A Continuous Estimation of Distribution Algorithm by Evolving Graph Structures Using Reinforcement Learning

Xianneng Li, Bing Li, Shingo Mabu and Kotaro Hirasawa
Graduate School of Information, Production and Systems, Waseda University, Japan
Email: {sennou@asagi., libing@asagi., mabu@aoni., and hirasawa@}waseda.jp

***Abstract*—A novel graph-based Estimation of Distribution Algorithm (EDA) named Probabilistic Model Building Genetic Network Programming (PMBGNP) has been proposed. Inspired by classical EDAs, PMBGNP memorizes the current best individuals and uses them to estimate a distribution for the generation of the new population. However, PMBGNP can evolve compact programs by representing its solutions as graph structures. Therefore, it can solve a range of problems different from conventional ones in EDA literature, such as data mining and Reinforcement Learning (RL) problems. This paper extends PMBGNP from discrete to continuous search space, which is named PMBGNP-AC. Besides evolving the node connections to determine the optimal graph structures using conventional PMBGNP, Gaussian distribution is used for the distribution of continuous variables of nodes. The mean value μ and standard deviation σ are constructed like those of classical continuous Population-based incremental learning (PBILc). However, a RL technique, i.e., Actor-Critic (AC), is designed to update the parameters (μ and σ). AC allows us to calculate the Temporal-Difference (TD) error to evaluate whether the selection of the continuous value is better or worse than expected. This scalar reinforcement signal can decide whether the tendency to select this continuous value should be strengthened or weakened, allowing us to determine the shape of the probability density functions of the Gaussian distribution. The proposed algorithm is applied to a RL problem, i.e., autonomous robot control, where the robot's wheel speeds and sensor values are continuous. The experimental results show the superiority of PMBGNP-AC comparing with the conventional algorithms.**

I. Introduction

As one branch of Evolutionary Computation (EC), Estimation of Distribution Algorithm (EDA) [1] has received much attention in recent years, where many algorithms have been proposed to draw its success. There are many ways to classify the algorithms in EDA, such as dependency of variables [2] and individual representation [3], [4]. Particulary, from the perspective of individual structures, EDA can mainly be classified into two groups, which are Probabilistic Model Building Genetic Algorithm (PMBGA) [5], [6], [7] and Probabilistic Model Building Genetic Programming (PMBGP) [3], [8]. Most of the current EDAs belong to such groups, where their individuals are represented by GA's string or GP's tree structures. Recently a novel algorithm named Probabilistic Model Building Genetic Network Programming (PMBGNP) [9], [10] has been proposed to extend EDA to graph structures. It uses the directed graph structures of Genetic Network Programming (GNP) [11], [12], [13] which is a kind of graph-based EAs to represent its individual. The characteristics of PMBGNP are: 1) Different from conventional EDAs, PMBGNP allows higher expression ability by means of graph structures. Therefore, it can be classified to a new group – graph-based EDA. 2) Due to the unique features of its graph structures, PMBGNP can be applied to solve a range of problems different from conventional ones in EDA literature, such as data mining [9] and Reinforcement Learning (RL) problems [4], [14].

However, standard PMBGNP is mainly designed for discrete optimization problems. Therefore, it cannot deal with (or directly handle) continuous variables. To solve this problem, discretization of continuous variables should be employed, which will cause the loss of solution precision. In this paper, a new PMBGNP algorithm is proposed to directly handle the continuous variables.

On the other hand, much progress has been made in the development of EDA for continuous domains [15], [16], [17]. This starts with the early attempt of continuous Population-based incremental learning (PBILc) [15] and continuous Univariate marginal distribution algorithm (UMDAc) [16], in which the probability density function is modeled by unidimensional Gaussian distribution and variable independencies assumption is employed. Naturally, some later research extend continuous EDA to cover multivariate interactions by factorization of multivariate Gaussian distribution [16], [18]. Some other continuous EDAs include Gaussian kernels, mixtures of Gaussian distribution [19], binary encoding by histograms [20], and variants using clustering [21] and niching [22] techniques.

This paper proposed a continuous PMBGNP named PMBGNP with Actor Critic (PMBGNP-AC). In PMBGNP-AC, Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ is used for the distribution of continuous variables of nodes. The mean value μ and standard deviation σ are constructed like those of classical PBILc [15]. However, a RL technique, i.e., Actor-Critic (AC) [23], [24], is designed to update the parameters (μ and σ). AC is applied to calculate the Temporal-Difference (TD) error to evaluate whether the selection of the continuous value (action) is better or worse than expected. We formulate the evaluation result as a scalar reinforcement signal which can decide whether the tendency to select this continuous value should be strengthened or weakened, allowing us to determine the shape of the probability density functions (μ and σ) of the

978-1-4673-1509-8/12/$31.00 ©2012 IEEE

Gaussian distribution.

The fundamental differences between PMBGNP-AC and conventional continuous EDAs are:

- PMBGNP-AC uses a directed graph structure to represent its solutions, differing from conventional GA's string-based approaches.
- Two probability distributions corresponding to node connections and continuous variables in each node are to be constructed and evolved simultaneously to determine the optimal solutions in PMBGNP-AC. This is quite different from conventional continuous EDAs which only consist of one probabilistic model of continuous variables due to their fixed string structures.
- PMBGNP-AC can model not only the univariate interactions explicitly as conventional PBILc, but also multivariate interactions implicitly according to AC.

Therefore, with these unique features, the probabilistic model of PMBGNP-AC can be viewed as a mixture of discrete (node connections) and continuous (continuous variables) distribution to represent the solutions. Nevertheless, most of the current continuous EDAs are designed for function optimization problems, where PMBGNP-AC is applied to a RL problem, i.e., autonomous robot control [25], in which the robot's wheel speeds and sensor values are continuous. The experimental results show the superiority and scalability of PMBGNP-AC comparing with the conventional algorithms.

The paper is organized as follows: Section II presents the proposed algorithm in details. The experimental study is carried out in section III. Section IV presents the conclusions and future work.

II. PMBGNP-AC

A. Individual Representation

PMBGNP-AC uses the directed graph structure of GNP [13] to represent its individual, which can be illustrated by the phenotype and genotype expression. Phenotype shows the directed graph structure, while genotype encodes the directed graph to computer programs.

As shown in Fig. 1, the nodes of the directed graph are encoded into bit-strings. Each node has a unique identification number NID_i unchanged during the evolution process, while the functions of the nodes with the same identification number in different individuals are the same. Let i represent a node number. NT_i defines the node type, where $NT_i = 0$ means a start node, $NT_i = 1$ means judgment nodes and $NT_i = 2$ denotes processing nodes. NF_i represents the function of node i, such as judgment and processing functions. d_i is the time delay spent on the judgment or processing of node i. C_{ik} indicates the node connected from node i by its k_{th} branch, and d_{ik} represents the time delay spent on this node transition. V_i is the state value of node i which is obtained by AC and used for updating the Gaussian distribution. x_i is the Gaussian continuous variable of node i and $B(i)$ is its set of suffixes of branches.

Each program (individual) is composed of one start node, multiple judgment nodes and processing nodes. The start

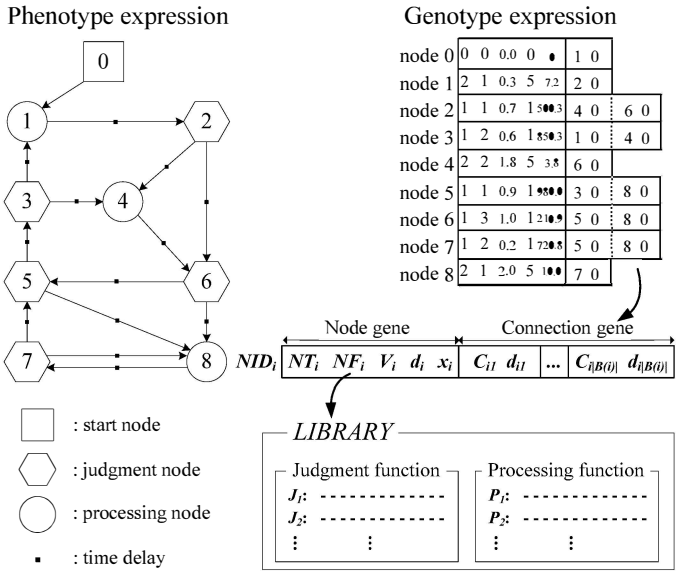


Fig. 1. The directed graph structure.

node having no function and conditional branch is only used to decide the first node to be transited. Judgment nodes work as "*if-then*" type decision-making functions to judge the environments by dealing with the specific inputs of the problems, such as the returned sensor values of the robot in this paper. Each judgment node has several conditional branches corresponding to several judgment results, which can be defined flexibly by the problems. Processing nodes preserve the processing/action functions to the environments, such as determining the wheel speeds of the robot. Processing node has no conditional branch, since the processing function only determines the agent's actions. By separating judgment and processing functions, the directed graph structure can handle various combinations of judgments and processings to efficiently evolve the compact programs by only selecting the necessary judgments and processings. And such separation and selection by necessity can efficiently generate Partially Observable Markov Decision Process (POMDP) [26].

The number of judgment nodes and processing nodes is predefined by designers appropriately. Therefore, the directed graph structure never causes the bloat problem [11], [13]. Meanwhile, even if a small number of nodes are prepared in advance, such a directed graph structure can perform quite well by realizing the repetitive processes based on the frequent reuse of nodes.

Furthermore, this distinguished directed graph structure has time delays in each node and branch. The motivation of time delays is to model the agents like human brain that needs the time for thinking. Time delays are developed in three classes: that spending on judgments, processings and node transitions. They can be defined by designers in advance for concrete problems, i.e., 1 time unit for judgments, 5 time units for processings and 0 time unit for node transitions in this paper. In addition, one step of an agent's behavior is defined by the

number of time units used in time delays. For example, in this paper, one step ends when five or more time units are used. That is, the agent can only do less than five judgments and one processing, or five judgments in one step. By introducing time delays, the directed graph structure can efficiently implement flexible programs considering the real-world environments, where agents need to solve problems in a limited time.

B. Probabilistic modeling

The probabilistic modeling of PMBGNP-AC consists of two parts: the distribution P_{nc} of node connections and the distribution P_{cv} of continuous variables.

1) Probabilistic modeling of node connections: The construction of P_{nc} is inspired by most of the classical EDAs, which directly studies its graph structure. It estimates the probabilities of connections between the nodes, where $P_{nc}(b(i), j)$ represents the connection probability from branch $b(i)$ of node i to node j. For each branch in the graph structure, the probabilities to connect to the next node is calculated to represent the probabilistic model P_{nc}.

To calculate the probabilities of node connections, various approaches proposed in PMBGNP framework [4], [9] can be applied. In this paper, we use a Sarsa-Learning [23] based method called PMBGNP-RL [4] to construct P_{nc}. In this approach, each node connection is defined as a state-action pair[1] of RL. By factorizing individuals to the sequences of state-action pairs, we can generate and maintain state-action Q tables efficiently. These Q values can measure the qualities of their corresponding node connections, which are used to calculate the probabilities of node connections.

In the first generation, the population is initialized randomly. In each generation, the individuals are used to execute the given task and the executed node transitions are used to form the sequences of state-action pairs. M individuals will generate M corresponding Q tables. The state-action pairs can be obtained by factorizing the individuals, which can be substituted by the set of node connections in PMBGNP structure, respectively, as shown in Fig. 2-Middle.

After obtaining the state-action pairs, Sarsa-Learning is applied to update the Q value of each state-action pair. Suppose the state of individual n at time t is branch $b(i)$ of node i and its corresponding action is node j, which means the state-action pair can be formed by $(s_t, a_t) = (b(i), j)$. Meanwhile, the state-action pair of individual n at time $t+1$ is $(s_{t+1}, a_{t+1}) = (b(j), k)$. Then, the Q value of $(b(i), j)$ in Q table of individual n, i.e., Q_n, can be updated as follows:

$$Q_n(b(i),j) \leftarrow Q_n(b(i),j) + \alpha_s \Big[r_j + \gamma_s Q_n(b(j),k) - Q_n(b(i),j) \Big], \quad (1)$$

where,
α_s: learning rate of Sarsa-Learning.

[1]In this approach, 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 $(b(i), j)$.

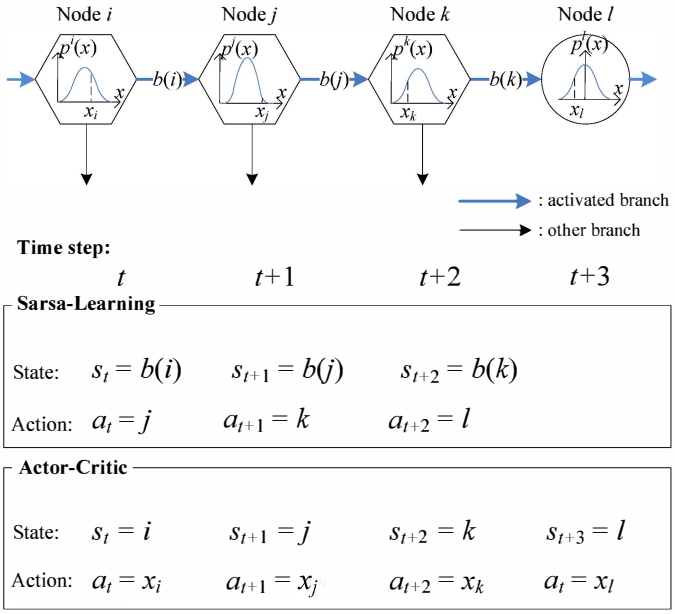


Fig. 2. (Top): Example of node transitions of PMBGNP-AC individuals; (Middle): Factorization of node transitions to state-action pairs in Sarsa-Learning of P_{nc}; (Bottom): Factorization of node transitions to state-action pairs in Actor-Critic of P_{cv}.

γ_s: discounted factor of Sarsa-Learning.
r_j: the reward of choosing node j at branch $b(i)$ of node i.

If node j is a processing node, then r_j is given after processing node j, which is application specific. Otherwise $r_j = 0$. The initial Q values can be either 0 or positive constants. In this paper, we initialize the Q values to 0. At the end of each generation, Eq. (1) is applied to update all the Q values.

Probabilistic model P_{nc} is built using the Q values obtained by Sarsa-Learning, in which $P_{nc}(i_{b(i)}, j)$ is calculated by:

$$P_{nc}(b(i),j) = \frac{\exp\left[\frac{\sum_{n \in N} Q_n(b(i),j)}{T}\right]}{Z(T)}, \quad (2)$$

where,
T: temperature parameter.
N: set of suffixes of promising individuals.

$Z(T)$ is the normalization function defined as follows:

$$Z(T) = \sum_{j' \in A(b(i))} \exp\left[\frac{\sum_{n \in N} Q_n(b(i),j')}{T}\right],$$

where,
$A(b(i))$: set of suffixes of nodes connected from branch $b(i)$.

The temperature T is defined in a non-uniform way, where T is decreased w.r.t. generations. In early generations, T is set at a large value to get more exploration of the search space, and decreases during evolution. A simple monotonically decreasing function is used to change the value of T: $T = GEN/g$, where GEN is the maximal number of generations and g is the current generation.

This approach has been confirmed to outperform a simple Maximum Likelihood Estimation based method [9] since Q values play roles as weights to quantify the importance of node connections [4].

2) Probabilistic modeling of continuous variables: Most attempts on extending EDA to continuous domains are to use Gaussian distribution [15], [16], [17]. The advantages of utilizing Gaussian distribution is its simplicity of implementation and without loss of generality for solutions which ensure the performance in continuous domains. Most of the current continuous EDAs are based on Gaussian distribution or related variants. PMBGNP-AC proposed in this paper is a straightforward extension of classical univariate continuous EDAs. That is, unidimensional Gaussian distribution is used for the distribution of continuous variables in each nodes of PMBGNP-AC.

Inspired by classical PBILc [15] and UMDAc [16], PMBGNP-AC optimizes Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ of each node. That is, the mean value μ and standard deviation σ of each continuous variable in each node are to be evolved. The probabilistic model P_{cv} of continuous variables in PMBGNP-AC consists of a set of probabilities $P^i_{cv}(x; \mu, \sigma)$, where $P^i_{cv}(x; \mu, \sigma)$ represents the probability density function (pdf) of the continuous variable of node i, described by:

$$P^i_{cv}(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right], \tag{3}$$

where,
x: continuous variables of node i.

In order to update the mean value μ and standard deviation σ of the continuous variable in each node, one may concern with the incremental learning [15] or Maximum Likelihood Estimation [16]. In this paper, we use a novel method to update the Gaussian distribution by AC. We first calculate the partial derivative of parameters μ and σ as follows:

$$\frac{\partial P^i_{cv}(x; \mu, \sigma)}{\partial \mu} = \underbrace{\frac{2}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]}_{>0} \cdot (x - \mu), \tag{4}$$

$$\frac{\partial P^i_{cv}(x; \mu, \sigma)}{\partial \sigma} = \underbrace{\frac{1}{\sqrt{2\pi}\sigma^2} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]}_{>0} \cdot \left[\frac{(x-\mu)^2}{\sigma^2} - 1\right] \tag{5}$$

As shown above, the front terms in the right side of Eq. (5) and (6) are always greater than 0. Therefore, inspired by the idea of gradient descent we can obtain the updating directions of μ and σ given a sampled value x of this Gaussian distribution as follows:

$$\nabla(\mu; x) = x - \mu, \tag{6}$$

$$\nabla(\sigma; x) = \frac{(x-\mu)^2}{\sigma^2} - 1. \tag{7}$$

These two equations are the simplified versions of Eq. (5) and (6). Then, we have the following updating rules of μ and σ:

$$\mu \leftarrow \mu + \alpha_\mu \nabla(\mu; x), \tag{8}$$

$$\sigma \leftarrow \sigma + \alpha_\sigma \nabla(\sigma; x). \tag{9}$$

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

Using the sampled values x from the promising individuals, Eq. (8) and (9) become similar to those of PBILc and UMDAc. Based on this foundation, we propose an extended version using a RL technique, i.e., Actor-Critic (AC) [23], [24].

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, we propose our algorithm PMBGNP with Actor-Critic (PMBGNP-AC) to update the Gaussian distribution. To incorporate AC, we define the state and action according to the structure of PMBGNP-AC as follows (Fig. 2-Bottom):

Definition 1 (State): State s is defined as a node in the directed graph of PMBGNP-AC.

Definition 2 (Action): Action a is defined as the selection of continuous variables in each node.

Therefore, the set of states refers to the set of nodes in the directed graph of PMBGNP-AC. Consequently, the set of actions is infinite and bounded according to the range of the continuous variables. Note that these two definitions are different from that of P_{nc} (see Fig. 2). With such definitions, we incorporate AC to our algorithm, where the Gaussian distribution is known as the *actor* since it is used to select actions (sample continuous variables), and we formulate the *critic* as state-value function 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:

$$\delta_t = r_t + \gamma_{ac} V(s_{t+1}) - V(s_t), \tag{10}$$

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:

$$V(s_t) \leftarrow V(s_t) + \alpha_{ac} \delta_t, \tag{11}$$

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

Algorithm 1 : PMBGNP-AC

1: $t \leftarrow 0$;
Initialize population $Pop(t)$ with the size of M and two probabilistic models P_{nc} and P_{cv};
2: Evaluate the fitness of $Pop(t)$;
3: Select a set of promising individuals $Best(t)$ in which ($|Best(t)| = N < M$);
4: **for** $i \in Best(t)$ **do**
5: Update the Q values according to Eq. (1);
6: Update the TD error δ and state-value function V according to Eq. (10) and (11), respectively;
7: Calculate P_{cv} by updating μ and σ according to Eq. (13) and (14), where $\nabla(\mu; x)$, $\nabla(\sigma; x)$, θ_t is obtained by Eq. (6), (7) and (12), respectively;
8: **end for**
9: Calculate P_{nc} according to Eq. (2);
10: Generate $P(t+1)$ by sampling P_{nc} and P_{cv};
$t \leftarrow t + 1$;
11: Go back to step 2 until the terminal criteria is met.

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.

$$\theta_t = \begin{cases} -1, & \text{for } \delta_t < 0 \\ 0, & \text{for } \delta_t = 0 \\ 1, & \text{for } \delta_t > 0 \end{cases} \tag{12}$$

Inserting this scalar reinforcement signal into Eq. (8) and (9), we get the final updating rules of the Gaussian distribution of PMBGNP-AC as follows:

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

$$\sigma \leftarrow \sigma + \alpha_\sigma \nabla(\sigma; x)\theta_t. \tag{14}$$

We can observe that by inserting AC, we can not only model the univariate interactions by Gaussian distribution explicitly, but also model the multivariate interactions implicitly, since the reinforcement signal θ_t is formulated by considering the future state-values, i.e., $V(s_{t+1})$.

C. Algorithm of PMBGNP-AC

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 shown in Algorithm 1. As a result, the final probability of generating individual n by PMBGNP-AC is:

$$P(n) = \prod_{i \in N_{\text{node}}} \left[P^i_{cv}(x_i; \mu_i, \sigma_i) \prod_{b(i) \in B(i)} P_{nc}(b(i), j) \right]. \tag{15}$$

where,
N_{node}: set of suffixes of nodes in one individual.
$B(i)$: set of suffixes of branches in node i.

The initial Q values and state-values V are prepared in advance, which are set at 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.

III. Simulations

Different from most of conventional EDAs doing function optimization problems, PMBGNP-AC is applied to controlling the behaviors of an autonomous robot – Khepera robot [25], which can be classified to a kind of Reinforcement Learning (RL) problems [23].

Khepera robot (Fig. 3) [25] is a wheeled mobile robot with 8 distance sensors allowing the robot to detect the proximity of objects around it by reflection. Each sensor returns a continuous value ranging from 0 to 1023 (0 means that there is no object around the sensor, while 1023 means that an object is very close to the sensor). Two motors corresponding to the left and right wheel can take speed values ranging from -10 to +10, where different combinations of the two speeds would control the robots for different moving behaviors.

The following algorithms are selected for comparisons:
1) *Standard GNP*: both of node connections and continuous variables of nodes are evolved by crossover and mutation.
2) *PMBGNP* [4]: discretization is used to transfer the continuous variables into discrete values, where the value of each node remains unchanged during evolution. In this case, only P_{nc} is constructed to evolve the node connections by Sarsa-Learning.
3) *PBILc*: μ and σ of continuous PMBGNP are evolved like those of PBILc [15].
4) *Sarsa-Learning* (*Sarsa*) [23]: A classical RL metod, Sarsa-Learning in which the continuous state and action space is defined as the discrete space. ϵ-greedy policy is used for action selection.
5) *PMBGNP-AC*: the proposed algorithm.

The experiments are done by controlling Khepera robot to solve the wall-following problem. The reward and fitness functions are designed based on [27] and shown as follows

$$\text{Reward} = \frac{v_R + v_L}{20}\left(1 - \sqrt{\frac{|v_R - v_L|}{20}}\right)C, \tag{16}$$

$$\text{Fitness} = \frac{\sum_{step=1}^{S_{max}} \text{Reward}}{S_{max}}, \tag{17}$$

where,
v_R, v_L: the speed of right and left wheels,
S_{max}: the user predefined steps controlling the robot's running time, which is equivalent to the problem size.

$$C = \begin{cases} 1, & \text{all the sensor values are less than 1000,} \\ & \text{and at least one of them is more than 100,} \\ 0, & \text{otherwise.} \end{cases}$$

Figure 3 shows the environment used in this paper, where the size of the simulated environment is 1 m × 1 m. The aim of the fitness evaluation is to control the robot to move following the wall as fast as and as straight as possible until the predefined steps S_{max} reaches. S_{max} is set at {100, 200, 300, 400, 500} to define the problems from simple cases to complex ones.

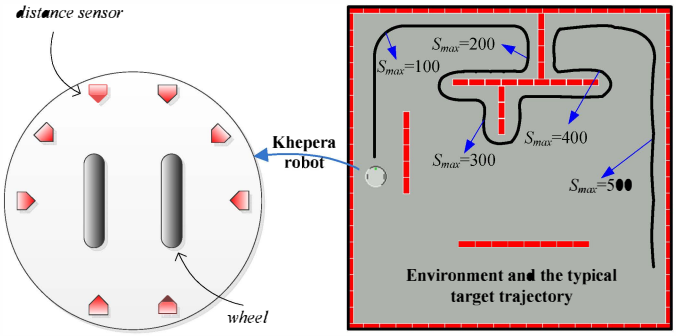


Fig. 3. Khepera robot and simulation environment.

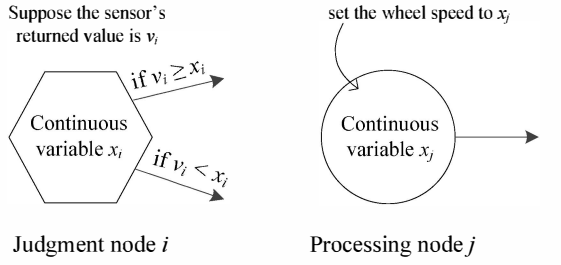


Fig. 4. Roles of continuous variables in judgment/processing nodes.

A. Node functions and continuous variables

The node functions are shown in Table I. There are total 8 judgment functions for judging the sensors' values of the robot. In this paper, the number of branches of each judgment node is set at 2. 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 divides the domain of its sensor value into two intervals ([0, x_i) and [x_i, 1023]), where the selection of branches is determined by the comparison of real returned value and x_i.

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

These two kinds of continuous variables are to be evolved to determine the final solutions, as an example shown in Fig. 4. Meanwhile, in standard PMBGNP, these two types of variables are discretized in advance. For judgment nodes, the domain is divided into two fixed intervals, [0, 1000) and [1000, 1023]. In processing nodes, the domain is discretized into the set of {-10, -5, 0, 5, 10} to determine the robot's speed.

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. Therefore, there are total $2^8 = 256$ states and $5 \times 5 = 25$ actions in Sarsa.

B. Parameter settings

The time delay of judgment nodes is set at 1 time unit, that of node transition is set at 0 time unit and that of processing nodes is set at 5 time units. The robot will take one step of actions if 5 time units or more are reached. In each step, GNP judges the sensor values and determines the speed of the wheels to control the movement of the robot. The simulation ends when the step exceeds S_{max}.

The number of judgment nodes for each judgment function is set at 5. The number of processing nodes for each processing function is set at 10. Therefore, the program size[2] of each individual is $5 \times 8 + 10 \times 2 = 60$.

The simulation conditions are defined as shown in Table II. All these settings are the appropriate ones defined by hand-tuning.

TABLE I
NODE FUNCTIONS USED FOR KHEPERA ROBOT.

Node	*NF*	Function	Domain
J_1, J_2, ..., 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]

TABLE II
SIMULATION CONDITIONS.

	GNP	PMBGNP	PBILc	PMBGNP-AC
Population size M	1000	1000	1000	1000
– elite ind.	100	100	100	100
– crossover ind.	400	–	–	–
– mutation ind.	500	–	–	–
– promising ind. N	–	500	500	500
Program size $\lvert N_{node} \rvert$	60	60	60	60
Crossover rate	0.1	–	–	–
Mutation rate	0.01	–	–	–
Other parameters	(— Sarsa-Learning) $\alpha_s = 0.1$, $\gamma_s = 0.9$			
	(— Actor-Critic) $\alpha_{ac} = 0.1$, $\gamma_{ac} = 0.9$			
	$\alpha_\mu = 0.05$, $\alpha_\sigma = 0.05$			
	(— ϵ-greedy policy) $\epsilon = 0.1$			
Terminal condition	300,000 fitness evaluations			

C. Simulation results and analysis

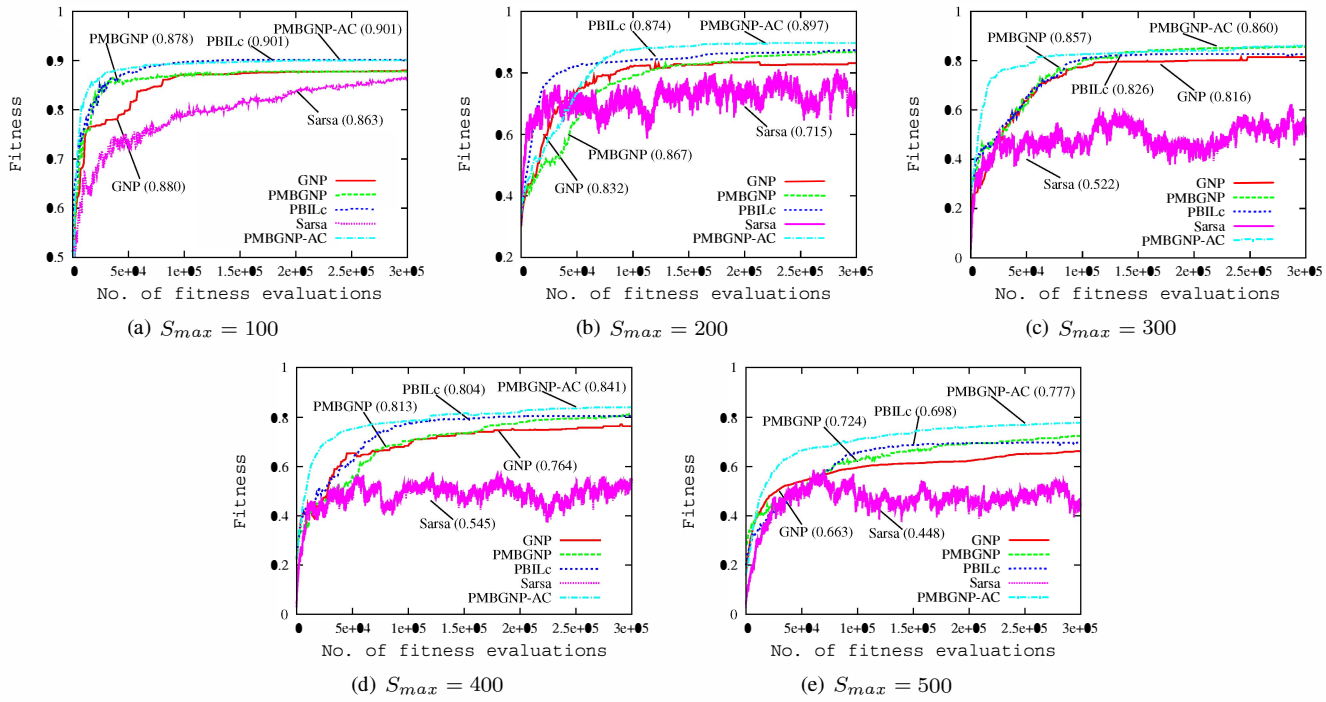
Five simulations of $S_{max} \in \{100, 200, 300, 400, 500\}$ are carried out to testify the effectiveness and scalability of the proposed algorithm. The simulation results are the average over 30 independent runs.

1) Fitness results: The detailed fitness values and curves of the compared methods are shown in Table III and Fig. 5. Each value of Table III indicates the average fitness with standard deviation, where the bold ones denote the best results of the problems. We perform the analysis of different methods as follows:

GNP: In most cases of the five problems, GNP performs worse results than the variants of PMBGNP, and only outperforms Sarsa. This is due to the lack of evolution ability by standard genetic operators, i.e., crossover and mutation. The results show that such genetic operators cause GNP achieve worse evolution ability than that of EDA based PMBGNP.

PMBGNP: PMBGNP achieves better performance than that of Sarsa and GNP. However, it has worse performance than PMBGNP-AC. This is due to the loss of the solution precision by discretization. On the other hand, PMBGNP actually obtains very stable performances among five problems (has the

[2]Start node is not taken into account in this case.



(a) $S_{max} = 100$ (b) $S_{max} = 200$ (c) $S_{max} = 300$

(d) $S_{max} = 400$ (e) $S_{max} = 500$

Fig. 5. Fitness curves in five wall-following problems.

TABLE III
THE FITNESS RESULTS OVER 30 INDEPENDENT RUNS.

	Fitness (std. dev.)				
S_{max}	100	200	300	400	500
GNP	**0.880** ± **0.019**	**0.832** ± **0.111**	**0.815** ± **0.091**	**0.764** ± **0.052**	**0.663** ± **0.147**
PMBGNP	**0.878** ± **0.022**	**0.867** ± **0.054**	**0.857** ± **0.050**	**0.813** ± **0.094**	**0.724** ± **0.085**
PBILc	**0.901** ± **0.009**	**0.874** ± **0.092**	**0.826** ± **0.085**	**0.804** ± **0.148**	**0.697** ± **0.116**
Sarsa	**0.865** ± **0.035**	**0.831** ± **0.111**	**0.522** ± **0.207**	**0.545** ± **0.187**	**0.448** ± **0.222**
PMBGNP-AC	**0.901** ± **0.016**	**0.897** ± **0.032**	**0.860** ± **0.098**	**0.841** ± **0.144**	**0.777** ± **0.108**

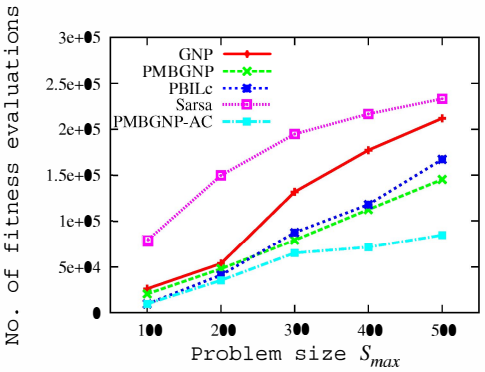


Fig. 6. Average fitness evaluation for five wall-following problems.

smallest standard deviation), which is because of the balance of exploitation and exploration by Boltzmann distribution [4].

PBILc: In this method, only the information of three individuals (the best two and the worst one) are used to update the pdf of Gaussian distribution [15]. It might cause that the search space is explored in a too restricted region and slow speed. Therefore, this method can ensure quite good performance in simple problems ($S_{max} = 100, 200$), however, obtain poor results when the problem size increases ($S_{max} = 500$).

Sarsa: Discretization causes that Sarsa has to update a huge size of Q table (consists of 256×25 state-action pairs in each time step). This might cause the slow learning speed of Sarsa. Therefore, Sarsa can only work well in simple problems among five simulation results. However, with the increase of the problem size, it cannot find the optimal solution.

PMBGNP-AC: The simulation results confirm the effectiveness and scalability of PMBGNP-AC. It ensures the best results among all the five problems. Although in a certain respect it is more volatile than PMBGNP (larger standard deviation), we can still say that PMBGNP-AC works quite well in directly handling the continuous variables.

Fig. 6 plots the average number of fitness evaluations to solve the wall-following problems. Results confirm the higher evolution ability of PMBGNP-AC over the other methods.

2) Statistical analysis: In order to further analyze the simulation results, we display the statistical analysis of different methods by t-test (one-tailed, paired). Table IV details the t-test results. This shows that PMBGNP-AC statistically outperforms GNP and Sarsa among all the problems, and outperforms PMBGNP and PBILc in some problems. Particularly, in the case of the largest problem size ($S_{max} = 500$), PMBGNP-AC is statistically superior than the other methods.

IV. CONCLUSIONS AND FUTURE WORK

The main originality of PMBGNP is to extend EDA from string/tree structures to graph structures. At this level, it can solve a wide range of problems differing from conventional EDAs. This paper extends PMBGNP from the discrete

TABLE IV
THE T-TEST RESULTS OF PMBGNP-AC AND THE OTHER METHODS (THE BOLD ONES DENOTE STATISTICALLY SIGNIFICANT DIFFERENCE).

	t-test (p value)				
S_{max}	100	200	300	400	500
PMBGNP-AC vs. GNP	$\mathbf{2.273 \times 10^{-5}}$	$\mathbf{2.630 \times 10^{-3}}$	$\mathbf{1.451 \times 10^{-2}}$	$\mathbf{1.417 \times 10^{-2}}$	$\mathbf{1.218 \times 10^{-3}}$
PMBGNP-AC vs. PMBGNP	$\mathbf{8.665 \times 10^{-5}}$	$\mathbf{3.609 \times 10^{-3}}$	4.342×10^{-1}	2.126×10^{-1}	$\mathbf{2.523 \times 10^{-2}}$
PMBGNP-AC vs. PBILc	3.855×10^{-1}	1.098×10^{-1}	9.505×10^{-2}	1.834×10^{-1}	$\mathbf{1.225 \times 10^{-3}}$
PMBGNP-AC vs. Sarsa	$\mathbf{1.711 \times 10^{-6}}$	$\mathbf{3.560 \times 10^{-6}}$	$\mathbf{1.930 \times 10^{-8}}$	$\mathbf{3.580 \times 10^{-8}}$	$\mathbf{2.460 \times 10^{-8}}$

domain to continuous cases. We 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 the conventional methods, including conventional genetic operators based EA, discretization based EDA, PBILc based variant and classical RL method. 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 or some related probabilistic graphical models, and little attention has been deserved to RL techniques even though it is an important branch of ML. The proposed PMBGNP-AC in this paper uses the techniques of RL to build its probabilistic models and shows better performance than classical RL method, which provides an attempt to bridge the gap between EDA and RL.

As a preliminary study of this topic, the following further research will be considered: 1) Further empirical and theoretical research will be done to study PMBGNP-AC comprehensively. 2) Results show that PMBGNP-AC achieves more volatile performance than that of PMBGNP. This might be due to the loss of population diversity, which has been addressed in many research on continuous EDAs. Further research will be done to solve this problem by considering various existing techniques, such as clustering and niching. 3) AC can be easily combined with eligibility traces. This may speed up the learning efficiency which will be studied in the future.

REFERENCES

[1] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, 2002.

[2] M. Pelikan, D. E. Goldberg and F. G. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, Kluwer Academic Publishers, Vol.21, pp. 5-20, 2002.

[3] Y. Shan, R. I. McKay, D. Essam and H. A. Abbass, "A survey of probabilistic model building genetic programming," In M. Pelikan, K. Sastry and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, pp. 121-160, 2006.

[4] X. Li, B. Li, S. Mabu and K. Hirasawa, "A novel estimation of distributon algorithm using graph-based chromosome representation and reinforcement learning," *In Proc. of the IEEE Congress on Evol. Comput.*, pp. 37-44, 2011.

[5] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive leaning," Tech. Report. No. CMU-CS-94-163, Carnegie Mellon University, 1994.

[6] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions I. Binary parameters," *In Proc. of the 4th Conf. on Parallel Problem Solving from Nature*, pp. 178-187, 1996.

[7] M. Pelikan, D. E. Goldberg and E. Cantu-Paz, "Linkage problem, Distribution estimation, and Bayesian networks," *Evol. Comput.*, Vol. 8, No. 3, pp. 311-341, 2002.

[8] R. P. Salustowicz and J. Schmidhuber, "Probabilistic incremental program evolution," *Evol. Comput.*, Vol. 5, No. 2, pp. 123-141, 1997.

[9] X. Li, S. Mabu, H. Zhou, K. Shimada and 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.*, pp. 2673-2680, 2010.

[10] X. Li, S. Mabu, H. Zhou, K. Shimada and K. Hirasawa, "Genetic network programming with estimation of distribution algorithms for class association rule mining in traffic prediction," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 14, No. 5, pp. 497-509, 2010.

[11] K. Hirasawa, M. Okubo, H. Katagiri, J. Hu and J. Murata, "Comparison between genetic network programming (GNP) and genetic programming (GP)," *In Proc. of the IEEE Congress on Evol. Comput.*, pp. 1276-1282, 2001.

[12] H. Katagiri, K. Hirasawa and J. Hu, "Genetic network programming -Application to intelligent agents," *In Proc. of the IEEE Int'l Conf. on Systems, Man and Cybernetics*, pp. 3829-3834, 2000.

[13] S. Mabu, K. Hirasawa and J. Hu, "A graph-based evolutionary algorithm: Genetic network programming (GNP) and its extension using reinforcement learning," *Evol. Comput.*, Vol.15, No.3, pp. 369-398, 2007.

[14] X. Li, S. Mabu and K. Hirasawa, "Use of infeasible individuals in probabilistic model building genetic network programming," *In Proc. of the Genetic and Evol. Comput. Conf.*, pp. 601-608, 2011.

[15] M. Sebag and A. Ducoulombier, "Extending population-based incremental learning to continuous search spaces," *In Proc. of the 5th Conf. on Parallel Problem Solving from Nature*, pp. 418-427, 1998.

[16] P. Larrañaga, R. Etxeberria, J. A. Lozano and 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.

[17] P. A. N. Bosman and D. Thierens, "Numerical optimization with real-valued estimation-of-distribution algorithms," In M. Pelikan, K. Sastry and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, pp. 91-120, 2006.

[18] P. A. N. Bosman and D. Thierens, "Expanding from discrete to continuous estimation of distribution algorithms: The IDEA," *In Proc. of the 6th Conf. on Parallel Problem Solving from Nature*, pp. 767-776, 2000.

[19] P. A. N. Bosman and D. Thierens, "Mixed IDEAs," Utrecht University technical report UUCS- 2000-45, 2000.

[20] S. Tsutsui, M. Pelikan and D. E. Goldberg, "Probabilistic model-building genetic algorithms using marginal histograms in continuous domain," *In Proc. of the KES'2001*, pp. 112-121, 2001.

[21] Q. Lu and X. Yao, "Clustering and learning Gaussian distribution for continuous optimization," *IEEE Trans. on Syst. Man & Cyber. - C*, Vol. 35, No. 2, pp. 195-204, 2005.

[22] W. Dong and X. Yao, "NichingEDA: Utilizing the diversity inside a population of EDAs for continuous optimization," *In Proc. of the IEEE Congress on Evol. Comput.*, pp. 1260-1267, 2008.

[23] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[24] S. Mabu, Y. Chen, K. Hirasawa and J. Hu, "Stock trading rules using genetic network programming with actor-critic," *In Proc. of the IEEE Congress on Evol. Comput.*, pp. 508-515, 2007.

[25] Cyberbotics Corp., "Webots software", http://www.cyberbotics.com/.

[26] T. Eguchi, K. Hirasawa, J. Hu and N. Ota, "A Study of Evolutionary Multiagent Models Based on Symbiosis," *IEEE Trans. on Syst. Man & Cyber. - B*, Vol.36, No.1, pp. 179-193, 2006.

[27] P. Nordin, W. Banzhaf and M. Brameier, "Evolution of a world model for a miniature robot using genetic programming," *Robotics and Autonomous Systems*, Vol. 25, pp. 105-116, 1998.