A Novel Graph-Based Estimation of the Distribution Algorithm and Its Extension Using Reinforcement Learning

Xianneng Li, *Student Member, IEEE,* Shingo Mabu, *Member, IEEE,* and Kotaro Hirasawa, *Member, IEEE*

***Abstract*—In recent years, numerous studies have drawn the success of estimation of distribution algorithms (EDAs) to avoid the frequent breakage of building blocks of the conventional stochastic genetic operators-based evolutionary algorithms (EAs). In this paper, a novel graph-based EDA called probabilistic model building genetic network programming (PMBGNP) is proposed. Using the distinguished graph (network) structure of a graph-based EA called genetic network programming (GNP), PMBGNP ensures higher expression ability than the conventional EDAs to solve some specific problems. Furthermore, an extended algorithm called reinforced PMBGNP is proposed to combine PMBGNP and reinforcement learning to enhance the performance in terms of fitness values, search speed, and reliability. The proposed algorithms are applied to solve the problems of controlling the agents' behavior. Two problems are selected to demonstrate the effectiveness of the proposed algorithms, including the benchmark one, i.e., the Tileworld system, and a real mobile robot control.**

***Index Terms*—Agent control, estimation of distribution algorithm (EDA), genetic network programming (GNP), graph structure, reinforcement learning (RL).**

I. Introduction

IN THE LAST few years, there has been a significant development of the estimation of distribution algorithm (EDA) in both theory and practice [1]–[4]. Unlike the conventional evolutionary algorithms (EAs) that use stochastic ways to simulate the biological genetic operators for new population generation, EDA constructs a probabilistic model using the techniques of statistics or machine learning to estimate the probability distribution of the current population, and samples the model to generate a new population. Many studies have investigated whether EDA can outperform conventional EA by avoiding the premature convergence and speeding up of the evolution process in some problems [5]–[8]. A large number of studies have been conducted on EDA to propose numerous algorithms. Particularly, from the perspective of individual representation, EDA can be simply classified into two categories, which are probabilistic model building genetic algorithm (PMBGA, or genetic algorithm-based EDA) [9] and probabilistic model building genetic programming (PMBGP, or genetic programming-based EDA) [10]. PMBGA employs GA's string structure to represent its individuals and is mainly applied to solve optimization problems, while PMBGP uses GP's tree structure to represent its individuals for program evolution.

In this paper, a novel graph-based EDA called probabilistic model building genetic network programming (PMBGNP) [11] is described. The aim of developing PMBGNP is to extend EDA from the string and tree structures to the graph structure, where the directed graph (network) structure of genetic network programming (GNP) [12], [13] is employed. Some previous research has shown the superiority of graph-based EAs in terms of higher expression ability than that of conventional GP [12]–[16]. GNP is one such graph-based EA, which extends GA and GP by using a directed graph (network) structure to represent its individuals. Different from the other graph-based EAs, GNP is first designed for solving the problems of controlling the agents' behavior, while in recent years it has been extended to many other problems, such as multiagent systems [17], data mining [18], elevator system control [19], intrusion detection system [20], etc. Therefore, from the perspective of individual representation, PMBGNP has higher expression ability than the conventional EDAs to efficiently solve some problems due to the directed graph structure of GNP.

On the other hand, another challenge in EDA is using it to explore many other problems. This paper applies PMBGNP to solve the problems of controlling the agents' behavior, where most of the current EDAs are designed to solve the other problems. Two problems are selected to demonstrate the effectiveness of this paper, including the benchmark one, i.e., the Tileworld system [21], and a real mobile robot control, Khepera robot control [22], [23]. Therefore, there are mainly two primary features of PMBGNP.

1) EDA is extended to graph-based EA.
2) EDA is applied to solve the problems of controlling the agents' behavior.

In addition, we propose an extended algorithm called reinforced PMBGNP (RPMBGNP) that combines reinforcement

Manuscript received July 5, 2012; revised October 27, 2012; accepted December 24, 2012. Date of publication January 9, 2013; date of current version January 27, 2014.

The authors are with the Graduate School of Information, Production and Systems, Waseda University, Fukuoka 808-0135, Japan (e-mail: sennou@asagi.waseda.jp; mabu@aoni.waseda.jp; hirasawa@waseda.jp).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org provided by the authors. This includes information on the detailed experimental settings and two additional experiments. This material is 825 KB in size.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2013.2238240

learning (RL) [24] and PMBGNP in order to enhance its performance. Inspired by behavioral psychology, RL is concerned with reinforcing the growth of the individuals by learning their experiences to approximate the Q values of state–action pairs. RPMBGNP first factorizes the entire solution to a sequence of state–action pairs, and calculates the Q values of state–action pairs that are further incorporated into the probabilistic modeling. By defining the state–action space using the PMBGNP's graph structure, RL allows us to study the substructure (node connections) of individuals during their executions. The learned knowledge, i.e., Q values, is used to build the probabilistic model, which directly benefits the problem solving of controlling the agents' behavior.

This paper is organized as follows. The next section briefly introduces related work and highlights the contribution of our work through comparison with others. Section III describes PMBGNP in detail. In Section IV, the details of combining PMBGNP and RL are presented. The experimental studies are carried out in Section V. Section VI presents the discussion and Section VII concludes this paper.

II. Related Work and Comparison

A. Estimation of Distribution Algorithm

Various EDAs have been proposed to draw its success. Despite many different implementations, the basic procedure of EDA can be summarized below.

1) Randomly generate an initial population.
2) Probabilistic modeling from the selected individuals.
3) Sample the model to generate the new population.
4) Go back to Step 2 until the terminal conditions.

The idea of EDA was first introduced in the field of binary GA [1] and has attracted much attention in the last decade. The class of EDAs using GA's individual representation can be called PMBGA. From the perspective of model complexity, PMBGA mainly consists of three groups: univariate interactions, pairwise interactions, and multivariate interactions, which study the building blocks (BBs) of different complexities. The univariate algorithms, such as population-based incremental learning (PBIL) [1], univariate marginal distribution algorithm (UMDA) [2], and compact genetic algorithm (CGA) [25], assume that there is no relationship between the variables. These algorithms work well for the linear problems, but fail on the problems with strong interactions among variables. Therefore, some algorithms propose pairwise models for solving the quadratic problems [26]–[28], where the BBs of order two can be estimated, and another class of algorithms extends PMBGA to estimate multivariate interactions for many hard problems that consist of multivariate BBs [5], [29]–[31].

Some work used the idea of EDA in tree structure GP to propose a new research topic called PMBGP [10]. Applying EDA from string structure to tree structure is a more complex way of representing solutions, which explore EDA in solving a large class of problems, such as program evolution. Based on the model complexity, PMBGP can also be classified into three groups. Probabilistic incremental program evolution (PIPE) [32] is the first algorithm of PMBGP that extends the univariate PBIL to GP for program evolution. The estimation of distribution programming (EDP) [33] was derived later to model pairwise interactions. Extended compact GP (ECGP) [34] and program optimization with linkage estimation (POLE) [8] are the multivariate PMBGPs that can recombine more complex BBs.

These PMBGPs are called prototype tree-based methods, which directly extend EDA to standard GP. There is another class of PMBGPs that applies EDA to grammar guided GP (GGGP) [35], called grammar model-based PMBGP [10], [36]. Another work called N-gram GP [37] explores extending PMBGP to linear GP. The EDAs mentioned above are generally used for the problems of discrete domains, which can be broadly called discrete EDA. There is also a wide range of EDAs extending the existing discrete EDAs to solve the problems of continuous domains [38], [39].

The work of this paper is to extend EDA from GA- and GP-style individual representations to a more complex one—graph-based individual representation—to propose a new EDA called PMBGNP. This paper is a direct extension of PMBGA and prototype tree-based PMBGP in discrete domains, while the primary feature is that the directed graph (network) structure of a graph-based EA called GNP is adopted to represent the individuals. This leads to the advantage that it is easy to apply the existing techniques of EDA to PMBGNP. For example, the probabilistic modeling of PMBGNP in this paper is derived from the univariate EDAs, i.e., PBIL, UMDA, and PIPE. The preliminary work of this graph-based EDA was previously published as [11]. However, the issue of its diversity loss (*Theorem 1* of this paper) was not addressed, which caused the problem of premature convergence for the complex problems [40]. In this paper, we incorporate the Boltzmann distribution into the probabilistic modeling of PMBGNP to overcome this problem, and its effectiveness is confirmed empirically.

B. Applications of EDA

Various empirical studies have been conducted to clarify the performance of EDA in the benchmark problems of GA and GP, i.e., function optimization problems by PMBGAs, symbolic regression and Royal trees problems by PMBGPs. Meanwhile, EDA has also been successfully applied to a number of applications, such as multiobjective optimization [41], dynamic problems [42], bioinformatics [7], etc. Despite many different implementations, one important challenge of EDA is to explore it in some other applications.

Particularly, there are many studies on EAs, such as GP and evolutionary programming (EP) [43], to successfully solve the problems of controlling the agents' behavior, while until now, there are only a few works on studying EDA to solve such problems. In [32] and [44], the limited work that applies PIPE, a univariate PMBGP, to solve the problems of controlling the agents' behavior are shown. Another work on applying EDA to control the agents is a recent one called EDA-RL [45], which has been reported to successfully solve the simple problems but fail on some complex partially observable problems. It is meaningful to study this problem, since many real-world applications can be solved by such agent systems, i.e., stock trading, ForeX and robot control, etc.

C. Combination of EC and RL

Many previous studies have successfully applied RL to EC for the problems of controlling the agents' behavior [13], [46], [47] . In certain respects, this topic is the combination of global search (population evolution) and local search (individual learning), which has been widely discussed by the Baldwin effect [46]. It relies on the idea of phenotypic plasticity, that an individual is able to adapt to its environment during its lifetime, where the successful adaptation that increases fitness results will tend to proliferate in the population. To achieve adaptability, Downing [46] proposed reinforced GP (RGP) by adding a special form of leaf nodes called choice nodes to GP's tree structure, where the functions of the choice nodes are determined during the individual learning by Q learning. Therefore, Q learning is the computational analog of phenotypic plasticity in biological evolution. Kamio and Iba [47] proposed GP+RL to integrate GP and Q learning, where evolution and learning are executed step by step. That is, evolution is first done to find the roughly optimal trees of GP, then Q learning is carried out to adapt the functions of action nodes in GP. Mabu *et al.* [13] proposed GNP-RL to integrate GNP and Sarsa learning (Sarsa), where the most important difference comparing with the previous methods is how to create state–action spaces (Q tables). RGP uses the GP structures, i.e., statements from root to leaf nodes, to define its state space, while GP+RL uses the real states to define it. Both of them face the problem that the state space may become extremely large, causing difficulty in the learning process, especially when solving the complex problems. On the other hand, GNP-RL provides an alternative way to overcome this problem. That is, it creates the Q tables using its graph structure, where the nodes of GNP's graph structure represent the states and the selection of a function/subnode in each node correspond to actions. In other words, the size of its state space is only based on the number of nodes in GNP's graph structure, which is generally small.

The key point of these remarkable methods of integrating EC and RL is to create the adaptation of individuals. Therefore, they need to change the original tree/graph structures of GP/GNP to create the adaptation ability. In this paper, we propose a very different algorithm to integrate EC and RL, called reinforced PMBGNP (RPMBGNP). The basic idea of RPMBGNP is to create state–action space using GNP's graph structure (however, without changing the original graph structure), and apply Sarsa to update the Q table based on the individual executions. At the end of each generation, the learning knowledge, i.e., Q value, is incorporated into the probabilistic modeling. Therefore, the primary objective of RPMBGNP is to improve the evolutionary efficiency in terms of constructing a more accurate probabilistic model by RL, rather than to achieve the adaptation ability. Moreover, in conventional methods and our preliminary work of integrating PMBGNP and RL [48], each individual is learned by RL independently. In other words, they create multiple Q tables and each individual maintains its own Q table, which generally causes the problems of low learning efficiency and memory cost. On the other hand, RPMBGNP only creates one Q table and maintains it by learning the executions of the promising individuals, leading to high learning efficiency and memory saving.

In the EDA field, a univariate PMBGA called reinforcement learning EDA (RELEDA) [49] is the only limited study on integrating EDA and RL. It introduces a number of parameters corresponding to the variables, and the estimation of the probability distribution is based on the update of these parameters by RL in the iterative process. On the other hand, RPMBGNP holds the assumption of RL called the Markov property. That is, the conditional probability distribution $\mathbb{P}(\cdot)$ of state s_t at time t depends only upon its previous state s_{t-1}, formulated as

$$\mathbb{P}(s_t|s_{t-1}, s_{t-2}, \ldots, s_0) = \mathbb{P}(s_t|s_{t-1}). \quad (1)$$

Although some tasks are non-Markov, the assumption of being Markovian provides a theoretical basis of RL for predicting future rewards and for selecting actions to obtain good performance [24]. From this point of view, there is a class of EDAs based on the Markov random fields/Markov network [50]–[52]. However, these methods and RELEDA are used to solve function optimization problems rather than solving the problems of controlling the agents' behavior as does RPMBGNP.

III. Probabilistic Model Building Genetic Network Programming

A. Directed Graph (Network) Structure

1) *Basic Structure:* PMBGNP uses the directed graph (network) structure of GNP to represent its individual structure. Its basic structure is shown in Fig. 1, which can be represented by the phenotype and genotype expressions. Phenotype shows the directed graph structure in which the nodes are connected by directed branches and genotype demonstrates the bit-strings encoding of the chromosomes.

Each program (individual) is composed of one start node, multiple judgment nodes, and processing nodes. The start node, having no function and conditional branch, is only used to decide the first node to be transited. Judgment nodes work as "if-then" decision-making functions to judge the environments by dealing with the specific inputs of the problems, such as the returned sensor information of the Tileworld system and Khepera robot in this paper. Each judgment node has several conditional branches corresponding to the judgment results, which are defined by the problems. Processing nodes preserve the processing functions to the environments, such as determining the agent's actions of the Tileworld system and the wheel speeds of Khepera robot. Each 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 processing. That is, the evolution can efficiently evolve the compact programs by only selecting the necessary judgments and processing.

The number of judgment nodes and processing nodes is predefined by designers appropriately. Therefore, the directed

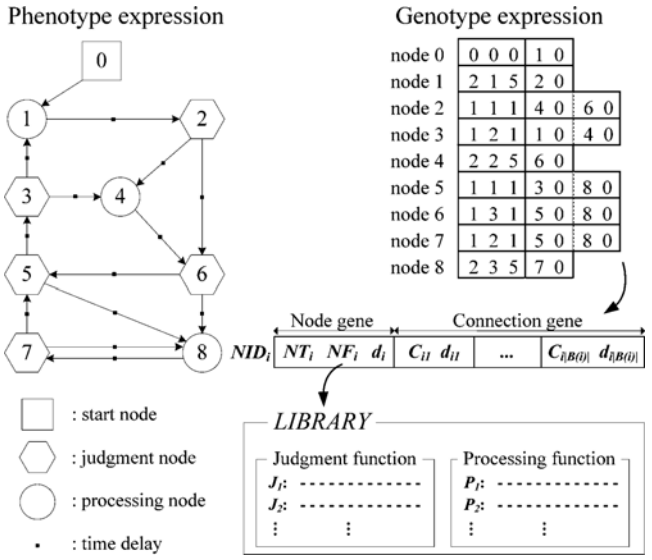

Fig. 1. Directed graph (network) structure of GNP (and PMBGNP).

graph structure of GNP never causes the bloat problem of GP [12]. Such a directed graph structure can perform quite well by realizing the repetitive processes based on the frequent reuse of nodes, which work like the automatically defined functions in GP. As a result, the directed graph structure of GNP can generate efficient programs based on both the current and past information.

In addition to its distinguished directed graph structure, GNP has time delays in each node and branch. The motivation of introducing time delays is to model the agent with a human brain that needs the time for thinking. Time delays in GNP are designed in three types: delays spending time on judgment nodes, processing nodes, and node transitions. They are defined by designers in advance based on the problems. For example, for both the Tileworld system and Khepera robot in this paper, the time delay of each judgment node is set at one time unit, that of each processing node is five time units, and that of each node transition is zero. As a result, the one step of an agent's behavior can be defined. In this paper, one step ends when five or more time units are used. That is, the agent can only do fewer than four judgments and one processing, or five judgments, in one step. By introducing time delays, the directed graph of GNP can efficiently implement flexible programs considering the real-world environments, where the agents need to solve problems in constrained time.

2) *Gene Structure and Notations:* As shown in Fig. 1, the nodes of GNP's directed graph are encoded into bit strings. Each node has a unique number NID that remains unchanged during evolution, while the functions of the nodes with the same number in different individuals are identical. Let i represent a node number of GNP. NT_i defines the node type, where $\mathrm{NT}_i = 0$, 1, or 2 for start, judgment, or processing node, respectively. NF_i represents the function, 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.

The notations of the directed graph of GNP used are as follows:

N_J: set of suffixes of judgment nodes in one individual.
N_P: set of suffixes of processing nodes in one individual.
N_{node}: set of suffixes of nodes in one individual.
$B(i)$: set of suffixes of branches in node i. $|B(i)|$=1 in both start and processing nodes, while $|B(i)|$ in judgment nodes is determined by its judgment function NF_i.
B: set of suffixes of branches in one individual.

The start node in the directed graph is only used to determine the first node to be executed, where it does not have a function and its connected node is predefined and fixed. Therefore, to simplify the explanation, the start node and its branch are not considered in N_{node} and B, since they are also not studied in the probabilistic modeling of PMBGNP. We use G=(N_{node}, B) to denote the directed graph structure of GNP.

The total number of nodes can be calculated by

$$|N_{\mathrm{node}}| = |N_J| + |N_P| \tag{2}$$

and the total number of branches is

$$|B| = \sum_{i \in N_J} |B(i)| + |N_P|. \tag{3}$$

The values of these variables are predefined by designers and fixed during evolution to generate compact programs.

B. Details of PMBGNP

PMBGNP constructs a probabilistic model from a set of selected individuals, and uses the model to generate a new population. The method of probabilistic modeling in PMBGNP is derived from the univariate EDAs.

1) *Probabilistic Model of PMBGNP:* In PMBGNP, the probabilistic model P is composed of a set of probabilities $P(b(i), j)$, which represents the connection probability from branch $b(i)$ of node i to node j, shown as

$$P = \{P(b(i), j) | i, j \in N_{\mathrm{node}}; b(i) \in B(i)\}.$$

Fig. 2 shows an example of the probabilistic model in PMBGNP, where the size of the model, i.e., the number of probabilities, is determined by the directed graph of individuals that can be computed by

$$|P| = |B|(|N_{\mathrm{node}}| - 1). \tag{4}$$

Here, the term -1 denotes that the node cannot connect to itself, which will cause the infinite loop of the program.

To calculate the connection probability $P(b(i), j)$, the following equation is used:

$$P(b(i), j) = \frac{\exp\left[\frac{\sum_{n=1}^{N}\left(\delta_n(b(i), j) + \eta\sigma_n(b(i), j)\right)}{T}\right]}{Z(b(i))} \tag{5}$$

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

$$Z(b(i)) = \sum_{j' \in A(b(i))} \exp\left[\frac{\sum_{n=1}^{N}\left(\delta_n(b(i), j') + \eta\sigma_n(b(i), j')\right)}{T}\right]$$

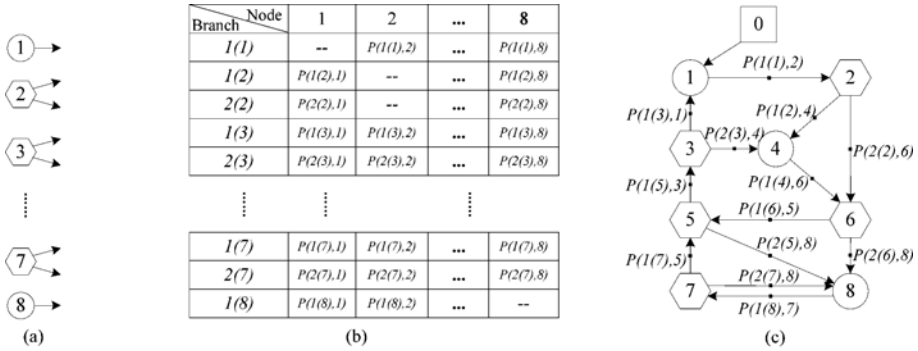

Fig. 2. Example of the probabilistic model in PMBGNP. "–" in (b) denotes that there does not exist connection probability, since the node cannot connect to itself to avoid the infinite loop. (a) Elements of the directed graph. (b) Probabilistic model of PMBGNP. (c) Example of the individuals sampled by the model.

where

N: number of best individuals;

$A(b(i))$: set of suffixes of possible nodes connected from branch $b(i)$ of node i;

$\delta_n(b(i), j)$: value defined by

$$\delta_n(b(i), j) = \begin{cases} 1 & \text{if branch } b(i) \text{ of node } i \text{ in individual } n \text{ is connected to node } j, \\ 0 & \text{otherwise.} \end{cases}$$

$\sigma_n(b(i), j)$: value defined by

$$\sigma_n(b(i), j) = \ell \quad \text{if the transition from branch } b(i) \text{ of node } i \text{ to node } j \text{ in individual } n \text{ occurs } \ell \text{ times.}$$

η: coefficient.

T: temperature parameter.

The probabilities are calculated by considering the connection and transition information between different nodes. The term $\delta_n(b(i), j)$ in (5) is used to count the frequency of node connection $(b(i), j)$, which is inspired by the classical univariate EDAs. Another term $\sigma_n(b(i), j)$ is also used to consider the reusability of node connections. This is because in the directed graph of GNP, not all the node connections are used to solve the tasks, where the node transitions are made by selecting the necessary nodes. Therefore, the reusability of node connections is useful to measure the importance. Parameter η is used to balance the effects of these two factors.

2) *Boltzmann Distribution:* Besides, the Boltzmann distribution is employed in the probabilistic modeling of PMBGNP to correct its convergence. The Boltzmann distribution is introduced due to the issue of diversity loss in PMBGNP, which causes the large required population size M.

Theorem 1: The required population size M of PMBGNP is larger than that of univariate EDAs, i.e., UMDA and PIPE.

The proof of *Theorem 1* is shown in Appendix A. As a result, to estimate an accurate probabilistic model, PMBGP and PMBGNP need a very large M; however, this is impractical in many real-world problems. One way to overcome this problem is to introduce some exploration methods, i.e., mutation operator [32], [40] and Laplace correction [33], to evolve practical population. Rather than simply adding the exploration regardless of the probability distribution, we use the Boltzmann distribution to relax the sample pressure w.r.t. on the estimated distribution of PMBGNP like (5). The fundamental basis of the Boltzmann distribution relies on the Hammersley-Clifford theorem [51], [53] that any probability distribution with the Markov property can be represented with Boltmann distribution, since the process of GNP program can be viewed as MDP in certain degrees [13], [17]. In the meantime, the Boltzmann distribution also makes all probabilities of the probabilistic model to be positive values, which allows all node connections to be sampled possibly.

In the Boltzmann distribution that is also known as Gibbs measure, i.e., $p(x) = \exp(E(x)/T)/\sum_{x'} \exp(E(x')/T)$, a parameter called temperature (T) is introduced to explicitly control the convergence of the probabilistic model, where:

1) if $T \to \infty$, the model becomes uniform distribution. In this case, all variables are sampled equally, regardless the values of their energy functions;
2) if $T \to 0$, the model becomes greedy distribution. That is, the variable with the highest energy will be sampled with probability 1.

In this paper, the energy function of the probabilistic model is the information of node connections and transitions, which is $E(b(i), j) = \sum_{n=1}^{N} (\delta_n(b(i), j) + \eta\sigma_n(b(i), j))$.

Therefore, the temperature parameter T can control the convergence of the probabilistic model, which can be either fixed or changed during the evolution. In this paper, we use an adaptive method to determine T, where large values in early generations and small ones in later generations are set to avoid the premature convergence and to find the global optimum smoothly by balancing the tradeoff between the exploitation and exploration. A monotonically decreasing function is used in this paper to determine the temperature

$$T = \frac{\tau}{t+1} \tag{6}$$

where

τ: coefficient.

t: the current number of generations.

The value of τ is determined based on the energy function $E(b(i), j)$, which can explicitly control the shape of (6).

Algorithm 1 Algorithm of PMBGNP

1: $|\text{Pop}| = M$
$t \leftarrow 1$
2: $\text{Pop}(t) \leftarrow$ generate the initial population randomly;
$\text{Fit}(t) \leftarrow$ evaluate the fitness of $\text{Pop}(t)$;
3: $\text{Best}(t) \leftarrow$ execute truncation selection to select a set of best individuals, where $|\text{Best}(t)| = N$, $(N \leq M)$;
4: $P \leftarrow$ construct a probabilistic model from $\text{Best}(t)$ according to (5);
5: $\text{Pop}(t+1) \leftarrow$ generate the new population by sampling P;
$\text{Fit}(t+1) \leftarrow$ evaluate the fitness of $\text{Pop}(t+1)$;
6: $t \leftarrow t+1$
if the termination conditions are not met, go back to 3.

Algorithm 2 Algorithm for the generation of a new individual

1: set all branches of all nodes unconnected;
2: **for** all $i \in N_{\text{node}}$ **do**
3: **for** all $b(i) \in B(i)$ **do**
4: connect branch $b(i)$ to node j with the probability of $P(b(i), j)$;
5: **end for**
6: **end for**

3) *Algorithm of PMBGNP:* The pseudocode of PMBGNP is shown in Algorithm 1, which is similar to conventional EDAs.

After constructing the probabilistic model, it is sampled to generate a new population, corresponding to Step 5 of Algorithm 1.

Since the directed graph G is predefined and fixed, it is easy to generate a new individual. In each iteration, branch $b(i)$ is decided to connect to node j based on the obtained probability $P(b(i), j)$. The process is repeated until all node connections are generated. The pseudocode is presented in Algorithm 2. The time complexity of generating one individual is $O(|B|)$.

IV. Reinforced PMBGNP

Standard PMBGNP is mainly based on the frequencies of node connections by simple maximum likelihood estimation (MLE) that is inspired by the classical univariate EDAs. In these algorithms, when we observe an element, i.e., variable/function, we just simply give it a weight 1, which may not measure its importance precisely. In other words, all the observed elements are treated equally in the probabilistic modeling, regardless of the significance of them from different individuals with different fitness. Therefore, to estimate a more precise model, one may consider this matter in the probabilistic modeling. That is, the observed elements are used in the probabilistic modeling w.r.t. the fitness of their individuals [33], [54], which may not guarantee the performance of PMBGNP, however, due to the hard control of their weights [55]. Following this viewpoint, reinforced PMBGNP (RPMBGNP) is proposed in this paper, where RL is combined with evolution to overcome the above problem. The aim of RPMBGNP is to learn knowledge/experience, i.e., Q values, using RL to measure the quality of node connections of PMBGNP, where the Q values are used in the probabilistic modeling of RPMBGNP to estimate a more precise model.

A. Preliminaries

In the problems of controlling the agents' behavior, each PMBGNP individual $n \in G$ can be viewed as a policy of the agents [24], where the transitions among the nodes of the individual are used to control the agents.

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

Definition 2 (State): State s is defined as a branch of a node in G. Therefore, the set of states S refers to the set of branches in G, represented by B (S=B).

Definition 3 (Action): Action a is defined as a node in G. Therefore, the set of actions A corresponds to the set of nodes in G (A=N_{node}).

With such definitions, the PMBGNP population with total M individuals can generate M episodes, which can be further factorized to M sequences of state–action pairs. For each episode, the state–action pairs can be represented as

$$(S, A)_n = \{(s_1, a_1)_n, (s_2, a_2)_n, \ldots, (s_L, a_L)_n\} \tag{7}$$

where L is length (final time step) of episode n.

Fig. 3 depicts an example of the procedure to obtain a sequence of state–action pairs. Concretely speaking, an activated branch corresponds to the current state, while the selection of the next node represents an action. Taking Fig. 3 as an instance, when the agent stands in the current state at time step t_3, i.e., branch 1 of node 4, it decides an action to select node 6 to transit. Therefore, the states and actions can be substituted by branches and nodes of G, respectively, which says that a node connection is equivalent to a state–action pair.

Definition 4 (State-action pair): A state–action pair (s, a) corresponds to a node connection from branch s to node a.

Based on these definitions, RL, i.e., temporal-difference (TD) learning [24], can be easily combined, where the state–action Q values ($Q(s, a)$) can be approximated by learning the individual executions. In this paper, an on-policy method called Sarsa [24] is used to update the Q values. At time step t, the main function of Sarsa for updating the Q values depends on the current state–action pair (s_t, a_t) of the agent, the reward r_t the agent observes, and the new state–action pairs (s_{t+1}, a_{t+1}) in the next time step, as shown in

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big] \tag{8}$$

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

Sarsa follows the true experience of the agents to update the Q values. Consequently, the actual knowledge of the individuals can be collected to measure the quality of the node connections.

B. Updating of the Q values

After defining the directed graph structure G of PMBGNP, a Q table that consists of the $Q(S, A)$ values for all possible

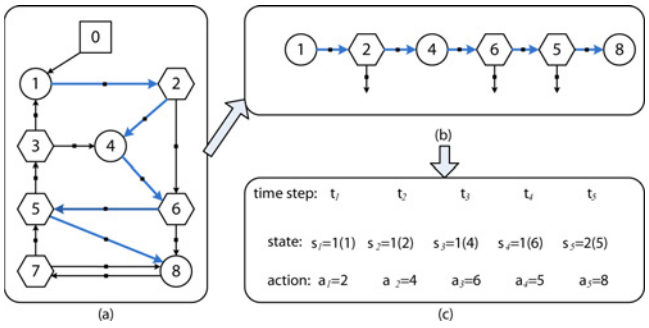

Fig. 3. Example of the procedure to factorize a PMBGNP individual to the sequence of state–action pairs. (a) Individual. (b) Sequence of state–action pairs (node transitions).

Algorithm 3 Algorithm for the updating of the Q values

1: $n \leftarrow 1$;
2: **for** $n \leq N$ **do**
3: execute individual n, and obtain the episode (sequence of state–action pairs) $(S, A)_n$;
4: update the corresponding Q values by (9);
5: $n \leftarrow n + 1$;
6: **end for**

combinations of state–action pairs is generated. By *Definition 5*, we can substitute the state–action pairs with the node connections of PMBGNP. As a result, the structure of the Q table is similar to that of probabilistic model of PMBGNP, as shown in Fig. 2(b). The initial $Q(S, A)$ values can be either 0 or positive constants, which are problem specific. In this paper, we initialize all $Q(S, A)$ values to 0.

During each individual execution, the executed node transitions are recorded sequentially, where the memorized sequence of state–action pairs forms the episode of RL. After obtaining the episodes, Sarsa is applied to update the Q values. Suppose the current state and action at time step t are, respectively, $b(i)$ and j, which means that the current state–action pair (s_t, a_t) is formed by node connection $(b(i), j)$, and the state–action pair in the next time step is (s_{t+1}, a_{t+1})=$(b(j), k)$. Then, the Q value of $(b(i), j)$ is updated by

$$Q(b(i), j) \leftarrow Q(b(i), j) + \alpha\left[r_j + \gamma Q(b(j), k) - Q(b(i), j)\right] \quad (9)$$

where r_j is reward of choosing node j at branch $b(i)$ of node i, and

1) in the case of $NT_j = 1$ (j is a judgment node), $r_j = 0$;
2) in the case of $NT_j = 2$ (j is a processing node), r_j is given after processing node j, which is problem specific.

The procedure of updating Q values in each generation is shown in Algorithm 3. It describes that the Q values are updated based on the execution of the best individuals (truncation selection with size N). As a result, we can collect the experience of the promising individuals into one Q table that consists of all $Q(S, A)$ values.

C. Probabilistic Model of RPMBGNP

The good state–action pairs will be rewarded with higher Q values by RL, and vice-versa, which implies that the collected Q values can explicitly express the quality of node connections of PMBGNP. By using these learned information properly, we can estimate a more accurate probabilistic model for PMBGNP.

Incorporating the learned Q values, the connection probability $P(b(i), j)$ of RPMBGNP is calculated as

$$P(b(i), j) = \frac{\exp\left(\frac{Q(b(i), j)}{T}\right)}{Z(b(i))}. \quad (10)$$

The normalization function $Z(b(i))$ is calculated by

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

and the temperature parameter T is obtained by (6).

D. Algorithm of RPMBGNP and Comparison With PMBGNP

The pseudocode of RPMBGNP is similar to PMBGNP shown in Algorithm 1. However, RPMBGNP applies RL to update the Q values by Algorithm 3 and the probabilistic modeling is based on the learned Q values.

Comparing with PMBGNP, the probabilistic model of RPMBGNP replaces the connection and transition information between different nodes [the terms $\delta_n(b(i), j)$ and $\sigma_n(b(i), j)$ in (5)] with the learned Q values. For instance, in PMBGNP, when we observe state–action pair $(b(i), j)$ $\bar{n}$ times among the best N individuals, we just simply give it a weight $\bar{n}$ and its count of node transitions. However, in RPMBGNP, the weight of the observed $(b(i), j)$ is determined by $Q(b(i), j)$.

By calculating the frequencies of node connections and transitions using simple MLE, PMBGNP aims to gather the information of the elite individuals for its probabilistic modeling. However, RPMBGNP can implicitly collect these two factors simultaneously. That is, the Q values can convey both of the node connections and transitions information. First, RL allows us to capture the information of node connections since a Q value will be updated if its corresponding state–action pair is observed in the best N individuals. Second, RPMBGNP records the node transitions as the episode of RL, and if a state–action pair is transited multiple times in one episode, its corresponding Q value will be updated multiple times. This implies that the influence of node transitions is implicitly considered into the updating of Q values. As a result, the single term $Q(b(i), j)$ can consider both of the node connections and transitions into one whole.

From the perspective of time complexity, PMBGNP records all the node connections of G in the best N individuals by MLE to construct its probabilistic model, which requires time $O(N|B|)$. RPMBGNP only spends time on updating the Q values, which need time $O(NL)$. This describes that even RPMBGNP incorporates the additional technique of RL, but it does not increase the time complexity of conventional MLE-based PMBGNP.

V. Experimental Study

To testify to the effectiveness in the problem of controlling the agents' behavior, the proposed algorithms are applied to a

TABLE I
JUDGMENT AND PROCESSING FUNCTIONS FOR THE TILEWORLD SYSTEM

Function	Symbol	Description	No. of Arguments	Content of Arguments	
J_1	JF	Judge Forward	5	1: Floor	2: Obstacle
J_2	JB	Judge Backward		3: Tile	4: Hole
J_3	JL	Judge Left			
J_4	JR	Judge Right		5: Agent	
J_5	DT	Judge the Direction of the nearest Tile from the agent	5	1: Forward	2: Backward
J_6	DH	Judge the Direction of the nearest Hole from the agent		3: Left	4: Right
J_7	DHT	Judge the Direction of the nearest Hole from the nearest Tile		5: Cannot find	
J_8	DST	Judge the Direction of the Second nearest Tile from the agent			
P_1	MF	Move Forward	0	–	
P_2	TL	Turn Left			
P_3	TR	Turn Right			
P_4	ST	Stay			

benchmark testbed—Tileworld system [21]—for comparison with traditional ones. Another experimental study on a mobile robot control can be found in the supplementary material.

A. Tileworld System

The Tileworld system [21] is a well-known parameterized environment to investigate the performance of intelligent agents. It consists of a grid of cells on which various objects could exist, including agents, floors, obstacles, tiles, and holes. It has been widely used for the development of intelligent agents and multiagents [13], [17], [56], [57].

The Tileworld system used in this paper is designed by a 12×12 2-D grid world; an example is shown in Fig. 4. In the world, the agent is capable of judging the environment around it. By taking the appropriate actions based on the returned results, the agent can move to its neighboring cells. The agent can push a tile to its forward cell by the action of move forward when there is no obstacle in the forward cell of the tile. Once a tile is pushed into a hole, this tile and hole disappear and the corresponding cell becomes a floor.

In this paper, the environment of the Tileworld system consists of three agents: tiles, and holes that are positioned in the Tileworld. In every step, three agents are controlled for movement simultaneously. The agent is designed to have eight sensor abilities to judge the environment around it, as shown in Table I. By the judgment functions J_1–J_4, the agent can perceive the contents of its neighboring cells. The judgment functions J_5–J_8 are designed to perceive the direction information of the tiles and holes. The agent can recognize four directions of it in the world. These four directions are defined as shown in the example in Fig. 4(b).

Based on the returned arguments, the agent can take the following four processing actions for movements: Move Forward, Turn Left, Turn Right, and Stay (see Table I). As a result, the programs of controlling the agents can be formulated by the combinations of these judgment sensors and processing actions.

B. Fitness Function and Experimental Environments

To evaluate the agents' behavior, the following three factors are taken into account: given constrained steps, the tiles should be pushed into 1) as many holes as possible and 2) using as few steps as possible, and 3) if there are remaining tiles, they should be pushed toward the holes as close as possible. As a result, the evaluation function of the solutions for the Tileworld system is formulated as

$$f = c_t \times \underbrace{\mathrm{DT}}_{1)} + c_s \times \underbrace{\Delta\mathrm{ST}}_{2)} + c_d \times \underbrace{\left[\sum_{t \in \mathrm{Tile}} \Big(D(t) - d(t)\Big)\right]}_{3)} \quad (11)$$

and ΔST denotes the remaining steps calculated by

$$\Delta\mathrm{ST} = (\mathrm{ST} - S_{\mathrm{used}})$$

where
DT: number of tiles that have been pushed into the holes.
ST: user-defined constrained steps.
S_{used}: number of steps that have been used.
Tile: set of suffixes of tiles.
$D(t)$: original distance from tile t to its nearest hole.
$d(t)$: distance from tile t to its nearest hole after ST steps.
c_t, c_s, c_d: parameters of controlling the weights of factor 1), 2), and 3), respectively.

As denoted in (11), the three marked terms correspond to factors 1), 2), and 3) discussed above. In this paper, ST is set at 60. To balance the effects of factors 1), 2), and 3), the parameters are set at c_t=100, c_s=3, and c_d=20.

Ten worlds are used to perform the experimental environments, as shown in Fig. 5. In these cases, the numbers of agents, tiles, and holes are set at 3. The positions of the agents, obstacles, and holes remain the same, while the positions of the tiles vary case by case in order to obtain a general strategy that could control the agent to handle almost the same grid world. The goal of this experiment is to find a strategy that can obtain the highest evaluation values to control the agents in these 10 environments. As a result, the fitness function of the Tileworld system used in this paper is

$$\mathrm{Fitness} = \sum_{w=1}^{10} f(w) \quad (12)$$

where w: ID of the world environments of Tileworld system.

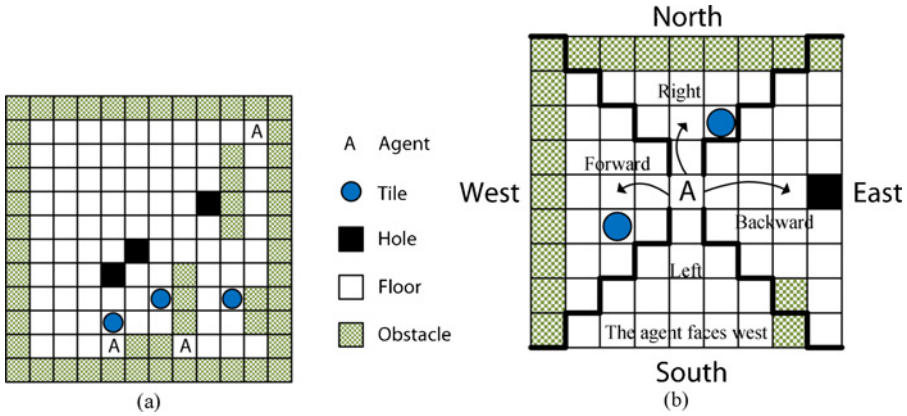

Fig. 4. (a) Tileworld system (12 × 12 2-D grid of cells). (b) Directions that the agent can recognize in the Tileworld system. The figure shows that when the agent faces west, the two tiles are located in its forward and right directions, while the hole is in its backward direction.

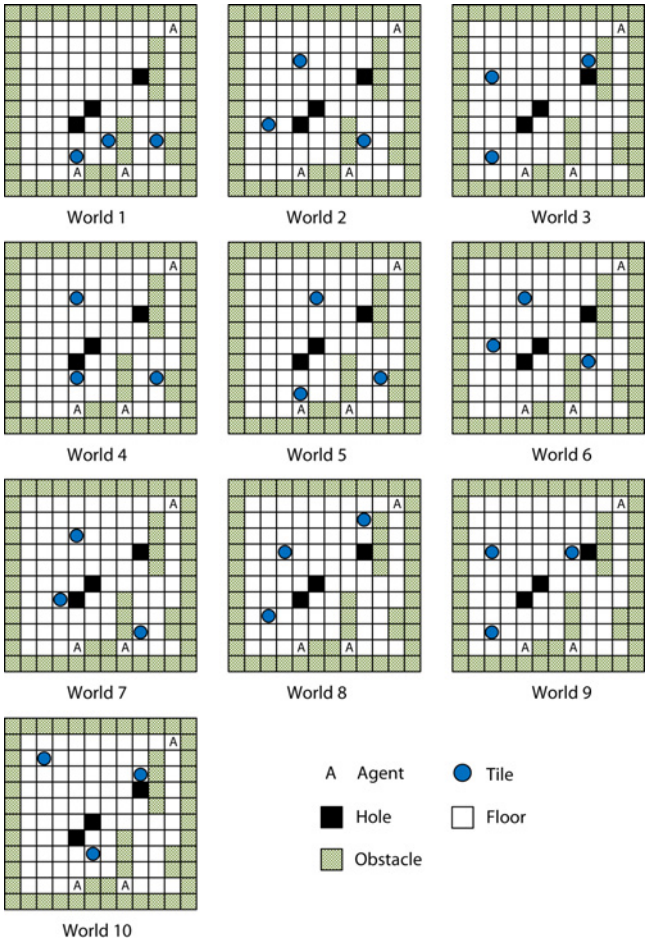

Fig. 5. Ten worlds of the Tileworld system used in this paper.

C. Compared Algorithms and Experimental Settings

In order to verify the effectiveness of the proposed algorithms, the following classical algorithms are selected from the literature of EC, EDA, and RL for comparison.

GNP [12], [13] and GP [58] are the representative algorithms for comparing this study with conventional EAs. GNP uses the directed graph structure described in Section III to represent its programs, while GP uses the tree structure for its programs.

TABLE II
PARAMETER SETTINGS FOR THE TILEWORLD SYSTEM

	GNP	GP	PIPE	EDP	Sarsa	PMBGNP	RPMBGNP
Population size M	300	300	200	1800	—	1800	1800
– elite ind.	1	1	1	100	—	100	100
– crossover ind.	120	120	—	—	—	—	—
– mutation ind.	179	179	—	—	—	—	—
– promising ind. N	—	—	1	900	—	900	900
Program size $\lvert N_{\text{node}}\rvert$	60	781	781	781	—	60	60
Tournament size	2	2	—	—	—	—	—
Crossover rate	0.1	0.9	—	—	—	—	—
Mutation rate	0.01	0.01	—	—	—	—	—
learning rate	—	—	0.02	0.1	0.2	—	0.2
Other parameters							
— PIPE	(fitness constant) = 1 (elitist update probability) = 0						
— Sarsa	(discount factor) γ=0.9 (ε-greedy policy) ε=0.1						
— PMBGNP	η = 0.02 τ = 200						
— RPMBGNP	(discount factor) γ = 0.9						

Probabilistic incremental program evolution (PIPE) [32], a univariate PMBGP, which uses probabilistic prototype tree (PPT) to represent its probabilistic model, and pairwise model based-estimation of distribution programming (EDP) [33] are selected for comparing this study with conventional EDAs.

Sarsa [24] is selected as a state-of-the-art RL algorithm for the comparison with the proposed algorithms. In Sarsa, the states are defined by the observations that the agent can possibly see in its four adjacent cells of the grid. There are total 625 states, four actions, and 625 × 4 = 2500 state–action pairs for Sarsa. ε-greedy policy (with $\varepsilon = 0.1$ to balance the exploitation and exploration appropriately) is used for the selection of actions.

All the settings are listed in Table II, which are determined by hand tuning to perform the best results of each algorithm, i.e., finding the best fitness under the maximum number of fitness evaluations. The discussion of each algorithm, their detailed experimental settings, and the degree of parameter tuning are reported in the supplementary material.

D. Experimental Results and Analysis

1) *Simulation I (Case Study in a Single World):* In the first experiment, we compare the algorithms in a single world of the Tileworld system. Here, World 1 of Fig. 5 is used to perform the experiment. In this case, the fitness is calculated

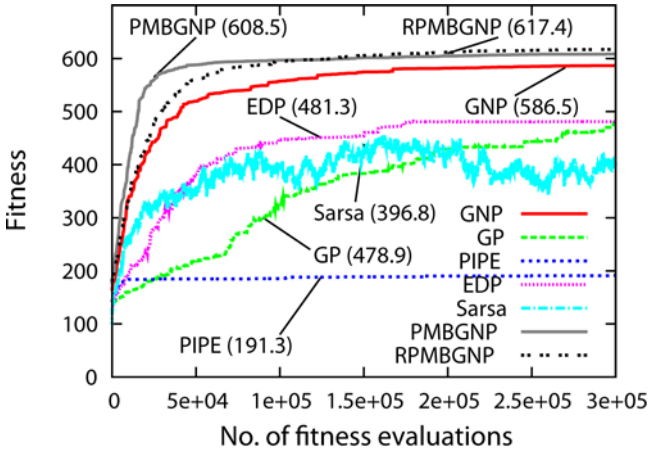

Fig. 6. Fitness curves of a single Tileworld system (Simulation I).

by the evaluation of World 1, which means Fitness = $f(1)$. In this world, theoretically the agents need at least 15 steps to push all tiles into holes. As a result, the maximum fitness value of World 1 can be calculated by

$$\begin{aligned} f_{\max}(1) &= 100 \times 3 + 3 \times (60 - 15) + 20 \times \\ &\quad \Big[(2-0) + (3-0) + (5-0)\Big] \\ &= 635. \end{aligned} \tag{13}$$

The objective of this experiment using a single world is to investigate the evolution/learning behavior of each algorithm, and whether they can find the optimal strategy to control agents successfully.

The terminal condition for each algorithm is the maximum number of fitness evaluations, where 300 000 is used. All the experimental results are the average of 30 independent trials. These settings are also used in the rest of the experiments of this paper.

Fitness results: The fitness curves of the compared algorithms are shown in Fig. 6 and the detailed fitness results are described in Table III. The results show the following.

Graph structures of GNP achieve better performance than that of tree structures of GP. From Fig. 6, we find that even though GNP has a smaller number of program size (60 for GNP, and 781 for GP with the maximum tree depth 4), it can achieve much better fitness results than that of GP. On the other hand, it is natural that GP programs have a higher expression ability as the maximum tree depth increases. However, GP programs will also increase the computation time and memory exponentially (the discussion of GP under different maximum tree depths is presented in the supplementary material). The same phenomenon holds in the comparison between the proposed algorithms and PIPE/EDP.

It is clear from the simulation results that the algorithms using EDA show better fitness results than that of conventional EAs. However, the only exception can be found in PIPE, where it achieves worse performance than GP. This is due to the fact that PPT of PIPE can only estimate the distribution of functions in each node, which cannot cover the interactions between judgment and processing functions of the Tileworld system.

Sarsa achieves worse fitness results than the other algorithms except for PIPE. This is because the most important lack of RL is that the optimal strategy is hardly learned when the state–action space (Q table size) becomes large. In this problem, the Q table size is 2500.

Both PMBGNP and RPMBGNP can achieve better results than the others. By incorporating RL, RPMBGNP can find slightly better results than PMBGNP. In RPMBGNP, Sarsa is used to learn the graph structure G. The learning of Q values is biased. That is, truncation selection is employed to select the promising individuals for the updating of Q values. Moreover, the learning of Q values is used to enhance the quality of node connections in G, which is different from that of Sarsa used as a control strategy. The results show that the Q values can be learned to measure the quality of node connections appropriately. By incorporating them into the probabilistic modeling, we can construct a probabilistic model by considering the interactions between agents and environments. On the other hand, we can find from the curves that RPMBGNP has a little lower evolution speed than PMBGNP in early generations, since the updating of the Q values plays the most important role in the evolution of RPMBGNP, which requires more time to gather sufficient information for probabilistic modeling. However, in later generations, RPMBGNP can continuously evolve to find better results.

Overall, as shown in the column of t-test 1 of Table III, t-test (two tailed, paired) results show the statistically significant differences between the proposed algorithms and the other algorithms. In the meantime, although RPMBGNP can obtain slightly better results than PMBGNP, it is found that there is no statistical difference between them. This is because for the simple problem of Simulation I, both PMBGNP and RPMBGNP can successfully push all the tiles into the holes using as few steps as possible.

Required fitness evaluations (RFEs) and reliability: The RFEs are further counted to test the search speed. The RFEs are calculated in the following way: for the successful trials that the agents can push all three tiles into the holes within ST (=60) steps, the exact RFE is used; for the failed trials that the agents cannot push all three tiles into the holes even after ST steps, the maximum fitness evaluations, i.e., 300 000, are counted. Afterward, the average value of 30 independent trials is calculated to form the final RFEs.

The results of RFEs for Simulation I are shown in Table III. The results show that on average PMBGNP can successfully push all the tiles into the holes with the fastest speed, while RPMBGNP stands in the second rank that requires slightly larger FEs. The t-test 2 results describe the statistically significant difference between PMBGNP/RPMBGNP and the other algorithms from the perspective of RFEs. Overall, from the perspective of reliability, the two proposed algorithms can succeed in all 30 independent trials, while GNP fails in two trials. EDP achieves the best result among all GP variants, while PIPE fails in all trials.

2) *Simulation II (Case Study in the Ten worlds):* In this experiment, the ten worlds shown in Fig. 5 are used to perform the comparison. The fitness is calculated by (12). This experiment is much more complex than that of Simulation

TABLE III
RESULTS OF A SINGLE TILEWORLD SYSTEM (SIMULATION I)

	Fitness (standard deviation)	suc%[1]	Rank	t-test 1[2]	RFEs (standard deviation)	t-test 2
GNP	586.5 ± 69.8	93.3	3	2.06e−01 **4.04e−02**	56 160 ± 73 983	**3.69e−03** **4.86e−02**
GP	478.9 ± 151.6	40.0	5	**4.03e−05** **2.07e−05**	238 090 ± 90 072	**5.89e−14** **9.64e−13**
PIPE	191.3 ± 18.0	0.0	9	**1.34e−26** **2.96e−36**	300 000 ± 0	**1.34e−59** **1.12e−35**
EDP	481.3 ± 99.9	76.7	4	**2.58e−06** **3.39e−08**	172 720 ± 101 916	**2.74e−09** **4.73e−08**
Sarsa	396.8 ± 114.3	26.7	6	**5.17e−10** **1.85e−11**	268 761 ± 66 519	**5.52e−19** **3.09e−18**
PMBGNP	608.5 ± 54.4	100.0	2	— 4.32e−01	12 600 ± 2865	— **4.28e−05**
RPMBGNP	617.4 ± 20.2	100.0	1	— —	28 320 ± 18 137	— —

t-test 1 analyzes the statistical difference of fitness results between PMBGNP/RPMBGNP and the other algorithms, while t-test 2 is studied by RFEs.
1 suc%: perc. that all three tiles can be pushed into the holes within ST steps in 30 independent trials.
2 the upper value is the p-value of t-test for PMBGNP, while the lower value is that for RPMBGNP. The bold value denotes there is statistically significant difference.

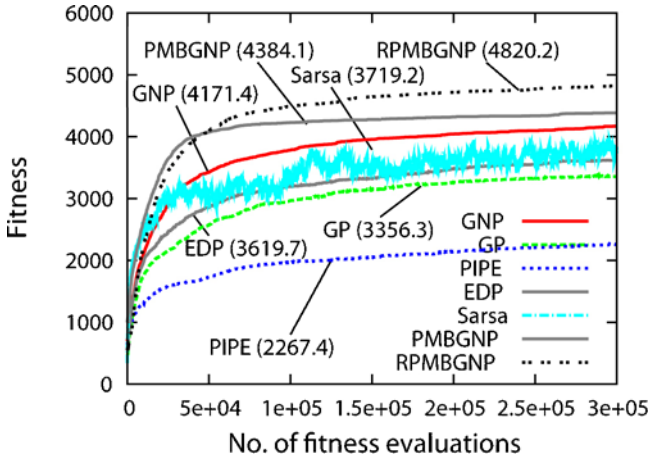

Fig. 7. Fitness curves of 10 Tileworld systems (Simulation II).

TABLE IV
RESULTS OF TEN TILEWORLD SYSTEMS (SIMULATION II)

	Fitness (std. dev.)	DT	perc.%[†]	Rank	t-test
GNP	4171.4 ± 692.0	23.0 ± 3.8	76.6	3	2.06e−01 **1.20e−04**
GP	3356.3 ± 679.2	17.6 ± 3.5	58.7	6	**2.44e−06** **2.04e−12**
PIPE	2267.4 ± 338.0	11.5 ± 2.4	38.4	9	**3.05e−15** **1.12e−21**
EDP	3619.7 ± 653.3	18.3 ± 4.0	61.0	5	**4.09e−05** **4.04e−11**
Sarsa	3719.2 ± 1106.2	18.9 ± 5.1	62.9	4	**1.72e−02** **2.68e−05**
PMBGNP	4384.1 ± 735.5	24.2 ± 3.7	80.7	2	— **7.20e−03**
RPMBGNP	4820.2 ± 495.0	26.5 ± 2.2	88.4	1	— —

[†] perc.%: perc. that the tiles has been pushed into the holes within ST steps in 30 independent trials. (For each trial, perc.%=100 × $\sum_{w=1}^{10}$ DT/30.)

I, since we need to find a strategy that can handle all the ten worlds well. It is almost impossible to explicitly know the expected maximum fitness value of this experiment. The objective of Simulation II is to illustrate the performance of the proposed algorithms in complex Tileworld systems.

The fitness curves of the compared algorithms are shown in Fig. 7, and the detailed fitness results are described in Table IV. The results show similar conclusions to Simulation I, where the proposed algorithms, i.e., PMBGNP and RPMBGNP, can obtain the highest fitness after 300 000 fitness evaluations.

From the fitness curves we can explicitly find that the probabilistic modeling allows PMBGNP to have faster convergence behavior than that of GNP. On the other hand, RPMBGNP has slower convergence speed than PMBGNP in early generations by incorporating RL; however, it gradually finds higher fitness values in later generations. As for GP and its variants, the results show smoother curves than that of Simulation I. This is because Simulation II uses ten worlds to perform the results, where the fitness landscape becomes more smooth. The results report that EDP can obtain the best fitness values, in comparison to the other GP variants, while PIPE performs the worst. The t-test results show the significant difference between the proposed algorithms and the others.

From the perspective of dropped tiles (DTs), the results report that the proposed algorithms can push as many tiles into the holes as possible within ST steps. On average the proposed algorithms can push the most tiles into the holes comparing with the other algorithms. As Simulation II reports the results for the complex problems of the Tileworld system, RPMBGNP can achieve better performance than PMBGNP in terms of solution quality (fitness values) and reliability (successful rate). This is because a higher evolution ability is required to find the acceptable solutions in complex problems. As a result, from the viewpoint of reliability, RPMBGNP can successfully push 88.4% tiles into the holes, while PMBGNP can succeed in pushing 80.7% tiles.

E. Effects of Different Methods and Parameters

The fundamental basis of the proposed algorithms to apply EDA to evolve the directed graph structure relies on the

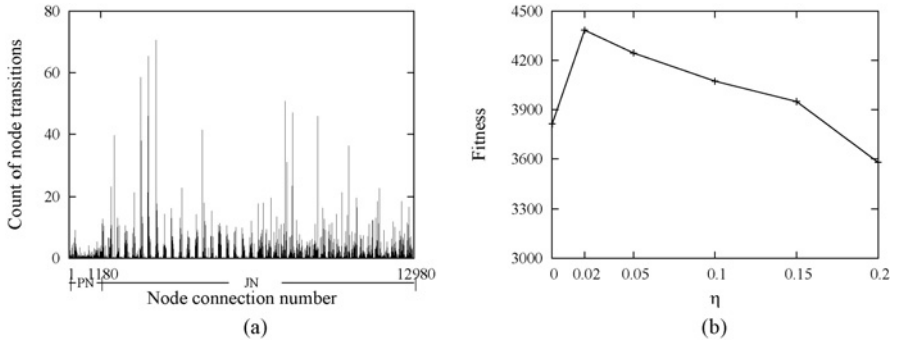

Fig. 8. Effect of η in Simulation II of the Tileworld system. (a) Count of node transition. (b) Fitness results with different η.

following three methods: MLE by combining both of the information of node connections and transitions, Boltzmann distribution, and RL. The following experiments address the effects of different methods by the discussion of their parameters.

1) *Node Connections and Transitions:* One advantage of the directed graph structure used in PMBGNP is the reusability of node connections. That is, the important node connections are preferred to be reused/transitted frequently; as an example, Fig. 8(a) reports the count of node transitions for each node connection in the elite individual of Simulation II.

In PMBGNP, one important feature is that its probabilistic modeling considers both the information of node connections and transitions. The node connections help us to learn the graph structures of promising individuals, while the node transitions denote the reusability of node connections. Parameter η plays the role of balancing the effects of these two factors. If η approaches 0, the count of node connections will play the most important role in the probabilistic modeling, while their reusability is not considered. On the other hand, if η is too large, only the highly reused node connections will appear, which will cause the premature convergence. The results of Fig. 8(b) show that η=0.02 can obtain the best result, compared to the other values, due to the appropriate balance of the node connections and transitions. Both the too small (=0, only consider the node connections) and too large (=0.2, pay more attention to the node transitions) η cannot ensure good performance.

2) *Boltzmann Distribution:* Due to the issue of diversity loss, PMBGNP applies Boltzmann distribution to its probabilistic modeling, whose effect is influenced by the parameter τ. With higher τ, PMBGNP copes with more exploration where its probabilistic model becomes uniform distribution. With lower τ, PMBGNP tends to change its probabilistic model to greedy distribution. As a result, the appropriate setting of τ can achieve the tradeoff between the exploration and exploitation. Fig. 9(a) shows the fitness values of different τ. It shows that when τ is small, i.e., 0.1, the fitness value becomes very small due to the premature convergence by greed distribution. On the other hand, with high τ, more exploration causes the slow evolution speed, leading to bad fitness values. In this experiment, τ=200 is the appropriate setting to obtain the best result.

The fitness curves of Fig. 9(b) show that pure MLE-based probabilistic modeling causes the premature convergence due to the sensitive diversity loss of PMBGNP (*Theorem 1*), while incorporating the Boltzmann distribution allows PMBGNP and finds a better result.

3) *RL:* As discussed in Section IV-D, η is removed from the probabilistic model, where the Q values play the role of automatically counting the information of both node connections and transitions. As a result, the learning rate α and discount factor γ should be configured to control the effect of RL.

Too small α will cause the slow learning speed for updating Q values (α=0 will make the agent not learn anything), while too large α will lead to the big update of Q values causing the unstable learning. The appropriate α should be set to update Q values gradually. As the results reported in Fig. 10(a), α=0.2 can achieve the highest fitness values, compared to the other values.

The discount factor γ determines how much future reward is taken into account for the updating of Q values. Small γ denotes that the agent only cares about the immediate reward obtained by the taken action, while high γ causes Q values to be updated by more strongly counting future rewards. Empirically speaking, for complex problems, long-term future rewards should be counted more seriously since finding the optimal solution requires us to consider not just the current action but also the consequent future actions. However, if γ approaches 1, the Q values may diverge. Fig. 10(b) reports the results with different γ, where γ=0.9 performs best.

F. Computation Time

Table V shows the computation time of each algorithm in Simulation II of the Tileworld system. All experiments are carried out on a PC with Intel Core i5 running at 2.80 GHz with 4 GB RAM. The complier is Visual Studio 2010 of Windows 7.

Sarsa requires the least computation time, since the updating of the Q values only needs linear time cost w.r.t. the maximum steps, i.e., ST. GNP is faster than GP, since GP has many more nodes. EDA-based algorithms need more time than conventional EAs due to the additional time for the estimation of probability distribution. The computation time is proportional to the accuracy of the probabilistic model. That is, learning a probabilistic model that can capture more complex variable interactions requires more time. Consequently, EDP requires more time than PIPE as it needs to maintain more probabilities.

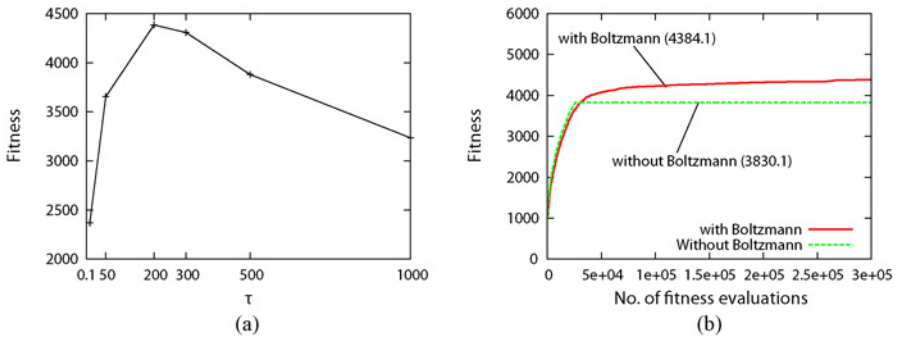

Fig. 9. Effect of Boltzmann distribution in Simulation II of the Tileworld system. (a) Effect of τ. (b) Effect of Boltzmann distribution.

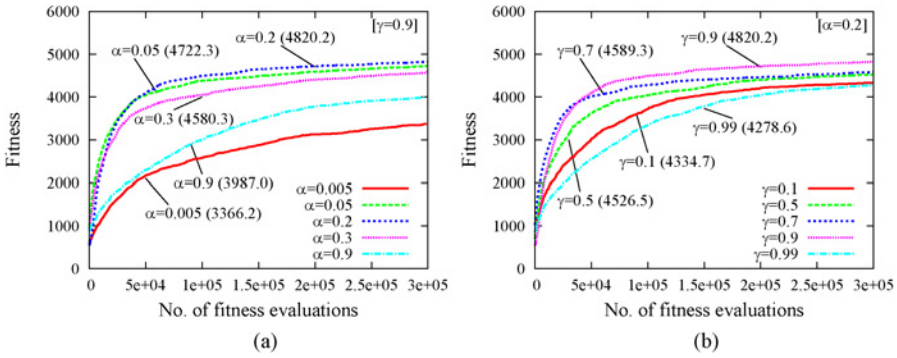

Fig. 10. Effects of α and γ in Simulation II of the Tileworld system.(a) Fitness results with different α. (b) Fitness results with different γ.

TABLE V
COMPUTATION TIME IN SIMULATION II OF THE TILEWORLD SYSTEM

	GNP	GP	PIPE	EDP	Sarsa	PMBGNP	RPMBGNP
Computation Time (Unit: sec.)	367.3	696.9	760.0	1081.2	284.6	429.5	482.0
Rank	2	5	6	7	1	**3**	**4**

TABLE VI
FITNESS RESULTS IN NEW ENVIRONMENTS OF THE TILEWORLD SYSTEM

	GNP	GP	PIPE	EDP	Sarsa	PMBGNP	RPMBGNP
Change tile positions	2282.8 ±944.9	1376.4 ±781.0	1148.4 ±870.0	1435.0 ±865.2	1903.8 ±864.1	2466.8 ±823.3	2770.8 ±944.9
Change tile and hole positions	552.0 ±860.1	219.0 ±830.0	203.8 ±786.6	244.2 ±850.6	324.4 ±641.4	594.8 ±833.8	650.0 ±918.6
Rank	3	6	7	5	4	**2**	**1**

As for the proposed algorithms, the results show that even integrating RL, RPMBGNP only requires similar computation time to PMBGNP. This could be explained by the discussion presented in Section IV-D, where their time complexities are at the same level.

G. Generalization Ability

To simply test the generalization ability of each algorithm, we applied the 30 best solutions found by Simulation II to the new environments. Two experiments are carried out, where one randomly changes the tile positions of the ten worlds in Fig. 5, and another randomly changes both of the tile and hole positions. Each best solution is applied to the new worlds and run 1000 times to calculate the average results. The results are reported in Table VI.

As in Fig. 5, the best solutions are trained and obtained by the ten worlds with different tile positions. Consequently, for the testing environments with random tile positions, the trained solutions perform quite well to achieve robust results. On the other hand, when randomly changing both of the tile and hole positions, the trained solutions show the lack of robustness. Based on the comparative study, PMBGNP and RPMBGNP show higher generalization ability than the others.

H. Experimental Analysis on Mobile Robot Control

Besides solving the benchmark problem of the Tileworld system, we further apply the proposed algorithms in a real mobile robot control problem, the Khepera robot [22], [23] control.

In this paper, the robot is controlled to solve the wall-following problem [59] by using different compared algorithms. Five problems with different problem sizes are tested to confirm the scalability of the proposed algorithms. The detailed experimental results and analysis are presented in the supplementary material. The results show that the proposed algorithms are capable of achieving better performance than the compared algorithms in terms of fitness values, search speed, reliability, and scalability.

Considering these experiments as a whole, we have evaluated the performance of this paper in both of the benchmark tested and a real case of the problems of controlling the agents' behavior.

VI. DISCUSSION

We have tested the proposed algorithms in the problems of controlling the agents' behavior, where a benchmark testbed—

Tileworld system—and a real robot control (shown in the supplementary material) are selected as the representatives. The results in these two problems show that the proposed algorithms outperform the classical algorithms of EAs, EDAs, and RL. Fundamentally, there are two main reasons to support the performance: 1) the proposed algorithms use GNP's directed graph to represent their individuals. Thus, compared to the GP variants, the expression ability of the individuals is increased to model some complex problems, such as the ones studied in this paper. 2) The proposed algorithms construct probabilistic models by learning the promising individuals, which are used to replace the stochastic genetic operators of EAs to generate new solutions.

PMBGNP is derived from the univariate EDAs, which indicate that its probabilistic model is a univariate model, as the one of PIPE. However, PIPE models the probabilities of functions in each node independently, which cannot capture any interaction between the functions/actions. On the other hand, PMBGNP represents its univariate model by the probabilities of node connections. Although the node connections are learned independently, the probability itself can represent the relation between two nodes of the directed graph. Consequently, a kind of relation between judgment and processing functions can be captured by the univariate model of PMBGNP. Such "if-then" decision-making relations are essentially helpful to solve the problems of controlling the agents' behavior. As a result, PMBGNP and EDP (using conditional probabilities to model the pairwise interactions) have significant better performance than PIPE.

Besides the MLE-based method, many advanced EDAs have applied a Bayesian network or some other techniques to learn the promising individuals. However, these algorithms are computationally expensive. On the other hand, this paper introduces RL to learn the promising individuals. In RPMBGNP, RL is carried to learn the experience of individual executions, and the formulated Q values are used to the probabilistic modeling. As the experimental results address, RPMBGNP shows the significant improvement of fitness values, search speed, and reliability. Meantime, the computation time is still kept in the same level as PMBGNP.

It is natural that integrating RL will benefit the proposed algorithm in the problems of controlling the agents' behavior, since such problems can be viewed as the cases of RL problems [24]. However, whether the proposed algorithm of integrating RL can be explored to solve different sorts of problems is still unknown. The effectiveness of integrating RL can be studied by applying it to more general problems and extending it to the other EDAs, which inspires one remaining issue of our future work.

VII. Conclusion and Future Work

A novel EDA called PMBGNP has been proposed in this paper. PMBGNP extends the existing string and tree-based EDAs to a more complex representation—graph-based chromosome. With the unique graph structure, PMBGNP ensures higher expression ability to model the problems of controlling the agents' behavior. In addition, an extended algorithm called RPMBGNP has been also introduced by integrating PMBGNP and RL.

To evaluate the effectiveness of the proposed algorithms, a benchmark testbed called the Tileworld system and a real mobile robot control have been selected as the typical problems for our study. We compared the proposed algorithms with the classical algorithms from the literature of EC, EDA, and RL. The experimental results showed that PMBGNP inherits both of the features of the graph structure and EDA to achieve higher expression and evolution ability than the conventional algorithms. On the other hand, by introducing a new method of integrating EDA and RL, we showed that the performance of fitness values, search speed, and reliability can be significantly improved.

Since this research is the first attempt to study EDA in the graph chromosome structure, it would be interesting to take more in-depth studies to study PMBGNP in both theory and practice in the future, such as extending PMBGNP to solve the continuous optimization and dynamic optimization problems. On the other hand, improving the method of integrating EDA and RL, and extending it to the other EDAs, will be another piece of our future research.

Appendix

Proof of Theorem 1

The diversity loss rate of PMBGNP can be defined by

Definition 5 (Diversity loss rate): The diversity loss rate $\mathrm{DL}(b(i), j)$ of a node connection $(b(i), j)$ in the current generation is represented by the probability that the node connection $(b(i), j)$ is not sampled in the next generation.

It can be calculated by

Lemma 1: Given the population with total M individuals, $\mathrm{DL}(b(i), j)$ is calculated by

$$\mathrm{DL}(b(i), j) = \Big(1 - P(b(i), j)\Big)^M. \tag{14}$$

Lemma 1 can be easily proven and $\mathrm{DL}(b(i), j)$ is ranging at (0, 1]. In PMBGNP, once the probability $P(b(i), j)$ is equal to 0, the corresponding node connection $(b(i), j)$ will never be sampled in the future generations, which can say that its diversity is lost and $\mathrm{DL}(b(i), j)$=1.

By discussing its diversity loss rate, the required population size M of PMBGNP can be estimated. In the simplest way, we can obtain M by discussing its lower/upper bound of required sample size N (the number of best individuals in truncation selection) in the initial generation. Therefore, we have

$$M \in \left[\frac{1}{P(b(i^*), j^*)}\infty\right) \tag{15}$$

where $P(b(i^*), j^*)$ is the initial value of the probabilistic model, which is equal to $1/(|N_{\mathrm{node}}| - 1)$. Theoretically at least $|N_{\mathrm{node}}| - 1$ individuals should be obtained to sample all node connections in the population. However, simply obtaining the individuals with the lower bound cannot actually estimate an accurate model, and it is impractical to obtain infinite individuals to construct the perfect model. Therefore,

M is generally determined by the problems. Nevertheless, since the probability distribution of PMBGNP is actually the multinomial distribution, we can investigate M by confidence interval [8], in which the lower bound of M can be represented. Following this point of view, we can obtain M by

Lemma 2: Given a confidence level ϵ ($\epsilon \ll 1$) that defines the acceptable diversity loss rate, the required population size M of PMBGNP satisfies

$$M \geq \frac{\ln \epsilon}{\ln\left(1 - \frac{1}{|N_{\text{node}}|-1}\right)}. \tag{16}$$

Proof: With *Lemma 1*, for a given node connection $(b(i), j)$, we have $\text{DL}(b(i), j) = (1 - P(b(i), j))^M \leq \epsilon$. After the logarithmic process and using the initial value of $P(b(i), j)$, we can obtain (16). ■

Similarly, the required population sizes of univariate EDAs, i.e., UMDA and PIPE, can be derived using their initial probabilities ($1/2$ in the case of UMDA and $1/\phi$ in the case of PIPE[1]). Therefore, with *Lemma 2*, we can obtain the lower bound of M of PMBGNP, UMDA, and PIPE by

$$M_{\text{PMBGNP}} = \frac{\ln \varepsilon}{\ln\left(1 - \frac{1}{|N_{\text{node}}|-1}\right)} \tag{17}$$

$$M_{\text{UMDA}} = \frac{\ln \varepsilon}{\ln\left(1 - \frac{1}{2}\right)} = \frac{\ln \varepsilon}{\ln \frac{1}{2}} \tag{18}$$

$$M_{\text{PIPE}} = \frac{\ln \varepsilon}{\ln\left(1 - \frac{1}{|\phi|}\right)}. \tag{19}$$

The number of nodes $|N_{\text{node}}|$ in PMBGNP is much larger than 2 and $|\phi|$, i.e., 60 [13] or more [17]. Therefore, given the same confidence level, we can obtain the following relationship by (17)–(19): $M_{\text{PMBGNP}} > M_{\text{PIPE}} > M_{\text{UMDA}}$. Consequently, *Theorem 1* can be drawn.

REFERENCES

[1] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-94-163, 1994.

[2] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions I. Binary parameters," in *Proc. Conf. Parallel Problem Solving Nature*, 1996, pp. 178–187.

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

[4] Q. Zhang and H. Mühlenbein, "On the convergence of a class of estimation of distribution algorithms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 127–136, Apr. 2004.

[5] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "Linkage problem, distribution estimation, and Bayesian networks," *Evol. Comput.*, vol. 8, no. 3, pp. 311–341, 2002.

[6] Q. Zhang, J. Sun, and E. Tsang, "An evolutionary algorithm with guided mutation for the maximum clique problem," *IEEE Trans. Evol. Comput.*, vol. 9, no. 2, pp. 192–200, Apr. 2005.

[7] R. Santana, P. Larrañaga, and J. A. Lozano, "Protein folding in simplified models with estimation of distribution algorithms," *IEEE Trans. Evol. Comput.*, vol. 12, no. 4, pp. 418–438, Aug. 2008.

[1]In this case, we assume that PIPE uses the way of UMDA to construct its probabilistic model. ϕ represents the number of functions/symbols in each node of GP's tree structure, where generally $|\phi| = 10$ [8].

[8] Y. Hasegawa and H. Iba, "A Bayesian network approach to program generation," *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 750–764, Dec. 2008.

[9] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A survey of optimization by building and using probabilistic models," *Comput. Optimization Applicat.*, vol. 21, no. 1, pp. 5–20, 2002.

[10] Y. Shan, R. I. McKay, D. Essam, and H. A. Abbass, "A survey of probabilistic model building genetic programming," in *Scalable Optimization via Probabilistic Modeling*, M. Pelikan, K. Sastry, and E. Cantú-Paz, Eds. Berlin, Germany: Springer-Verlag, 2006, ch. 6, pp. 121–154.

[11] 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. IEEE Congr. Evol. Comput.*, Jul. 2010, pp. 2673–2680.

[12] K. Hirasawa, M. Okubo, H. Katagiri, J. Hu, and J. Murata, "Comparison between genetic network programming (GNP) and genetic programming (GP)," in *Proc. IEEE Congr. Evol. Comput.*, May 2001, pp. 1276–1282.

[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] A. Teller and M. Veloso, "PADO: Learning tree structured algorithms for orchestration into an object recognition system," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-95-101, 1995.

[15] R. Poli, "Parallel distributed genetic programming," School Comput. Sci., Univ. Birmingham, West Midlands, U.K., Tech. Rep. CSRP-96-15, 1996.

[16] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Proc. Eur. Conf. Genetic Programming*, 2000, pp. 121–132.

[17] T. Eguchi, K. Hirasawa, J. Hu, and N. Ota, "A study of evolutionary multiagent models based on symbiosis," *IEEE Trans. Syst., Man, Cybern. B*, vol. 36, no. 1, pp. 179–193, Feb. 2006.

[18] K. Shimada, K. Hirasawa, and J. Hu, "Genetic network programming with acquisition mechanisms of association rules," *J. Advanced Comput. Intelligence Intelligent Informat.*, vol. 10, no. 1, pp. 102–111, 2006.

[19] 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. Syst., Man, Cybern. C*, vol. 38, no. 4, pp. 535–550, Jul. 2008.

[20] S. Mabu, C. Chen, N. Lu, K. Shimada, and K. Hirasawa, "An intrusion detection model based on fuzzy class association rule mining using genetic network programming," *IEEE Trans. Syst., Man, Cybern. C*, vol. 41, no. 1, pp. 130–139, Jan. 2011.

[21] M. E. Pollack and M. Ringuette, "Introducing the Tile-world: Experimentally evaluating agent architectures," in *Proc. Conf. Amer. Assoc. Artificial Intelligence*, 1990, pp. 183–189.

[22] *Webots Software* [Online]. Available: http://www.cyberbotics.com/

[23] *Khepera Robot* [Online]. Available: http://www.k-team.com/

[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[25] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 287–297, Nov. 1999.

[26] J. S. D. Bonet, C. L. Isbell, and P. Viola, "MIMIC: Finding optima by estimating probability densities," in *Proc. Advances Neural Informat. Process. Syst.*, 1997, pp. 424–430.

[27] P. A. N. Bosman and D. Thierens, "Linkage information processing in distribution estimation algorithms," in *Proc. Genetic Evol. Comput. Conf.*, 1999, pp. 60–67.

[28] M. Pelikan and H. Mühlenbein, "The bivariate marginal distribution algorithm," in *Advances in Soft Computing: Engineering Design and Manufacturing*. Berlin, Germany: Springer, 1999, pp. 521–535.

[29] H. Mühlenbein and T. Mahnig, "FDA—A scalable evolutionary algorithm for the optimization of additively decomposed functions," *Evol. Comput.*, vol. 7, no. 4, pp. 353–376, 1999.

[30] R. Etxeberria and P. Larrañaga, "Global optimization using Bayesian networks," in *Proc. Symp. Artif. Intell.*, 1999, pp. 332–339.

[31] G. R. Harik, F. G. Lobo, and K. Sastry, "Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA)," in *Scalable Optimization via Probabilistic Modeling*, M. Pelikan, K. Sastry, and E. Cantú-Paz, Eds. Berlin, Germany: Springer-Verlag, 2006, ch. 3, pp. 39–61.

[32] R. P. Salustowicz and J. Schmidhuber, "Probabilistic incremental program evolution," *Evol. Comput.*, vol. 5, no. 2, pp. 123–141, 1997.

[33] K. Yanai and H. Iba, "Estimation of distribution programming based on Bayesian network," in *Proc. IEEE Congr. Evol. Comput.*, Dec. 2003, pp. 1618–1625.
[34] K. Sastry and D. E. Goldberg, "Probabilistic model building and competent genetic programming," in *Genetic Programming Theory and Practice*, R. L. Riolo and B. Worzel, Eds. Berlin, Germany: Springer, 2003, vol. 13, pp. 205–220.
[35] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill, "Grammar-based genetic programming: A survey," *Genetic Program. Evolvable Mach.*, vol. 11, no. 3-4, pp. 365–396, 2010.
[36] Y. Shan, R. I. McKay, R. Baxter, H. Abbass, D. Essam, and N. X. Hoai, "Grammar model-based program evolution," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2004, pp. 478–485.
[37] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Published via [Online]. Available: , 2008, (With contributions by J. R. Koza).
[38] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña, "Optimization by learning and simulation of Bayesian and Gaussian networks," Intelligent Systems Group, Dept. of Comput. Sci. Artif. Intell., Univ. Basque Country, Gipuzkoa, Spain, Tech. Rep. EHU-KZAA-IK-4-99, 1999.
[39] P. A. N. Bosman and D. Thierens, "Numerical optimization with real-valued estimation-of-distribution algorithms," in *Scalable Optimization via Probabilistic Modeling*, M. Pelikan, K. Sastry, and E. Cantú-Paz, Eds. Berlin, Germany: Springer-Verlag, 2006, ch. 5, pp. 91–120.
[40] X. Li, S. Mabu, and K. Hirasawa, "Towards the maintenance of population diversity: A hybrid probabilistic model building genetic network programming," *Trans. Japanese Soc. Evol. Comput.*, vol. 1, no. 1, pp. 89–101, 2010.
[41] Q. Zhang, A. Zhou, and Y. Jin, "RM-MEDA: A regularity model-based multiobjective estimation of distribution algorithm," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 41–63, Feb. 2008.
[42] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 12, no. 5, pp. 542–561, Oct. 2008.
[43] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 3–14, Jan. 1994.
[44] R. P. Salustowicz, M. A. Wiering, and J. Schmidhuber, "Learning team strategies: Soccer case studies," *Mach. Learning*, vol. 33, no. 2–3, pp. 263–282, 1998.
[45] H. Handa, "EDA-RL: Estimation of distribution algorithms for reinforcement learning problems," in *Proc. Genetic Evol. Comput. Conf.*, 2009, pp. 405–412.
[46] K. L. Downing, "Reinforced genetic programming," *Genetic Program. Evolvable Mach.*, vol. 2, no. 3, pp. 259–288, 2001.
[47] S. Kamio and H. Iba, "Adaptation technique for integrating genetic programming and reinforcement learning for real robots," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 318–333, Jun. 2005.
[48] X. Li, B. Li, S. Mabu, and K. Hirasawa, "A novel estimation of distribution algorithm using graph-based chromosome representation and reinforcement learning," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2011, pp. 37–44.
[49] T. K. Paul and H. Iba, "Reinforcement learning estimation of distribution algorithm," in *Proc. Genetic Evol. Comput. Conf.*, 2003, pp. 1259–1270.
[50] R. Santana, "Estimation of distribution algorithm with kikuchi approximations," *Evol. Comput.*, vol. 13, no. 1, pp. 67–97, 2005.
[51] S. K. Shakya, "DEUM: A framework for an estimation of distribution algorithm based on Markov random fields," Ph.D. dissertation, School Comput., Robert Gordon Univ., Aberdeen, U.K., Apr. 2006.
[52] A. Brownlee, J. McCall, S. Shakya, and Q. Zhang, "Structure learning and optimisation in a Markov-network based estimation of distribution algorithm," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 447–454.
[53] J. M. Hammersley and P. Clifford, "Markov fields on finite graphs and lattices," 1971, unpublished.
[54] M. Munetomo, N. Murao, and K. Akama, "Introducing assignment functions to Bayesian optimization algorithms," *Informat. Sci.*, vol. 178, no. 1, pp. 152–163, 2008.
[55] X. Li, S. Mabu, and K. Hirasawa, "Use of infeasible individuals in probabilistic model building genetic network programming," in *Proc. Genetic Evol. Comput. Conf.*, 2011, pp. 601–608.
[56] H. Iba, "Emergent cooperation for multiple agents using genetic programming," in *Proc. Conf. Parallel Problem Solving Nature*, 1996, pp. 32–41.
[57] M. E. Pollack, D. Joslin, A. Nunes, S. Ur, and E. Ephrati, "Experimental investigation of an agent commitment strategy," Dept. Comput. Sci., Univ. Pittsburgh, Pittsburgh, PA, Tech. Rep. 94-31, 1994.
[58] J. R. Koza, *Genetic Programming on the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.
[59] P. Nordin, W. Banzhaf, and M. Brameier, "Evolution of a world model for a miniature robot using genetic programming," *Robotics Autonomous Syst.*, vol. 25, no. 1–2, pp. 105–116, 1998.

Xianneng Li (S'10) received the B.E. degree from Nanjing University, Jiangsu, China, in July 2008, and the M.E. and Ph.D. degrees from Waseda University, Tokyo, Japan, in September 2010 and March 2013, respectively.

He was a Research Associate with Waseda University from April 2011 to March 2012. He has been a Research Fellow with the Japan Society for the Promotion of Science (JSPS) since 2012, where he is currently a Post-Doctoral Research Fellow with JSPS from April 2013. His current research interests include evolutionary computation, reinforcement learning, evolutionary robotics, data mining, and computational finance.

Shingo Mabu (S'03–M'07) received the B.E. and M.E. degrees from Kyushu University, Fukuoka, Japan, in 2001 and 2003, respectively, and the Ph.D. degree from Waseda University, Tokyo, Japan, in 2006.

From 2006 to 2007, he was a Visiting Lecturer at the Advanced Research Institute for Science and Engineering, Waseda University, where he has been an Assistant Professor since 2007.

Kotaro Hirasawa (M'91) received the B.E. and M.E. degrees in engineering from Kyushu University, Fukuoka, Japan, in 1964 and 1966, respectively.

From 2002 to 2012, he was a Professor in the Graduate School of Information, Production and Systems, Waseda University, Tokyo, Japan. Since April 2012, he has been an Honorary Professor of Waseda University.