A Novel Estimation of Distribution Algorithm Using Graph-based Chromosome Representation and Reinforcement Learning

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

***Abstract*—This paper proposed a novel EDA, where a directed graph network is used to represent its chromosome. In the proposed algorithm, a probabilistic model is constructed from the promising individuals of the current generation using reinforcement learning, and used to produce the new population. The node connection probability is studied to develop the probabilistic model, therefore pairwise interactions can be demonstrated to identify and recombine building blocks in the proposed algorithm. The proposed algorithm is applied to a problem of agent control, i.e., autonomous robot control. The experimental results show the superiority of the proposed algorithm comparing with the conventional algorithms.**

I. Introduction

Numerous algorithms have been proposed to draw the success of Estimation of Distribution Algorithms (EDA), in both theory and application [1], [2]. Different from conventional EA, EDA explicitly models the good individuals found so far and uses the probabilistic model to guide the further search. From the perspective of chromosome representation, there are two main categories of EDA, which are Probabilistic Model Building Genetic Algorithm (PMBGA) [3], [4], [5] and Probabilistic Model Building Genetic Programming (PMBGP) [6], [7]. PMBGA uses GA's binary string structure to represent its chromosome, while PMBGP uses GP's tree structure to represent its chromosome. However, there is few work on extending EDA to the graph-based EAs.

Some previous research has shown the superiority of graph-based EAs in terms of stronger expression and evolution ability than that of conventional GP [8], [9], [10]. Genetic Network Programming (GNP) [10], [11] is one of the graph-based EAs, which extends GA and GP by means of using a directed graph network to represent its chromosome. Different from the other graph-based EAs, GNP is firstly designed for solving the agent control problems, where many studies have also investigated that it can outperform the conventional algorithms in many other problems, such as Multi-agent Systems [12], [13], [14], Data Mining [15] and Elevator System Control [16], etc. A detailed comparison among GNP, conventional EAs and the other graph-based EAs can be found in [13].

The purpose of this paper is to propose a new EDA algorithm that uses GNP's directed graph structure to represent its chromosome. Although EDA has been applied to GNP in the previous research [17], [18], the method only models the building blocks by calculating the marginal probabilities of node connections using simple maximum likelihood estimation, which is similar to many other classical EDAs, like UMDA and PIPE. This paper proposes a new algorithm, where reinforcement learning is designed to estimate a more accurate distribution based on the interactions between agents and environments. Therefore, the problems can be solved more efficiently and reliably. The proposed algorithm is applied to control a kind of autonomous robots, Khepera robot [19].

Algorithm 1 Algorithm of EDA

1: $t \leftarrow 0$
randomly generate an initial population $S(t)$
2: evaluate the fitness of the initial population
3: execute selection operator to select a set of promising individuals $B(t)$
4: construct a probabilistic model P^t from $B(t)$
5: generate a new population $S(t+1)$ using the probabilistic model P^t
6: evaluate the fitness of the new population $S(t+1)$
set $t \leftarrow t+1$
7: go back to 3 until the terminal condition is reached

The paper is organized as follows: Section II briefly introduces the previous work on EDA and compares our work with many others. Section III roughly describes RL. Section IV presents the proposed algorithm in details. The experimental study is carried out in section V. Section VI presents the conclusions and future work.

II. Background

A. Related work on GA and GP based EDA

Algorithm 1 shows the basic pseudo-code of EDA. The idea of EDA was first introduced in the field of binary GA, and EDA has attracted much attention in the last decade. The class of EDA using GA's chromosome representation is called PMBGA. It mainly consists of three types of algorithms: no interactions, pairwise interactions and multivariate interactions, to study the building blocks of different complexities. The details of PMBGA can refer to a survey in [2].

Later, some research has extended EDA to GP, which uses tree structures to represent its chromosome. The algorithms in

978-1-4244-7835-4/11/$26.00 ©2011 IEEE

this topic are generally called PMBGP. PIPE [6] is the first approach of PMBGP, where a prototype tree is constructed and maintained to evolve GP individuals. PIPE corresponds to PMBGA without interactions, such as PBIL [3] and UMDA [4]. EDP [20] extends PMBGP to pairwise interactions, where Bayesian network is used to estimate the interactions between parent and child nodes. eCGP [21] and POLE [22] extend PMBGP to multivariate interactions where more complex building blocks in GP could be identified and recombined. These approaches are generally called prototype tree-based PMBGP, which are direct expansion of PMBGA, where the probability distribution over the chromosome structures is estimated. Another group of PMBGP is called grammar model-based PMBGP, where the probability distribution of grammars is estimated. A detailed survey of PMBGP can be found in [7].

The effectiveness of EDA has been proven in numerous problems, such as function optimization problems by PMBGA and symbolic regression problems by PMBGP, etc.

B. Overview

The work described in this paper is based on extending EDA to a graph-based EA, i.e., GNP. Although EDA has been extended to GNP to form a new algorithm named Probabilistic Model Building GNP (PMBGNP) [17], [18], it is an early attempt that uses marginal probabilities to estimate the interactions between the adjacent nodes, where the probabilistic model is mainly constructed by calculating the frequencies of node connections in the promising individuals. The effectiveness of PMBGNP has been proven in a real-world application for finding class association rules from traffic databases [17], [18].

This paper proposed an extended algorithm, PMBGNP with reinforcement learning (PMBGNP-RL), where a reinforcement learning (RL) technique named Sarsa-learning [23], [24] is applied to evaluate the importance of node connections based on the interactions between agents and environments. Therefore, the proposed algorithm can estimate more accurate distribution than conventional PMBGNP that only studies the frequencies of the structures of promising individuals.

The previous work on GNP has proven that Sarsa-learning can be easily applied to GNP framework, where a variant named GNP-RL [13] was proposed. GNP-RL tends to change the structure of GNP by introducing sub-nodes in each node and create one Q table for each individual. The states and actions in GNP-RL are defined by the introduced sub-nodes. However, the proposed algorithm PMBGNP-RL contributes in a different way. PMBGNP-RL also creates one Q table for each individual which is the same as GNP-RL, but it does not change the structure of GNP and uses another way to define the states and actions. The states and actions are defined by the branches and nodes of GNP rather than the sub-nodes in GNP-RL. Such kind of definitions has been proven to work well in GNP [25], where an Sarsa-learning based non-uniform mutation was proposed. Meanwhile, the calculated Q values of PMBGNP-RL are used to construct the probabilistic model, which is quite different from both of [13] and [25].

Nevertheless, most of the current EDA are designed for solving some benchmark problems of GA and GP. Different from them, the proposed algorithm is designed for evolving the behavior sequences of agents. In certain respects, the work of this paper is similar to a recent work called EDA-RL in [26]. Both of EDA-RL and the proposed algorithm are capable of handling the agent control problems. EDA-RL is a totally new EDA that estimates the policies of agents by Conditional Random Fields (CRFs) directly. It has its own chromosome representation by using the behavior sequences of agents (episodes). EDA-RL evolves the policies of agents from uniform distribution to optimal one. Different from EDA-RL, the proposed algorithm in this paper uses the directed graph of GNP to form its chromosome, where Sarsa-learning is used as a scheme to study the structures of individuals based on the interactions between agents and environments.

III. Reinforcement Learning

Reinforcement Learning (RL) [23] is a powerful technique to study the interactions between agents and environments. The aim of RL is to find a policy $\pi(s, a)$ which maximizes the expected future discounted sum of rewards received, where the policy $\pi(s, a)$ is generally the sequences of state-action pairs. The optimal policy is approximated by updating state-action values (Q values) $Q(s, a)$ w.r.t. the return of the next state visited.

Various RL algorithms have been proposed to find the optimal policy by using different forms of update rules. Among these algorithms, Sarsa-learning [24] is an on-policy approach, where the agent will interact with the environment and update the policy based on actions taken. At time step t, the main function for updating the Q values depends on the current state s_t, action a_t of the agent, reward r_t the agent gets, the next state s_{t+1} and the next action a_{t+1} of the agent, as shown in the following

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right), \quad (1)$$

where, α ($0 < \alpha \leq 1$) is the learning rate, and γ ($0 \leq \gamma < 1$) represents the discount factor.

Different from the off-policy of Q-learning [27], Sarsa-learning learns the Q values associated with the true actions the agent takes, rather than the maximal reward of total available actions in Q-learning. Both of Sarsa-learning and Q-learning have been proven to converge to the optimal solution [23]. However, Sarsa-learning can work efficiently in large state-spaces [24]. Due to such discussion, in this paper, Sarsa-learning is applied to PMBGNP to estimate a more accurate probability distribution formed by the Q values calculated based on the interactions between agents and environments.

IV. PMBGNP-RL

A. Outline of PMBGNP-RL

Similar to most of the current EDA, the proposed algorithm estimates the distribution of promising individuals to generate new individuals as shown in Algorithm 1. The main points that

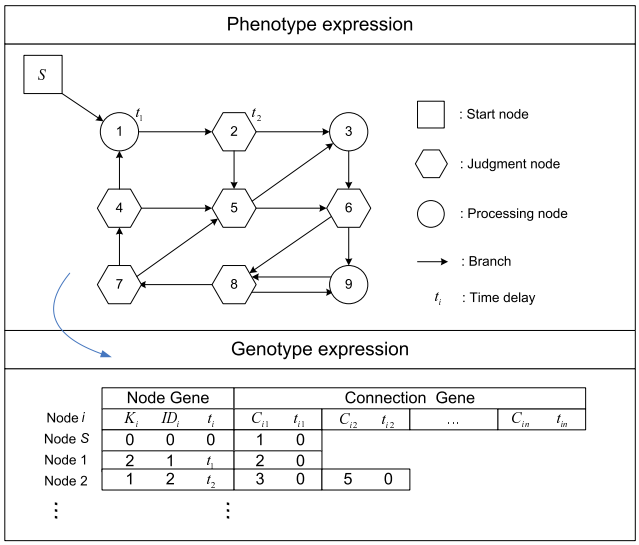


Fig. 1. The directed graph structure.

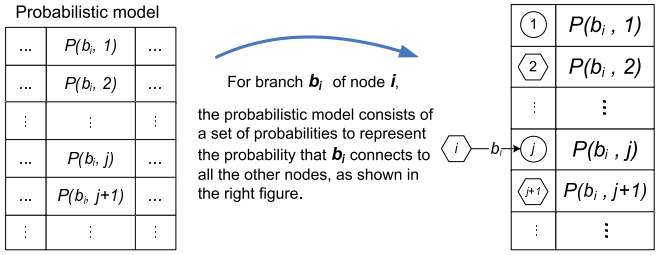


Fig. 2. An example of the probabilistic model.

make this work unique are: firstly, PMBGNP-RL is a graph-based EDA, where the directed graph structure of GNP is used to represented its chromosome; Secondly, RL is carried out to estimate the distribution of promising individuals, where the interactions between agents and environments are taken into account to form a more accurate probabilistic model. This section will focus on describing two main issues of PMBGNP-RL, which are graph-based chromosome representation and probabilistic model construction.

B. Directed graph structure

PMBGNP-RL uses the directed graph structure of GNP to represent its chromosome, which can be illustrated by the phenotype and genotype expression. Phenotype shows the directed graph structure, while genotype demonstrates the encoding of GNP. As shown in Fig. 1, let i represent a node number of GNP. K_i defines the type of node i such as start node, judgment node and processing node. ID_i identifies the node function, such as judgment functions and processing functions. C_{in} denotes the node which is directly connected from branch n of node i. t_i and t_{in} are the delay time, which are the time required to execute node i and time to transit from node i to node C_{in}, respectively.

Generally, one start node, a fixed number of judgment nodes and processing nodes composes the structure of GNP. The start node is only used to decide the first node to be transited, while the judgment and processing nodes have some functions corresponding to the concrete problem. Each judgment node works as "if-then" type decision making functions to judge the environment and to make a decision, while processing nodes preserve the action functions to determine the agent's action. For example, in this paper, the judgement nodes judge the sensor values of the Khepera robot, while processing nodes determine the robot's wheel speeds. By separating judgment and processing functions, GNP can handle various combinations of judgments and processings. That is, the evolution can efficiently evolve the compact programs by selecting the necessary judgements and processings to control agents.

In this structure, each judgment node consists of multiple branches connecting to different nodes, where the next node to be transited is determined by the judging result of the environment. Processing node only has one branch, since the processing functions only determine the agent's actions. Therefore, the agent will be controlled by transiting the nodes until the task is solved. Generally, when solving concrete problems, the number of judgment and processing nodes, the number of branches in judgment nodes and time delays are predefined. In this paper, the branches of judgment nodes is set at 2 to efficiently implement the IFLTE(a, b, c, d)[1] function in this paper.

C. Probabilistic model construction

The model construction of PMBGNP-RL is inspired by most of the classical EDA, which directly studies its graph-based chromosome structure. However, one should note that the traditional PMBGA estimates the probabilistic model by the probabilities of 0's or 1's in each gene loci, while PMBGP by the probabilities of functions in each node. On the other hand, the proposed algorithm estimates the probabilities of connections between the nodes. It is due to the characteristics of GNP, that is, the number of nodes and the functions of each node are predefined and fixed in GNP individuals to evolve compact programs. Moreover, the start node is not considered in our probabilistic model since its connection to the next node is predefined and fixed. The probabilistic model P of PMBGNP-RL is composed of a set of probabilities $P(b_i, j)$, where $P(b_i, j)$ represents the connection probability from branch b_i of node i to node j. Fig. 2 shows an example of the probabilistic model. For each branch in the graph structure, the probabilities to connect to the next node is calculated to represent the probabilistic model.

1) Definitions: Before applying RL in the proposed algorithm, we need to give the following definitions w.r.t. RL

Definition 1: Episode The task is executed by following the sequences of the node transitions in PMBGNP-RL individuals, which can be viewed as the episodes of RL. Each PMBGNP-RL individual generates one episode of RL.

Definition 2: State A state is defined as a branch of a node in PMBGNP-RL individual. Therefore, the set of states refers

[1]IFLTE(a, b, c, d) function means that if ($a < b$) then c else d, which is widely used for robot control.

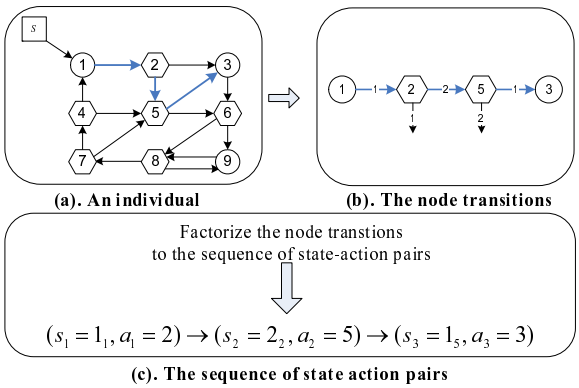


Fig. 3. An example of PMBGNP-RL runs.

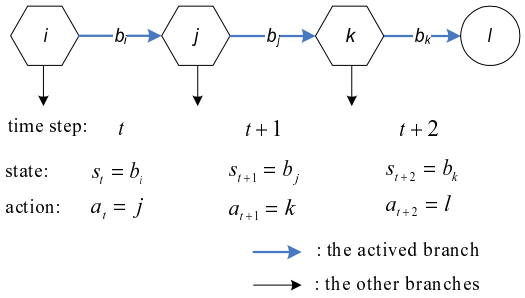


Fig. 4. Episode generated by the node transitions of PMBGNP-RL.

to the set of branches of PMBGNP-RL individual, which is predefined and fixed.

Definition 3: Action An action is defined as a node in PMBGNP-RL individual.

Fig. 3 shows an example of PMBGNP-RL runs. The agent of each individual is controlled by following the node transitions of PMBGNP-RL to solve one task, i.e., node $1 \rightarrow 2 \rightarrow 5 \rightarrow 3$. The branch of each node is selected based on its function, i.e., node 2 selects its second branch based on IFLTE(a, b, c, d) function. The agent observes its environment to decide an action, i.e., to select a node to transit. Concretely speaking, an activated branch corresponds to the current state, and the selection of the next node in the branch corresponds to an action. Since the number of branches and nodes are predefined, fixed and not so large in GNP structure, PMBGNP-RL can generate and maintain the state-action Q tables efficiently.

2) Sarsa-Learning: Fig. 4 shows the general form of an episode generated by the node transitions of PMBGNP-RL and the factorized state-action pairs in each time step.

In the first generation, the episodes are initialized randomly since the population of PMBGNP-RL is generated randomly. In each generation, the individuals are used to execute the given task and the executed node transitions are used to form the episodes. Therefore, M individuals will generate M episodes. Each episode generates one corresponding Q table, therefore total M Q tables are generated. The state-action pairs can be obtained by factorizing the episodes, which can be represented as:

$$(\mathcal{S}, \mathcal{A})_n = \{(s_1, a_1)_n, (s_2, a_2)_n, ..., (s_L, a_L)_n\}, \quad (2)$$

where,
L: the length of episode n.

The states $\mathcal{S}$ and corresponding actions $\mathcal{A}$ can be substituted by the set of branches and the set of nodes in PMBGNP-RL structure, respectively, as shown in Fig. 4.

After the factorization of the episodes, Sarsa-learning is applied to update the Q value of each state-action pair. Suppose the state of episode 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 episode 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 episode n, i.e., Q_n, can be updated as follows:

$$Q_n(b_i, j) \leftarrow Q_n(b_i, j) + \alpha \left(r_j + \gamma Q_n(b_j, k) - Q_n(b_i, j)\right), \quad (3)$$

where,
r_j: the reward of choosing node j at branch b_i.

The reward can be defined flexibly based on different problems. In this paper, we use the following rule to define the reward r_j:

1) If node j is a processing node, then r_j is given after processing node j, which is application specific.
2) If node j is a judgment node, then $r_j = 0$.

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. (3) is applied to update all the Q values.

3) Calculation of probabilistic model: The probabilistic model is built using the Q values calculated by Sarsa-learning, where the probability $P(b_i, j)$ that branch b_i of node i is connected to node j is calculated. Therefore, the probabilistic model is constructed by:

$$P(b_i, j) = \frac{\sum_{n \in N} Q_n(b_i, j)}{\sum_{j' \in A(b_i)} \sum_{n \in N} Q_n(b_i, j')}, \quad (4)$$

where,
N: set of suffixes of promising individuals.
$A(b_i)$: set of suffixes of nodes connected from branch b_i.

We can easily find that the probabilistic model can model the pairwise interactions explicitly, since it estimates the combinations between two nodes. However, the model can also represent the multivariate interactions implicitly, because Sarsa-learning updates the current Q values based on the future information as shown in Eq. (3).

In standard PMBGNP, when we observe a state-action pair, i.e., (b_i, j), we just simply give it a weight 1. However, in the proposed algorithm, a weight $Q(b_i, j)$ is given to the observed state-action pair (b_i, j), which is calculated by Eq. (3). On the other hand, in each generation, the increased computational time of the proposed algorithm compared with that of PMBGNP is only a constant w.r.t. the size of state-action space.

4) Boltzmann distribution: One important issue can be observed easily in the study of PMBGNP, that is, PMBGNP is generally much more sensitive to the population diversity than that of PMBGA and PMBGP. In the framework of PMBGNP, suppose the total number of nodes in the chromosome structure is $|N_{node}|$, then branch b_i has $|N_{node}|-1$ probabilities. On the other hand, in simple binary PMBGA, each gene loci has only 2 probabilities, i.e., $P(0)$ and $P(1)$. Similarly, in simple PMBGP, such as PIPE, the number of probabilities is decided based on the size of the function and terminal set, which are also generally small. Therefore, the required sample size in PMBGNP is generally larger than that of PMBGA and PMBGP in order to estimate an accurate distribution. Generally speaking, it is not a good way to simply increase the sample size, because the population diversity will be lost quickly during the evolution even with the large sample size. An alternative way should be studied to avoid the premature convergence of the model.

In this paper, we further introduce Boltzmann distribution to the proposed algorithm in order to control the convergence of the model. Therefore, Eq. (4) is rewritten as follows

$$P(b_i,j)=\frac{\exp\left(\frac{\sum_{n\in N}Q_n(b_i,j)}{T}\right)}{Z(T)}, \tag{5}$$

where,
T: temperature parameter.
$Z(T)$: the normalization function defined as follows

$$Z(T)=\sum_{j'\in A(b_i)}\exp\left(\frac{\sum_{n\in N}Q_n(b_i,j')}{T}\right). \tag{6}$$

If $T\to\infty$, the model becomes uniform distribution. If $T\to 0$, the model becomes greedy distribution, that is, the state-action pair with the highest Q value will be sampled. Therefore, the temperature parameter T can control the convergence of the probabilistic model, where the appropriate setting of T can be studied based on the concrete problems. The temperature T can either be fixed or be changed during the evolution process. However, changing the value of T during the evolution process would be a better choice, since a large temperature in early generations and a small temperature in later generations will help the proposed algorithm to avoid the premature convergence and find the global optimum smoothly.

D. Required population size

Theorem 1: Suppose the population size is denoted as M, given a state-action pair (b_i,j), the diversity loss rate of (b_i,j) can be given by

$$D_{(b_i,j)}=(1-P(b_i,j))^M \tag{7}$$

Proof: In order to generate one new individual, the probability that (b_i,j) is not sampled is $1-P(b_i,j)$. Therefore, when generating M individuals, the probability that (b_i,j) is not sampled is $(1-P(b_i,j))^M$. ■

Theorem 2: In PMBGNP-RL, Given a state-action pair (b_i,j) and its probability $P(b_i,j)$, if the acceptable diversity loss rate is defined by the confidence level ε, the required population size M satisfies the following condition

$$M\geq\frac{1}{P(b_i,j)}\log\frac{1}{\varepsilon}. \tag{8}$$

Proof: Based on *Theorem 1* and the following inequality

$$(1-x)\leq\exp(-x),\qquad 0\leq x\leq 1, \tag{9}$$

the diversity loss rate of (b_i,j) satisfy the following condition

$$\begin{aligned}(1-P(b_i,j))^M&\leq(\exp(-P(b_i,j)))^M\\&=\exp(-M\cdot P(b_i,j)).\end{aligned} \tag{10}$$

Then, since the diversity loss rate should not be larger than the confidence ε, the relation between M and ε can be represented as follows

$$\exp(-M\cdot P(b_i,j))\leq\varepsilon. \tag{11}$$

After logarithmic process, we can rewrite Eq. (11) to

$$M\geq\frac{1}{P(b_i,j)}\log\frac{1}{\varepsilon}.$$

■

The value of $P(b_i,j)$ can be obtained from the probabilistic model. Therefore, the population size M can be calculated by defining the confidence level ε. We can easily observe that when the population size is set at a small value, the diversity of state-action pair (b_i,j) will be lost significantly because the confidence level ε should approach to 1. On the other hand, in order to ensure the small diversity loss rate, the population size should be increased dramatically. In this paper, *Theorem 2* is used to set the population size to ensure the acceptable population diversity.

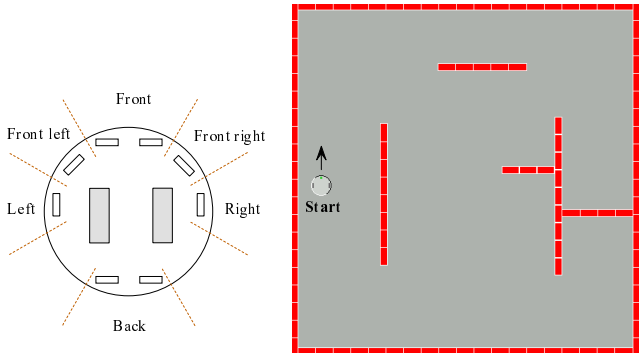
V. Simulations

Khepera robot (Fig. 5) [19] is a small (5.5cm) differential wheeled mobile robot, including 8 infrared sensors which allows 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 in front of 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 effectiveness of PMBGNP-RL is evaluated by controlling Khepera robot to solve the wall-following problem. A comparative study among standard GNP (SGNP), PMBGNP, PMBGNP-RL and PMBGNP-RL with Boltzmann distribution (PMBGNP-RLB) is carried out in this section.

The reward and fitness function of the wall following problem is designed based on [28] and shown as follows

$$\text{Reward}=\frac{v_R+v_L}{20}\cdot\left(1-\sqrt{\frac{|v_R-v_L|}{20}}\right)\cdot C, \tag{12}$$



(a). Khepera robot (b). Simulation environment

Fig. 5. Khepera robot and simulation environment.

$$\text{Fitness} = \frac{\sum_{step=1}^{S_{max}} \text{Reward}}{S_{max}}, \tag{13}$$

where,

v_R, v_L: the speed of right and left wheels,

S_{max}: the user predefined steps. S_{max} could control the robot's running time, which is equivalent to the problem size.

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

Figure 5 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.

A. Parameter settings

The node functions of GNP are shown in Table I. There are a total of 8 judgment functions to simulate the corresponding sensors of the robot. 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, where a is the return value of the simulated sensor. b is the user-defined threshold that determines which branch should be selected to transit. c and d denote the two branches of the judgement nodes. In this paper, b is set at 1000. Therefore, if the return value a is above 1000, branch c will be selected to transit, otherwise branch d will be selected. Each processing node determines the speed of the left or right wheel, i.e., $P_1(-10)$ means that the speed of the right wheel is changed to -10. 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 10 time units 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, so there are a total of $5 \times 8 = 40$ judgment nodes in each individual. The number of processing nodes for each

TABLE I
NODE FUNCTIONS USED FOR KHEPERA ROBOT.

Node	*ID*	Function
J_1, ..., J_8	1, ..., 8	Judge the value of the sensor of 1, ..., 8
$P_1(-10), P_1(-5), P_1(0), P_1(5), P_1(10)$	1-1, ..., 1-5	Determine the speed of the right wheel like -10, -5, 0, 5 or 10
$P_2(-10), P_2(-5), P_2(0), P_2(5), P_2(10)$	2-1, ..., 2-5	Determine the speed of the left wheel like -10, -5, 0, 5 or 10

processing function is set at 2, which means each individual consists of $2 \times 10 = 20$ processing nodes. Therefore, there are a total of 60 nodes and $40 \times 2 + 20 = 100$ branches in each individual. Meanwhile, based on the definitions of the state and action in the previous section, the size of the states is $|\mathcal{S}| = 100$ and that of actions is $|\mathcal{A}| = 60 - 1 = 59$.

Generally, as the other conventional EAs, SGNP is not so sensitive to the population size, while EDA approaches are much more sensitive to the population size. In the simulations, the population size of SGNP is set at 300, which consists of 1 elite individual, 120 crossover individuals and 179 mutation individuals. The crossover and mutation rates are set at 0.1 and 0.01, respectively. All these settings are the appropriate ones defined by hand-tuning. The population sizes of PMBGNP, PMBGNP-RL and PMBGNP-RLB are set based on the study of *Theorem 2*, which means their population sizes are set based on different values of confidence level ε. In these algorithms, the top 20% individuals are selected to construct the probabilistic models. The top 10% individuals are directly reproduced to the next population, while the rest are produced by the constructed model. The learning rate α and discount factor γ of PMBGNP-RL and PMBGNP-RLB are set at 0.9 and 0.1, respectively.

On the other hand, in the case of PMBGNP-RLB, the temperature T should be defined, we use a non-uniform method to set the value of temperature T, which means T is decreased w.r.t. the generations. In early generations, T is set at a large value to get more exploration of the search space, and decreased during the evolution process. Therefore, any monotonically decreasing function can be used to change the value of T. In this paper, we simply use the following equation for T:

$$T = \frac{GEN}{generation}, \tag{14}$$

where,

GEN: the maximal number of generations.

$generation$: the current number of generation, which is ranged from 1 to GEN.

Therefore, the value of T decreases from GEN to 1 during the evolution process.

B. Simulation results and analysis

The simulations are designed based on different problem sizes, i.e., the predefined step $S_{max} \in \{100, 200, 300, 400, 500\}$. Since the population sizes of these algorithms are different, therefore the terminal condition is the maximal number

TABLE II
THE FITNESS OF SGNP, PMBGNP, PMBGNP-RL AND PMBGNP-RLB OVER 20 INDEPENDENT RUNS.

Confidence	$S_{max} = 100$			$S_{max} = 200$			$S_{max} = 300$			$S_{max} = 400$			$S_{max} = 500$		
	PMBGNP	M1[a]	M2[b]	PMBGNP	M1	M2	PMBGNP	M1	M2	PMBGNP	M1	M2	PMBGNP	M1	M2
$\varepsilon = e^{-3}$	0.76	0.76	0.86	0.66	0.68	0.79	0.63	0.67	0.77	0.59	0.62	0.76	0.55	0.59	0.62
$\varepsilon = e^{-6}$	0.84	0.84	0.87	0.81	0.81	0.84	0.79	0.81	0.84	0.74	0.74	0.80	0.63	0.64	0.71
$\varepsilon = e^{-10}$	0.85	0.86	0.87	0.84	0.86	0.87	0.79	0.82	0.85	0.74	0.77	0.82	0.63	0.68	0.71
$\varepsilon = e^{-20}$	0.87	0.88	0.89	0.83	0.84	0.88	0.82	0.83	0.86	0.78	0.79	0.82	0.65	0.70	0.71
$\varepsilon = e^{-30}$	**0.89**	**0.89**	**0.89**	**0.88**	**0.88**	**0.88**	**0.87**	**0.87**	**0.87**	0.80	0.82	**0.83**	0.66	0.72	**0.74**
SGNP	**0.89**			0.83			0.82			0.76			0.66		

[a] M1: PMBGNP-RL without Boltzmann Distribution (PMBGNP-RL)
[b] M2: PMBGNP-RL with Boltzmann Distribution (PMBGNP-RLB)

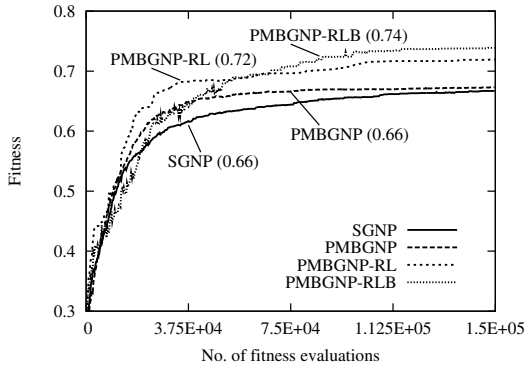


Fig. 6. The average fitness for wall following problem with S_{max}=500.

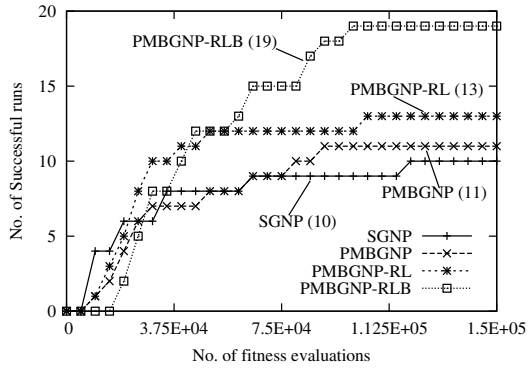


Fig. 7. The number of successful runs for wall following problem with S_{max}=500.

of fitness evaluations, where 150,000 is used in this paper. All the simulation results are the average over 20 independent runs.

In PMBGNP, PMBGNP-RL and PMBGNP-RLB, the population size is defined by the confidence level ε. The smaller ε is, the larger the population size is, and vice-versa. Note that given ε value, the population size of each generation can vary to adapt to the minimal probability of $P(b_i, j)$ of the current generation. However, in this paper, we simply use ε to define the population size in the initial generation, where the population size is preserved in the following generations. For example, in the case of $\varepsilon = e^{-30}$, the population size can be calculated by Eq. (8), which is $59 \times 30 = 1770$. The detailed fitness results of SGNP, PMBGNP, PMBGNP-RL and PMBGNP-RLB are shown in Table II. The simulation results are analyzed by the following points:

(1) Sensitivity of population size: Both of PMBGNP and PMBGNP-RL are sensitive to the population size. The results show that in the case of small population sizes, they cannot work well. However, with the appropriate settings of ε to ensure enough diversity, both of them can avoid the premature convergence to obtain much better results. On the other hand, comparing with PMBGNP and PMBGNP-RL, PMBGNP-RLB is not so sensitive to the population size. This is because the temperature parameter T of PMBGNP-RLB allows us to cope with more exploration and control the convergence speed. Therefore, even with small population sizes, it still work much better than PMBGNP and PMBGNP-RL. Meanwhile, with the appropriate setting of the population size, PMBGNP-RLB can converge to better results than PMBGNP and PMBGNP-RL.

(2) Fitness results: The results show that in the simple problems, i.e., $S_{max} = 100$, all the four algorithms perform equally. When the problem becomes more complex, SGNP achieves worse results than the others. Meanwhile, the proposed algorithm of PMBGNP-RL can achieve higher fitness than SGNP and PMBGNP, while PMBGNP-RLB obtains the best performance in all the problems.

Among the simulations, the case of $S_{max} = 500$ is the most complicated one since more efficient GNP programs should be evolved to control the robots to move longer steps. Therefore, we further emphasize on studying the performances of this case. Fig. 6 shows the average fitness of the case of $S_{max} = 500$. And the successful runs[2] are shown in Fig. 7. The results can be summarized as follows:

- Comparing between SGNP and PMBGNP, the probabilistic modeling of promising individuals allows PMBGNP to obtain faster convergence than SGNP. However, the final fitness results of these two algorithms are the same. This is because $\varepsilon = e^{-30}$ still cannot guarantee enough population diversity of PMBGNP to perform better performances than that of SGNP.
- PMBGNP-RL achieves better performances than SGNP and PMBGNP. Meanwhile, the convergence speed of

[2]One simulation run is denoted as a successful run if its best individual in the last generation can control the robot moving around the wall successfully.

PMBGNP-RL is also fast which is similar to PMBGNP.

- PMBGNP-RLB outperforms the other three algorithms. Meanwhile, since in early generations large values of temperature T are used, the probabilistic model copes with more exploration. Therefore, the fitness of PMBGNP-RLB is worse than the other three algorithms in early generations. However, with the decreasing of T during the evolution process, PMBGNP-RLB can finally converge to a better fitness.
- Among 20 independent runs, SGNP solves the problem successfully in 10 runs, while PMBGNP succeeds in 11 runs. The proposed algorithm PMBGNP-RL succeeds in 13 runs, and by applying Boltzmann distribution, 19 runs can solve the problem successfully.

VI. Conclusions and future work

In this paper, a novel EDA named PMBGNP-RL has been proposed. The proposed algorithm extends the graph chromosome-based PMBGNP by applying Sarsa-learning technique to measure the importance of the structure of its chromosome. Meanwhile, Boltzmann distribution is introduced to PMBGNP-RL. The effectiveness of the proposed algorithm is testified in an agent control problem, i.e., robot control. The simulations show that the proposed algorithm of PMBGNP-RL outperforms the conventional algorithms in several wall-following problems with different complexities. On the other hand, by introducing Boltzmann distribution, PMBGNP-RL tends to become less sensitive to the population diversity and converge to better results. The main points that make PMBGNP unique in the EDA fields are: Firstly, PMBGNP introduces EDA to the graph-based EAs; Secondly, PMBGNP is capable of solving the agent control problems rather than the conventional EDA designed for some benchmark problems. This paper introduces Sarsa-learning to PMBGNP to estimate the interactions between two nodes, which is based on the assumption of the first-order Markov Model. Therefore, in the future, higher-order Markov Model will be studied in PMBGNP framework to estimate multivariate interactions. Moreover, extended PMBGNP will be studied to control the robot in noisy or dynamic environments. Also, applying PMBGNP to some other problems would be another challenge.

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] 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.

[4] 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.

[5] 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.

[6] R. P. Salustowicz and J. Schmidhuber, "Probabilistic Incremental Program Evolution", *Evol. Comput.*, Vol. 5, No. 2, pp. 123-141, 1997.

[7] 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.

[8] A. Teller and M. Veloso, "PADO: Learning tree structured algorithms for orchestration into an object recognition system", *Tech. Report. No. CMU-CS-95-101, Carnegie Mellon University*, 1995.

[9] J. F. Miller and P. Thomson, "Cartesian Genetic Programming", *In Proc. of the 3rd European Conf. on Genetic Programming*, pp. 121-132, 2000.

[10] 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.

[11] 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.

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

[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] T. Murata and T. Nakamura, "Multi-Agent Cooperation Using Genetic Network Programming with Automatically Defined Groups", *In Proc. of the Genetic and Evol. Comput. Conf.*, pp. 712-714, 2004.

[15] K. Shimada, K. Hirasawa and J. Hu, "Genetic Network Programming with Acquisition Mechanisms of Association Rules", *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 10, No. 1, pp. 102-111, 2006.

[16] K. Hirasawa, T. Eguchi, J. Zhou, L. Yu and S. Markon, "A Double-Deck Elevator Group Supervisory Control System Using Genetic Network Programming", *IEEE Trans. on Systems, Man and Cybernetics*, Part C, Vol.38, No.4, pp. 535-550, 2008.

[17] 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.

[18] 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.

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

[20] K. Yanai and H. Iba, "Estimation of Distribution Programming based on Bayesian Network", *In Proc. of the IEEE Congress on Evol. Comput.*, pp. 1618-1625, 2003.

[21] K. Sastry and D. E. Goldberg, "A Probabilistic Model Building and Competent Genetic Programming", In R. L. Riolo and B. Worzel, editors, *Genetic Programming Theory and Practice*, Vol. 13, pp. 205-220, 2003.

[22] Y. Hasegawa and H. Iba, "A Bayesian Network Approach to Program Generation", *IEEE Trans. on Evol. Comput.*, Vol. 12, No. 6, pp. 750-764, 2008.

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

[24] G. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems", *Tech. Rep. CUED/F-INFENG/TR 166, Cambridge University, Engineering Department*, 1994.

[25] Q. Meng, S. Mabu and K. Hirasawa, "Genetic Network Programming with Sarsa Learning Based Nonuniform Mutation", *In Proc. of the IEEE Int'l Conf. on Systems, Man and Cybernetics*, pp. 1273-1278, 2010.

[26] H. Handa, "EDA-RL: Estimation of Distribution Algorithms for Reinforcement Learning Problems", *In Proc. of the Genetic and Evol. Comput. Conf.*, pp. 405-412, 2009.

[27] C. Watkins, "Learning from Delayed Rewards", *Ph.D thesis, Cambridge University, King's College*, 1989.

[28] 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.