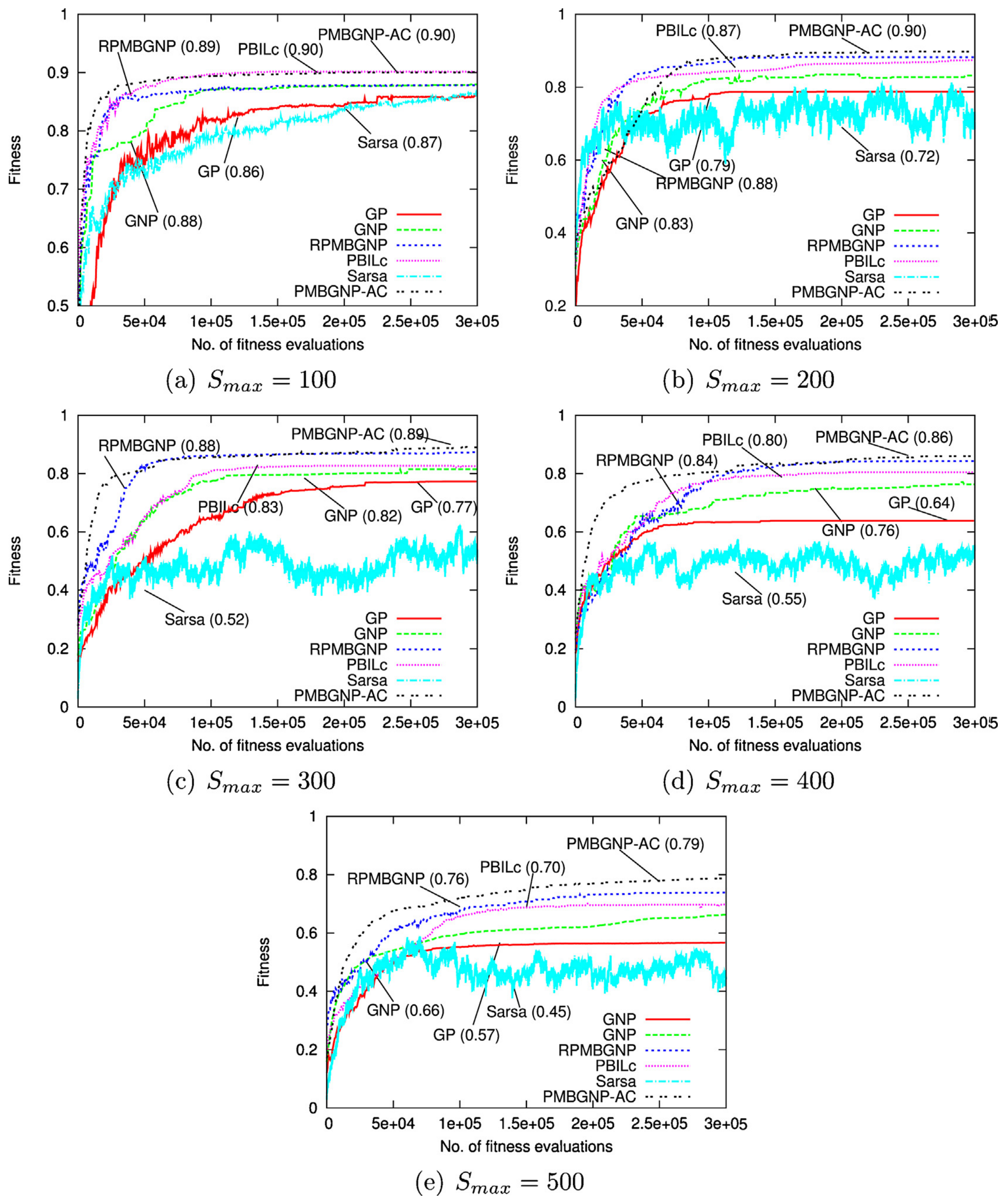


Fig. 7. Fitness curves in five wall-following problems.

best results of the problems obtained among the compared algorithms. The analysis of different algorithms is performed as follows:

In all the five problems with different problem sizes, the conventional EAs, i.e., GP and GNP, performs worse than the variants of EDAs, such as RPMBGNP and the proposed algorithm, which is due to the lack of evolution ability by standard genetic operators. Throughout the generations, the directed graph structure allows GNP and the variants of PMBGNP to achieve much better fitness values than the tree structure based GP in terms of higher expression ability. Although Sarsa discretizes the continuous search space into

Table 3
The fitness values over 30 independent runs.

	Fitness value ± Standard deviation				
ST	100	200	300	400	500
GP	0.86 ± 0.05	0.79 ± 0.15	0.77 ± 0.15	0.64 ± 0.08	0.57 ± 0.10
GNP	0.88 ± 0.02	0.83 ± 0.11	0.82 ± 0.09	0.76 ± 0.08	0.66 ± 0.15
RPMBGNP	0.89 ± 0.01	0.88 ± 0.06	0.88 ± 0.03	0.84 ± 0.10	0.76 ± 0.09
PBILc	**0.90 ± 0.01**	0.87 ± 0.09	0.83 ± 0.09	0.80 ± 0.15	0.70 ± 0.12
Sarsa	0.87 ± 0.04	0.72 ± 0.11	0.52 ± 0.21	0.55 ± 0.19	0.45 ± 0.22
PMBGNP-AC	**0.90 ± 0.02**	**0.90 ± 0.03**	**0.89 ± 0.10**	**0.86 ± 0.14**	**0.79 ± 0.11**

Table 4
The required fitness evaluations over 30 independent runs.

	RFEs ± Standard deviation				
ST	100	200	300	400	500
GP	67, 200 ± 56, 564	140, 427 ± 97, 119	164, 180 ± 82, 590	208, 893 ± 121, 929	268, 827 ± 70, 595
GNP	52, 300 ± 47, 379	107, 733 ± 76, 205	132, 000 ± 101, 807	177, 300 ± 87, 890	211, 833 ± 111, 542
RPMBGNP	18, 000 ± 14, 325	37, 377 ± 14, 584	51, 167 ± 32, 248	73, 593 ± 21, 606	98, 500 ± 75, 761
PBILc	**9500 ± 7319**	82, 400 ± 107, 993	87, 800 ± 67, 612	118, 300 ± 96, 899	167, 267 ± 93, 515
Sarsa	52, 800 ± 51, 392	149, 850 ± 97, 569	194, 707 ± 99, 035	216, 529 ± 106, 187	232, 970 ± 100, 870
PMBGNP-AC	9633 ± 7513	**35, 483 ± 29, 800**	**45, 500 ± 41, 924**	**61, 467 ± 52, 881**	**73, 100 ± 22, 129**

a finite state-action space, it still requires to update a large size of *Q* table. This might cause the slow learning speed. Therefore, Sarsa can only work well in simple problems, i.e., *ST* = 100, but fail in the complex ones. On the other hand, even though discretization process is carried out in RPMBGNP to simplify the search space, RPMBGNP achieves good performance to solve the Wall-Following problems comparing with the other classical algorithms except the proposed one. By balancing the exploitation-exploration using Boltzmann distribution [16], RPMBGNP obtains very stable performances in terms of the smallest standard deviation.

Overall, the proposed algorithm can achieve the best fitness values than the others. By formulating the continuous search space using Gaussian distribution, the solution precision is preserved, hence more accurate solutions are found. On the other hand, to evaluate the effect of the proposed algorithm of integrating AC, we perform the comparison with the classical algorithm PBILc. In the case of PBILc, only the information of three individuals (the best two and worst one) are used to update μ of Gaussian distribution, due to the assumption that the fitness evaluation could correctly measure the quality of each sampled value, however, which might not be true as discussed in Section 3.5. Moreover, such an updating process might cause that the search space is explored in a too restricted region causing the slow learning efficiency. Therefore, PBILc can ensure quite good fitness values in simple problems, i.e., *ST* = 100 and 200, however, obtain poor results when the problem size becomes large. PMBGNP-AC proposes a novel algorithm to update the pdf of Gaussian distribution without the assumption that PBILc holds. That is, it applies AC to measure the quality of each sampled value by calculating TD error δ. δ is used to measure the quality of each sampled value, replacing the role of fitness value used in conventional EDAs. The results show that PMBGNP-AC works quite well in directly handling the continuous variables comparing with the state-of-the-art algorithms.

4.4.2. *Required fitness evaluations and reliability*

To test the search speed of each compared algorithm, the average number of required fitness evaluations (RFEs) is further computed. The RFEs are calculated as follows: (1) For each successful run, where the robot can be controlled to correctly move along the wall, the exact number of fitness evaluations is counted; (2) For the failed run, where the robot cannot be controlled to move along the wall, the maximum number of fitness evaluation, i.e., 300,000, are counted. Finally, the RFEs of each algorithm are the average values of 30 independent runs.

The results of RFEs are shown in Table 4. All the compared algorithms can solve the simplest problem, i.e., *ST* = 100, successfully, however, with different RFEs. Overall, the proposed algorithm requires the smallest RFEs, while the others need much larger ones. The detailed RFEs are plotted in Fig. 8. It is found that the gaps among different algorithms become more significant with the increase of the problem size *ST*. When the problem size becomes large, the difference among the compared algorithms becomes clear.

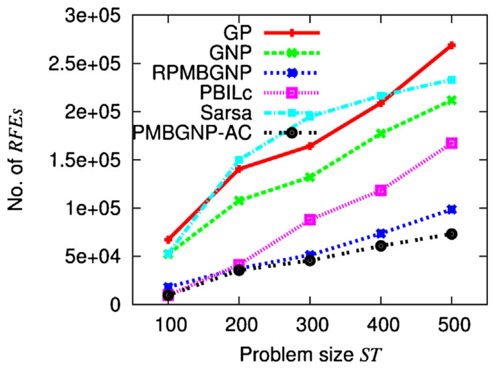


Fig. 8. Average fitness evaluation for five wall-following problems.

Table 5 compares the reliability of each algorithm in terms of showing the successful rate over 30 independent runs. In simple problems, the successful runs appear frequently in all algorithms. However, when the problem size becomes large, Sarsa and GP almost cannot find the acceptable trajectories under the limited fitness evaluations, while GNP can only find a small number of successful runs. The variants of EDAs have much higher probability than the classical algorithms. RPMBGNP can succeed in most cases but sometimes fail in complex problems. Overall, the proposed algorithm is capable of finding the acceptable trajectory in all the independent runs.

4.4.3. *Statistical analysis*

To compare the proposed algorithm with the classical ones in a sound statistical context, the *t*-test (two-tailed, paired) of each paired algorithms is performed in the contexts of both fitness values and RFEs.

The details *t*-test results (*p*-value) are reported in Table 6 and 7. The bold values of these two tables denote that there are statistically significant difference between PMBGNP-AC and the compared algorithm. It is found that PMBGNP-AC significantly outperforms most of the compared algorithms in terms of both fitness values and RFEs. Only limited exceptions can be found in comparison with RPMBGNP and PBILc, however, which happen in simple problems (small *ST*). When the problem becomes complex, i.e., *ST* = 500, PMBGNP-AC achieves the best results with statistical meaning.

Table 5
The successful rates over 30 independent runs.

	Successful rate				
ST	100 (%)	200 (%)	300 (%)	400 (%)	500 (%)
GP	**100**	76.7	63.3	36.7	20.0
GNP	**100**	90.0	76.7	70.0	43.3
RPMBGNP	**100**	**100**	**100**	**100**	93.3
PBILc	**100**	93.3	93.3	80.0	76.7
Sarsa	**100**	60.0	36.7	23.3	16.7
PMBGNP-AC	**100**	**100**	**100**	**100**	**100**

Table 6
The *t*-test results of fitness values.

	t-test (*p*-value)				
ST	100	200	300	400	500
PMBGNP-AC vs. GP	$\mathbf{3.13 \times 10^{-3}}$	$\mathbf{3.05 \times 10^{-4}}$	$\mathbf{3.86 \times 10^{-4}}$	$\mathbf{4.05 \times 10^{-8}}$	$\mathbf{5.28 \times 10^{-9}}$
PMBGNP-AC vs. GNP	$\mathbf{4.19 \times 10^{-3}}$	$\mathbf{7.49 \times 10^{-3}}$	$\mathbf{5.08 \times 10^{-3}}$	$\mathbf{2.26 \times 10^{-3}}$	$\mathbf{1.58 \times 10^{-3}}$
PMBGNP-AC vs. RPMBGNP	5.68×10^{-1}	1.76×10^{-1}	3.14×10^{-1}	1.84×10^{-1}	$\mathbf{8.90 \times 10^{-3}}$
PMBGNP-AC vs. PBILc	1.65×10^{-1}	1.20×10^{-1}	$\mathbf{1.11 \times 10^{-2}}$	1.26×10^{-1}	$\mathbf{1.24 \times 10^{-2}}$
PMBGNP-AC vs. Sarsa	$\mathbf{2.84 \times 10^{-3}}$	$\mathbf{1.32 \times 10^{-5}}$	$\mathbf{3.48 \times 10^{-9}}$	$\mathbf{4.05 \times 10^{-8}}$	$\mathbf{9.03 \times 10^{-9}}$

Table 7
The *t*-test results of RFEs.

	t-test (*p*-value)				
ST	100	200	300	400	500
PMBGNP-AC vs. GP	$\mathbf{5.87 \times 10^{-6}}$	$\mathbf{7.54 \times 10^{-7}}$	$\mathbf{6.01 \times 10^{-8}}$	$\mathbf{3.09 \times 10^{-8}}$	$\mathbf{2.24 \times 10^{-16}}$
PMBGNP-AC vs. GNP	$\mathbf{6.66 \times 10^{-5}}$	$\mathbf{9.50 \times 10^{-5}}$	$\mathbf{8.59 \times 10^{-5}}$	$\mathbf{1.30 \times 10^{-6}}$	$\mathbf{5.18 \times 10^{-8}}$
PMBGNP-AC vs. RPMBGNP	$\mathbf{1.59 \times 10^{-2}}$	7.57×10^{-1}	5.30×10^{-1}	$\mathbf{1.62 \times 10^{-2}}$	$\mathbf{4.71 \times 10^{-2}}$
PMBGNP-AC vs. PBILc	9.52×10^{-1}	$\mathbf{2.79 \times 10^{-2}}$	$\mathbf{2.67 \times 10^{-3}}$	$\mathbf{1.73 \times 10^{-3}}$	$\mathbf{1.68 \times 10^{-5}}$
PMBGNP-AC vs. Sarsa	$\mathbf{4.68 \times 10^{-5}}$	$\mathbf{2.05 \times 10^{-6}}$	$\mathbf{6.01 \times 10^{-8}}$	$\mathbf{2.99 \times 10^{-8}}$	$\mathbf{6.89 \times 10^{-10}}$

5. Conclusions

This paper extended a recent EDA algorithm named PMBGNP from the discrete domain to continuous cases. This study followed the conventional research on the topic of continuous EDAs and reformulated a novel method to learn Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ by a Reinforcement Learning method, i.e., Actor-critic (AC). The resulting PMBGNP-AC method can be thought as an extension of PBILc, where AC can implicitly update the PDF of Gaussian distribution by considering multivariate interactions. The results show that in the wall-following problems of autonomous robots, PMBGNP-AC outperforms several state-of-the-art algorithms, including conventional genetic operators based EAs, discretization based EDA, PBILc based variant and classical RL. Moreover, the scalability of PMBGNP-AC is confirmed by different settings of the problem size.

On the other hand, although EDA has been addressed to be the combination of EC and Machine Learning (ML) techniques, most of the existing EDAs use the techniques of Bayesian Network (BN) or some related probabilistic graphical models, and little attention has been deserved to RL techniques even though it is an important branch of ML. This work uses the techniques of RL to build its probabilistic models and shows attractive performance, which is expected to provide an attempt to bridge the gap between EDA and RL. Comparing with the BN based approaches, RL provides a direction to construct the accurate probabilistic models with less computational effort in EDA community.

In the future, the proposed method of incorporating AC will be extended from univariate Gaussian distribution to multivariate Gaussian distribution in order to model more complex variable interactions.

References

[1] J.H. Holland, Adaptation in Natural and Artificial Systems, Ann-Arbor, University of Michigan Press, Ann-Arbor, MI, USA, 1975.
[2] D.E. Goldberg, Genetic Algorithm in Search, Optimization and Machine Learning, Addison-Wesley, Boston, MA, USA, 1989.
[3] P. Larrañaga, J.A. Lozano, Estimation of distribution algorithms, in: A New Tool for Evolutionary Computation, Kluwer Academic Publishers, Boston, MA, USA, 2002.
[4] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, Linkage problem, distribution estimation, and bayesian networks, Evol. Comput. 8 (3) (2002) 311–341.
[5] J.-H. Zhong, J. Zhang, Z. Fan, MP-EDA: a robust estimation of distribution algorithm with multiple probabilistic models for global continuous optimization, in: Proc. of the Int'l Conf. on Simulated Evolution and Learning, 2010, pp. 85–94.
[6] M. Pelikan, D.E. Goldberg, F.G. Lobo, A survey of optimization by building and using probabilistic models, Comput. Optim. Appl. 21 (1) (2002) 5–20.
[7] Y. Hasegawa, H. Iba, A bayesian network approach to program generation, IEEE Trans. Evol. Comput. 12 (6) (2008) 750–764.
[8] H. Mühlenbein, G. Paaß, From recombination of genes to the estimation of distributions. I. Binary parameters, in: Proc. of the Conf. on Parallel Problem Solving from Nature, 1996, pp. 178–187.
[9] Y. Shan, R.I. McKay, D. Essam, H.A. Abbass, A survey of probabilistic model building genetic programming, in: M. Pelikan, K. Sastry, E. Cantú-Paz (Eds.), Scalable Optimization via Probabilistic Modeling, Springer-Verlag, Berlin, Germany, 2006, pp. 121–154, Ch. 6.
[10] M. Sebag, A. Ducoulombier, Extending population-based incremental learning to continuous search spaces, in: Proc. of the Conf. on Parallel Problem Solving from Nature, 1998, pp. 418–427.
[11] P. Larrañaga, R. Etxeberria, J.A. Lozano, J.M. Peña, Optimization by learning and simulation of Bayesian and Gaussian networks, Tech. Report EHU-KZAA-IK-4-99, Intelligent Systems Group, Dept. of Comput. Sci. and Artif. Intell., University of the Basque Country, 1999.
[12] Q. Zhang, H. Mühlenbein, On the convergence of a class of estimation of distribution algorithms, IEEE Trans. Evol. Comput. 8 (2) (2004) 127–136.
[13] Y. Wang, B. Li, A restart univariate estimation of distribution algorithm: sampling under mixed gaussian and lévy probability distribution, in: Proc. of the IEEE Congress on Evol. Comput., 2008, pp. 3917–3924.
[14] X. Li, S. Mabu, H. Zhou, K. Shimada, K. Hirasawa, Genetic network programming with estimation of distribution algorithms for class association rule mining in traffic prediction, in: Proc. of the IEEE Congress on Evol. Comput., CEC '10, 2010, pp. 2673–2680.
[15] X. Li, S. Mabu, K. Hirasawa, Towards the maintenance of population diversity: a hybrid probabilistic model building genetic network programming, Trans. Japan. Soc. Evol. Comput. 1 (1) (2010) 89–101.
[16] X. Li, S. Mabu, K. Hirasawa, A novel graph-based estimation of distribution algorithm and its extension using reinforcement learning, IEEE Trans. Evol. Comput. 18 (1) (2014) 98–113.
[17] K. Hirasawa, M. Okubo, H. Katagiri, J. Hu, J. Murata, Comparison between genetic network programming (GNP) and genetic programming (GP), in: Proc. of the IEEE Congress on Evol. Comput., CEC '01, 2001, pp. 1276–1282.
[18] S. Mabu, K. Hirasawa, J. Hu, A graph-based evolutionary algorithm: genetic network programming (GNP) and its extension using reinforcement learning, Evol. Comput. 15 (3) (2007) 369–398.
[19] S. Mabu, K. Hirasawa, Efficient program generation by evolving graph structures with multi-start nodes, Appl. Soft Comput. 11 (4) (2011) 3618–3624.
[20] X. Li, W. He, K. Hirasawa, Genetic network programming with simplified genetic operators, in: Neural Information Processing, Vol. 8227 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2013, pp. 51–58.
[21] X. Li, W. He, K. Hirasawa, Adaptive genetic network programming, in: Proc. of the IEEE Congress on Evol. Comput., 2014, pp. 1808–1815.
[22] X. Li, S. Mabu, H. Zhou, K. Shimada, K. Hirasawa, Genetic network programming with estimation of distribution algorithms and its application to association rule mining for traffic prediction, in: Proc. of the ICCAS-SICE, 2009, pp. 3457–3462.
[23] X. Li, S. Mabu, H. Zhou, K. Shimada, K. Hirasawa, Analysis of various interestingness measures in class association rule mining, SICE Journal of Control, Meas. Syst. Integr. 4 (4) (2011) 295–304.
[24] X. Li, B. Li, S. Mabu, K. Hirasawa, A novel estimation of distribution algorithm using graph-based chromosome representation and reinforcement learning, in: Proc. of the IEEE Congress on Evol. Comput., CEC '11, 2011, pp. 37–44.
[25] X. Li, S. Mabu, K. Hirasawa, Use of infeasible individuals in probabilistic model building genetic network programming, in: Proc. of the Genetic and Evol. Comput. Conf., GECCO '11, 2011, pp. 601–608.

[26] X. Li, S. Mabu, K. Hirasawa, An extended probabilistic model building genetic network programming using both of good and bad individuals, IEEJ Trans. Electr. Electron. Eng. 8 (4) (2013) 339–347.
[27] S. Tsutsui, M. Pelikan, D.E. Goldberg, Probabilistic model-building genetic algorithms using marginal histograms in continuous domain, in: Proc. of the KES'2001, 2001, pp. 112–121.
[28] F. Neri, G. Iacca, E. Mininno, Compact optimization, in: I. Zelinka, V. Snasel, A. Abraham (Eds.), Handbook of Optimization, Intelligent Systems, Springer-Verlag, Berlin, Germany, 2013, pp. 337–364.
[29] F. Neri, E. Mininno, G. Iacca, Compact particle swarm optimization, Inf. Sci. 239 (2013) 96–121.
[30] E. Mininno, F. Neri, F. Cupertino, D. Naso, Compact differential evolution, IEEE Trans. Evol. Comput. 15 (1) (2011) 32–54.
[31] E. Mininno, F. Cupertino, D. Naso, Real-valued compact genetic algorithms for embedded microcontroller optimization, IEEE Trans. Evol. Comput. 12 (2) (2008) 203–219.
[32] G. Iacca, F. Neri, E. Mininno, Compact bacterial foraging optimization, in: Proc. of the 2012 Int'l Conf. on Swarm and Evol. Comput., 2012, pp. 84–92.
[33] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Boston, MA, USA, 1998.
[34] Webots software. http://www.cyberbotics.com/
[35] Khepera robot. http://www.k-team.com/
[36] T. Eguchi, K. Hirasawa, J. Hu, N. Ota, A study of evolutionary multiagent models based on symbiosis, IEEE Trans. Syst. Man Cybern. B 36 (1) (2006) 179–193.
[37] A. Teller, M. Veloso, PADO: learning tree structured algorithms for orchestration into an object recognition system, Tech. Report CMU-CS-95-101, Carnegie Mellon University, 1995.
[38] R. Poli, Parallel distributed genetic programming, Tech. Report CSRP-96-15, in: School of Computer Science, The University of Birmingham, 1996.
[39] J.F. Miller, P. Thomson, Cartesian genetic programming, in: Proc. of the Eur. Conf. on Genetic Programming, 2000, pp. 121–132.
[40] D.B. Fogel, An introduction to simulated evolutionary optimization, IEEE Trans. Neural Netw. 5 (1) (1994) 3–14.
[41] S. Baluja, Population-based incremental learning: a method for integrating genetic search based function optimization and competitive leaning, Tech. Report CMU-CS-94-163, Carnegie Mellon University, 1994.
[42] S. Rudlof, M. Köppen, Stochastic hill climbing with learning by vectors of normal distributions, in: Proc. of the 1st Online Workshop on Soft Computing, 1996, pp. 60–70.
[43] M. Pelikan, Hierarchical Bayesian Optimization Algorithm, Springer, Berlin Heidelberg, 2005.
[44] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in: Proc. of the IEEE Congress on Evol. Comput., 2005, pp. 1769–1776.
[45] J.R. Koza, Genetic Programming, on the Programming of Computers by Means of Natural Selection, MIT Press, Boston, MA, USA, 1992.