Use of Infeasible Individuals in Probabilistic Model Building Genetic Network Programming

Xianneng Li
Graduate School of Information, Production and Systems
Waseda University, Japan
sennou@asagi.waseda.jp

Shingo Mabu
Graduate School of Information, Production and Systems
Waseda University, Japan
mabu@aoni.waseda.jp

Kotaro Hirasawa
Graduate School of Information, Production and Systems
Waseda University, Japan
hirasawa@waseda.jp

ABSTRACT

Classical EDAs generally use truncation selection to estimate the distribution of the feasible (good) individuals while ignoring the infeasible (bad) ones. However, various research in EAs reported that the infeasible individuals may affect and help the problem solving. This paper proposed a new method to use the infeasible individuals by studying the sub-structures rather than the entire individual structures to solve Reinforcement Learning (RL) problems, which generally factorize their entire solutions to the sequences of state-action pairs. This work was studied in a recent graph-based EDA named Probabilistic Model Building Genetic Network Programming (PMBGNP) which can solve RL problems successfully. The effectiveness of this work is verified in a RL problem, i.e., robot control, comparing with some other related work.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms

Keywords

EDA, probabilistic model building genetic network programming, infeasible individuals, reinforcement learning

1. INTRODUCTION

Various EDAs have been proposed in Evolutionary Computation to draw its success. Despite many different implementations, EDAs can be summarized below: 1) Randomly generate an initial population; 2) Construct the probabilistic model from the feasible (good) individuals; 3) Use the probabilistic model to generate the new population; 4) Go back to step 2) until the terminal conditions.

GECCO'11, July 12–16, 2011, Dublin, Ireland.

Most of the current EDAs use truncation selection to estimate the distribution of the feasible individuals, while ignoring the infeasible ones[1]. However, many studies have reported that utilizing the infeasible individuals would benefit the evolution process in both of EAs [25] and EDAs [2].

In general, there are two ideas to utilize the infeasible individuals in EDAs. The first idea is directly taking into account the infeasible individuals to the construction of probabilistic models. In certain aspects, this work is related to the selection mechanisms of EDAs, since it can be transformed to the problems of *how to select the individuals to be estimated with respect to the true distribution of the search space*. In [2], the authors selected a part of infeasible individuals to the model construction and obtained better performance in some problems, where the probabilistic model is unlikely to perfectly match with the fitness function. However, for many other problems, the infeasible individuals will break the useful information of feasible ones to construct an inaccurate model. Another attempt called EDAs without explicit selection [24, 15] uses the entire population to estimate the probabilistic model, where each individual is given a weight w.r.t. its fitness value. However, one important drawback is that it sometimes does not achieve good performances since it is hard to control the weights of different individuals.

The second idea is to use the infeasible individuals to filter sample errors [14, 7]. [14] divides the entire population to several groups and Bayesian classifiers are built to create new individuals taking into account the characteristics of the best classes and avoiding those of the worst classes. Similarly, [7] proposed a method that estimates two probabilistic models corresponding to the feasible and infeasible individuals, where the feasible model is used to sample new individuals and the infeasible one is applied to filter sample errors. These research showed some advantages of convergence speed over standard EDAs.

This paper proposed a novel method to use infeasible individuals. The point is to extract the useful information of infeasible individuals and utilize them to the probabilistic modeling, where the useful information can be represented as sub-structures of individuals. We can easily observe that if the useful sub-structures are extracted, they can be directly taken into account for the probabilistic modeling,

[1]In the paper, the feasible individuals denote a set of individuals with the highest fitness values in the current generation, while infeasible individuals represent a set of individuals with the lowest fitness values.

which benefits the problem solving. Therefore, in certain aspects, if adopting this idea to EDAs, the selection step of EDAs can be transformed to the problem of *how to identify and extract the useful sub-structures of individuals*. One way to achieve this task is to design the fitness functions which can explicitly identify the useful sub-structures, such as [3]. However, for most problems whose fitness functions are designed to evaluate the quality of the entire individuals, this task becomes hard to achieve.

In the case of Reinforcement Learning (RL) problems, its property that generally factorizes the entire solution to a sequence of state-action pairs makes it possible to identify and extract useful sub-structures, where the part of the state-action pairs can be identified as useful sub-structures. In this paper, a method to extract the useful sub-structures of infeasible individuals is proposed by introducing a RL technique named Sarsa learning [22] to a novel EDA called Probabilistic Model Building Genetic Network Programming (PMBGNP) [11, 10], and it is expected that another viewpoint would be provided to EDAs from this approach. The useful sub-structures are represented as the state-action pairs with higher Q values than the others, where the Q values are calculated by Sarsa learning. The proposed method are evaluated by applying to a RL problem, Khepera robot [1] control, and compared with some other methods of utilizing infeasible individuals proposed in EDAs community.

The paper is organized as follows: Section 2 presents the overview of PMBGNP; Section 3 introduces the proposed method in details; Section 4 describes the other methods of using the infeasible individuals in PMBGNP for comparison, which are inspired by the other work of this topic in EDAs community. The experimental study is carried out in section 5. Finally, we conclude this paper.

2. PMBGNP

2.1 Overview of PMBGNP

Probabilistic Model Building Genetic Network Programming (PMBGNP) is a novel algorithm that extends EDAs to the graph-based EAs. It uses the directed graph structure of Genetic Network Programming (GNP) [8, 6], which is a kind of graph-based EAs, to represent its chromosome. Some previous research has shown the superiority of graph-based EAs in terms of stronger expression and evolution ability than that of conventional GP in many problems [23, 13, 6]. Different from the other graph-based EAs, GNP is firstly designed for solving some RL problems, such as agent control problems. It consists of judgment and processing nodes to evolve relatively small structures to obtain compact programs. The previous research on GNP has shown its superiority over classical EAs in agent control problems [6, 12, 16]. Meanwhile, it has been applied to various real-world problems, such as data mining [21] and elevator system control [5]. Comparing with conventional EDAs that mostly proposed in the field of GA [9, 18] and GP [19, 20], PMBGNP inherits the advantages of GNP to evolve compact programs.

Another challenge in EDAs is to explore them to many other problems. Recently, a novel algorithm named EDA-RL [4] has been proposed to extend EDAs to solve Reinforcement Learning (RL) problems. Using the behavior sequences of agents (episodes) to form its own chromosome, EDA-RL applies CRFs to represent its probabilistic model

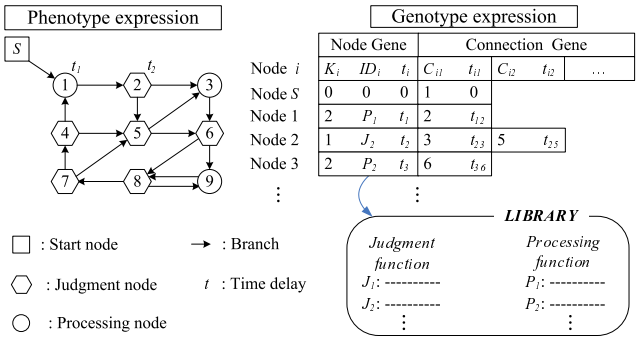


Figure 1: Directed graph structure.

and show some advantages to solve RL problems. On the other hand, the distinguishing directed graph structure of GNP can realize the repetitive processes, where the necessary nodes are executed repeatedly and create compact programs to avoid bloat problem of GP. GNP has been proven to successfully solve some RL problems and obtain better performances than conventional GP and EP [6, 12, 16]. Thus, such distinguishing features of the GNP structure provide the fundamental basis to ensure that it is possible for PMBGNP to handle RL problems well. The previous research of PMBGNP proved that it can also solve some RL problems, i.e., robot control, successfully and obtain better performances than conventional EAs [10].

Therefore, there are mainly two primary features of PMBGNP:

- Graph structure based EDA.
- Ability to handle RL problems.

2.2 Directed graph structure

PMBGNP uses the directed graph structure of GNP to represent its individuals, 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, for node i, 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 function and processing function. 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 required to execute node i and 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 transit and fixed during the evolution process. The judgment and processing nodes save some functions corresponding to the concrete problem and these nodes can be connected arbitrary. For agent control, each judgment node works as "if-then" type decision making functions to judge the environment to make a decision, while processing nodes preserve the action functions to determine the agent's action. By separating judgment and processing functions, GNP can handle various combinations of judgment and processing. That is, GNP can efficiently evolve the compact programs by selecting the necessary judgements and processings to control agents.

Each judgment node consists of multiple branches connecting to different nodes, where the next node to tran-

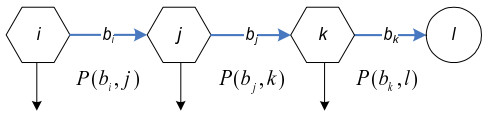


Figure 2: Probabilistic model of PMBGNP.

sit is determined by the judging result of the environment. Processing node has only one branch, since the processing functions only determine the agent's actions. Therefore, the agent is controlled by transiting the nodes until the task is solved. Generally, when solving problems, the number of judgment and processing nodes, the number of branches in judgment nodes and the time delays are predefined. Thus, the size of the GNP structure is fixed during evolution. Although the GNP structure is fixed, the number of used nodes generally varies by making use of the necessary nodes sequentially and repeatedly, which can represent compact programs to control the agents successfully.

2.3 Probabilistic model construction

The model construction of PMBGNP is inspired by most of the classical EDAs, which is to directly study its graph-based chromosome structure. PMBGNP estimates the probabilities of connections between the nodes. Thus, pairwise interactions can be estimated explicitly. The probabilistic model P 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, as shown in Fig. 2. For each branch in the graph structure, the probability to connect to the next node is calculated as follows:

$$P(b_i, j) = \frac{\sum_{n=1}^{N} \Big(\delta_n(b_i, j) + \eta \sigma_n(b_i, j) \Big)}{\sum_{j' \in A(b_i)} \sum_{n=1}^{N} \Big(\delta_n(b_i, j') + \eta \sigma_n(b_i, j') \Big)}, \qquad (1)$$

where,
N: the number of feasible individuals.
$A(b_i)$: set of suffixes of nodes connected from branch b_i of node i.
$\delta_n(b_i, j)$: value defined by

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

η: coefficient.
$\sigma_n(b_i, j)$: value defined by

$\sigma_n(b_i, j) = \ell$ if the transition from branch b_i of node i to node j in individual n occurs ℓ times.

Eq. (1) inspires that both of the information on the node connections and reusability of the node connections are taken into account to construct the probabilistic model of PMBGNP. The previous work on PMBGNP has testified its superiority to solve some problems over conventional algorithms, such as data mining [11] and robot control [10].

3. THE PROPOSED METHOD

As most of the other EDAs, PMBGNP applies truncation selection to bias the population towards feasible individuals, while the infeasible individuals are ignored. In this section, a new method is proposed to take into account infeasible individuals to the probabilistic modeling, where the useful sub-structures of infeasible individuals are extracted.

3.1 Factorization of individuals

Fig. 3 shows an example of GNP runs when applying it to the agent control problems. In each GNP individual, the agent is controlled by following the node transitions of the individual to solve the task, which can be considered as an episode of RL. After obtaining an episode, the transition can be factorized to a sequence of state-action pairs, due to the characteristics of RL. The state and action in GNP structure are defined as follows:

Definition 1. **State**: A state is defined as a branch in GNP structure. Therefore, the set of states $\mathcal{S}$ refers to the set of branches of GNP structure.

Definition 2. **Action**: An action is defined as a node in GNP structure. Therefore, the set of actions $\mathcal{A}$ refers to the set of nodes of GNP structure.

Concretely speaking, an activated branch corresponds to the current state, and the selection of the next node in the current state corresponds to an action. For example in Fig. 3, at time step t_3, when the agent stands in the current state i.e., branch 2 of node 5, it decides an action, i.e., to select node 6 to transit. Therefore, the states and actions can be substituted by the set of branches and set of nodes in the GNP structure, respectively.

Generally, suppose the population size is M. In one generation, after making M individuals execute the task, we can obtain M episodes, which is further factorized to the sequences of state-action pairs. The state-action pairs $(\mathcal{S}, \mathcal{A})_n$ of individual n can be represented as follows:

$$(\mathcal{S}, \mathcal{A})_n = \Big\{ (s_1, a_1)_n, (s_2, a_2)_n, ..., (s_T, a_T)_n \Big\}, \qquad (2)$$

where,
T: the total number of time steps of $(\mathcal{S}, \mathcal{A})_n$.

The factorization of individuals to $(\mathcal{S}, \mathcal{A})$ makes it possible to identify and extract useful sub-structures in infeasible individuals, since the state-action pairs are actually the sub-structures of GNP individual. The task is achieved by calculating the Q values of the state-action pairs, i.e., $Q(\mathcal{S}, \mathcal{A})$, using RL techniques.

3.2 Identification of useful sub-structures

3.2.1 Sarsa learning

In this paper, an on-policy RL technique named Sarsa learning [22] is used to calculate the Q values of state-action pairs of the GNP structure. In Sarsa learning, Q values are updated based on the true actions the agent takes rather than taking the action which gives the maximal reward in Q learning. At time step t, the updating function of Q values depends on the current state s_t of the agent, the current 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}, as shown in the following:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \Big), \qquad (3)$$

where, $Q(s_t, a_t)$ represents the Q values of state-action pair (s_t, a_t). α $(0 < \alpha \leq 1)$ is the learning rate, and γ $(0 \leq \gamma < 1)$ represents the discount factor.

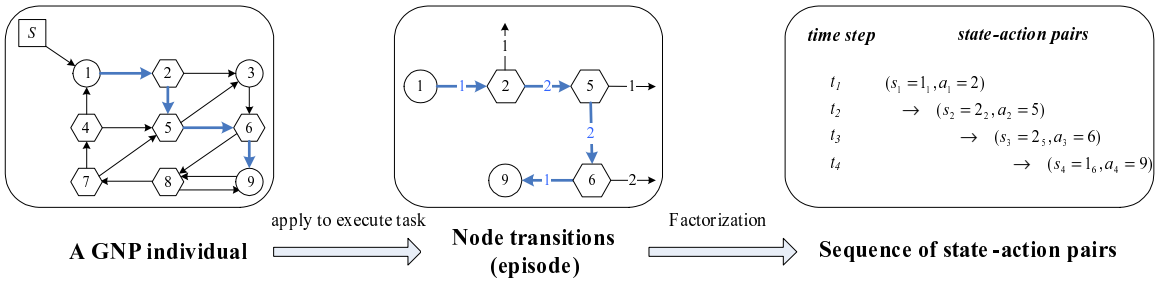


Figure 3: An example of GNP runs. The example shows the procedures of factorizing a GNP individual to the sequence of state-action pairs.

During the interactions between agents and environments, Sarsa learning updates $Q(s_t, a_t)$ by maximizing the expected sum of the future discounted rewards. Sarsa learning has been proven to converge to the optimal solution and work efficiently in large state-spaces [22]. Meanwhile, Sarsa learning has also been proven to be easily applied to the GNP structure, where a variant named GNP-RL [12] was proposed. GNP-RL tends to change the structures of GNP by introducing sub-nodes in each node, and the definitions of the state and action are based on the introduced sub-nodes. However, in this paper, we do not change the GNP structure and use another way to define the state and action. On the other hand, the calculated Q values are particularly used for identifying the useful sub-structures of infeasible individuals of PMBGNP, which is also quite different from GNP-RL.

3.2.2 Calculation of Q values

In the first generation, a Q table which consists of a set of $Q(\mathcal{S}, \mathcal{A})$ for all possible combinations of state-action pairs is generated and the Q values are initialized to 0. During the task executions, Sarsa learning is applied to update the Q values. Suppose the population consists of M individuals, the procedure to update Q values in each generation is as follows:

1: $n = 1$
2: **while** $n \leq M$ **do**
3: execute individual n, obtain the sequence of state-action pairs $(\mathcal{S}, \mathcal{A})_n$
4: update $Q(\mathcal{S}, \mathcal{A})$ based on $(\mathcal{S}, \mathcal{A})_n$ using Sarsa learning
5: $n++$
6: **end while**

In one generation, Sarsa learning is applied to update the Q table M times. With *Definition 1* and *2*, each $Q(s, a)$ implies the quality of transitions between two nodes in the GNP structure. Suppose the state of individual n at time t is branch b_i and its corresponding action is node j, which means the state-action pair can be formed by $(s_t, a_t)_n = (b_i, j)$. Meanwhile, the state-action at time $t+1$ is $(s_{t+1}, a_{t+1})_n = (b_j, k)$. Then, during the execution of individual n, the Q value of (b_i, j) can be updated as follows:

$$Q(b_i, j) \leftarrow Q(b_i, j) + \alpha\Big(r_j + \gamma Q(b_j, k) - Q(b_i, j)\Big), \qquad (4)$$

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

1. If node j is a processing node, then r_j is given after processing node j, which is defined depending on the concrete problem.
2. If node j is a judgment node, then $r_j = 0$.

Eq. (4) inspires that good state-action pairs are given higher Q values since they can obtain higher immediate reward and expected future reward, while bad state-action pairs tend to obtain lower Q values. Therefore, $Q(b_i, j)$ can explicitly judge whether state-action pair (b_i, j) is good or not.

3.2.3 Extraction of useful sub-structures

Given state b_i, the procedure to extract the set of good actions $A(b_i^G)$ is as follows:

Input: the set of Q values: $Q(b_i, \mathcal{A})$
the number of good actions: N^G
Output: the set of good actions: $A(b_i^G)$
1: $count = 0$
$A(b_i^G) = NULL$
2: **while** $count < N^G$ **do**
3: $\widehat{j} = \arg\max_{j \in \mathcal{A}} Q(b_i, j)$
4: save $\widehat{j}$ into $A(b_i^G)$
remove $Q(b_i, j')$ from $Q(b_i, \mathcal{A})$
5: $count++$
6: **end while**

The procedure is repeated for all states $\mathcal{S}$, and finally all good state-action pairs $(\mathcal{S}, \mathcal{A}^G)$ can be extracted. It means that the good actions for each state can be extracted from the action space based on the Q values of state-action pairs, while N^G can be defined to control the number of good actions.

As discussed in the previous sections, infeasible individuals may include some useful sub-structures to control the agent successfully. The extracted good state-action pairs $(\mathcal{S}, \mathcal{A}^G)$ can be directly denoted as useful sub-structures. Therefore, such kind of $(\mathcal{S}, \mathcal{A}^G)$ existed in infeasible individuals can be extracted and viewed as useful sub-structures, which are further incorporated into the probabilistic model.

3.3 Proposed probabilistic model

The population consisting of M individuals are separated to two classes, which consists of N feasible individuals and $M-N$ infeasible individuals. By utilizing useful sub-structures of $M - N$ infeasible individuals to the probabilistic model, Eq. (1) can be rewritten as follows:

$$P(b_i, j) = \begin{cases} \frac{1}{Z(b_i)} \left(\sum_{n=1}^{M} \Big(\delta_n(b_i, j) + \eta \sigma_n(b_i, j) \Big) \right) & \text{if } j \in A(b_i^G), \\ \frac{1}{Z(b_i)} \left(\sum_{n=1}^{N} \Big(\delta_n(b_i, j) + \eta \sigma_n(b_i, j) \Big) \right) & \text{otherwise,} \end{cases} \qquad (5)$$

where $Z(b_i)$ is the normalization function calculated as follows:

$$Z(b_i) = \sum_{j' \in A(b_i)} \sum_{n=1}^{N} \Big(\delta_n(b_i, j') + \eta\sigma_n(b_i, j')\Big) + \sum_{j' \in A(b_i^G)} \sum_{n=N+1}^{M} \Big(\delta_n(b_i, j') + \eta\sigma_n(b_i, j')\Big),$$

It implies that the good sub-structures of infeasible individuals are taken into account for the probabilistic model construction. For example, if action j at state b_i is identified as an good action (which means $j \in A(b_i^G)$), (b_i, j) will be denoted as a good sub-structure and all (b_i, j) in infeasible individuals will be directly taken into account for the probabilistic model construction.

4. THE COMPARED METHODS

The following are the methods to compare with the proposed method:

- GNP: the standard GNP which uses crossover and mutation to evolve the population.
- PMBGNP: the standard PMBGNP which only uses truncation selection to select feasible individuals for model construction.

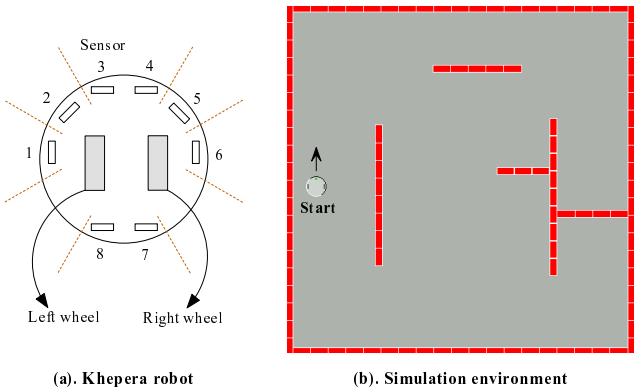
Meanwhile, in order to testify the proposed method, the other methods to utilize infeasible individuals in EDAs community are also designed into the PMBGNP framework for comparison. The following three methods are adopted into PMBGNP.

- Method 1: some of the worst individuals are selected for the probabilistic model construction [2].
- Method 2: all individuals are taken into account for the probabilistic modeling, but each individual is given a weight of its fitness value for $P(b_i, j)$ calculation:

$$P(b_i, j) = \frac{\sum_{n=1}^{M} \Big(\Big(\delta_n(b_i, j) + \eta\sigma_n(b_i, j)\Big) fit(n)\Big)}{\sum_{j' \in A(b_i)} \sum_{n=1}^{M} \Big(\Big(\delta_n(b_i, j') + \eta\sigma_n(b_i, j')\Big) fit(n)\Big)},$$

  where $fit(n)$ is the fitness value of individual n.

- Method 3: the method is designed based on [7]. In this method, two probabilistic models are constructed by standard PMBGNP which are called *feasible* model P^F and *infeasible* model P^I. Feasible model is constructed by estimating the probabilities from the feasible individuals, while infeasible model by infeasible individuals. P^F is used to sample new individuals, which is the same as standard PMBGNP. However, P^I also plays an important role to filter the sample errors. For example, given individual n generated by P^F, we calculate two probabilities that individual n can be sampled from P^F and P^I, respectively, denoted as $P^F(n)$ and $P^I(n)$. These two probabilities are compared. If $P^F(n) \geq P^I(n)$, individual n is generated successfully, otherwise it is denoted as a sample error.



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

Figure 4: Khepera robot and simulation environment.

5. SIMULATIONS

A comparative study is carried out in a RL problem, Khepera robot [1] control. Khepera robot (Fig. 4) is a mobile robot including 8 infrared sensors to detect the proximity of objects around it by reflection. Each sensor returns a value ranging from 0 to 1023, where 0 means that there is no object in front of the sensor and 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.

Khepera robot is controlled to solve the wall following problem. The reward and fitness functions of the wall following problem are designed based on [17] and shown as follows:

$$\text{Reward} = \frac{v_R + v_L}{20}\left(1 - \sqrt{\frac{|v_R - v_L|}{20}}\right) C, \tag{6}$$

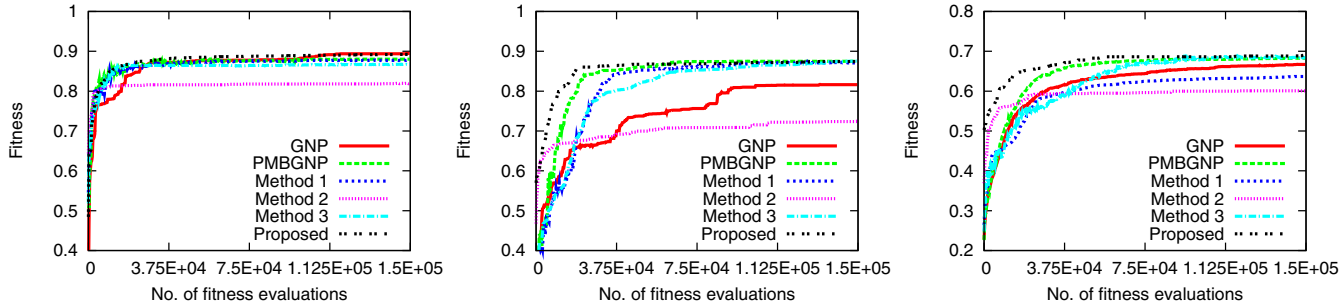
$$\text{Fitness} = \frac{\sum_{step=1}^{S_{max}} \text{Reward}}{S_{max}}, \tag{7}$$

where,
v_R, v_L: the speed of right and left wheels,
S_{max}: the user predefined steps. S_{max} could show 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 4 shows the environment used in this paper. 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. Three simulations are adopted, where $S_{max} \in \{100, 300, 500\}$.

In the context of the wall following problem, given a population consisting of a set of candidate individuals, although some of them are denoted as infeasible individuals due to their low fitness values, we can easily observe that some good state-action pairs still exist in these infeasible individuals for providing high reward values to move the robot successfully. If these good state-action pairs can be extracted correctly, the problem solving process will be sped up. Accordingly,



(a) $S_{max} = 100$ (b) $S_{max} = 300$ (c) $S_{max} = 500$

Figure 5: Fitness curves in three wall following problems.

Table 1: Node functions used for Khepera robot.

Function *ID*	Function
J_1, J_2, ..., J_8	Judge the value of the sensor of 1, 2, ..., 8
$P_1(-10), P_1(-5), P_1(0), P_1(5), P_1(10)$	Determine the speed of the right wheel like -10, -5, 0, 5 or 10
$P_2(-10), P_2(-5), P_2(0), P_2(5), P_2(10)$	Determine the speed of the left wheel like -10, -5, 0, 5 or 10

the objective of this paper and the effectiveness of the proposed method can be verified by the simulations.

5.1 Experimental settings

5.1.1 Settings of directed graph structure

The node functions of GNP are shown in Table 1. There are a total of 8 judgment functions to simulate the corresponding sensors of the robot. The number of branches of each judgment node is set at 2 to efficiently implement 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 robot will take one step of actions if a processing node is transited. Therefore, in each step, GNP judges the sensor values to determine the speed of the wheels to control the movement of the robot and to obtain a reward calculated by Eq. (6). The reward is also used to define the value of the immediate reward r_j in Eq. (4). One individual execution 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 processing function is set at 2, which means each individual consists of $2 \times 10 = 20$ processing nodes. Therefore, the total number of nodes is 60 and that of branches is $40 \times 2 + 20 = 100$ in each individual.

5.1.2 Settings of different methods

The population size of GNP 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, which is the best settings defined by hand-tuning. The population size of PMBGNP is set at 2000. In PMBGNP and its variants, such as Method 1, 2, 3 and the proposed method, the top 50% individuals are denoted as feasible individuals, and η is set at 0.01. PMBGNP directly uses feasible individuals to construct the probabilistic model. On the other hand, Method 1 further selects the worst 20% individuals as infeasible individuals to combine with feasible individuals for model construction. In Method 3, the remaining 50% individuals are denoted as infeasible individuals for infeasible model construction. In the proposed method, the learning rate α and discount factor γ are set at 0.1 and 0.9, respectively, which are determined by experiments, while the number of good actions N^G is set at 15. The terminal condition is defined by the maximal number of fitness evaluations, i.e., 150,000 in this paper.

Table 2: The fitness (std. dev.) results over 20 independent runs.

	$S_{max} = 100$	$S_{max} = 300$	$S_{max} = 500$
GNP	**0.89** (0.012)	0.82 (0.128)	0.66 (0.106)
PMBGNP	**0.89** (0.022)	**0.87** (0.017)	**0.68** (0.087)
Method 1	0.88 (0.010)	**0.87** (0.016)	0.64 (0.068)
Method 2	0.82 (0.077)	0.72 (0.121)	0.60 (0.101)
Method 3	0.87 (0.012)	**0.87** (0.017)	**0.68** (0.074)
Proposed	**0.89** (0.012)	**0.87** (0.020)	**0.68** (0.092)

5.2 Simulation results

5.2.1 Fitness results

Table 2 shows the average fitness and standard deviation over 20 independent runs, and the fitness curves are shown in Fig. 5.

The results show that among three problems, PMBGNP achieves better performances than GNP. On the other hand, the methods that utilize the infeasible individuals for probabilistic modeling are summarized as follows:

(1) Method 1: In simple problems, i.e., $S_{max} = 100$ or 300, Method 1 can achieve similar performance with PMBGNP. However, in the case of $S_{max} = 500$, Method 1 cannot obtain good result. This is because the worst individuals are treated equally with the feasible individuals for the probabilistic modeling, which sometimes provides wrong information and destroy the probabilistic model. [2] also mentioned that Method 1 can only work well in some problems.

(2) Method 2: In Method 2, one important drawback is

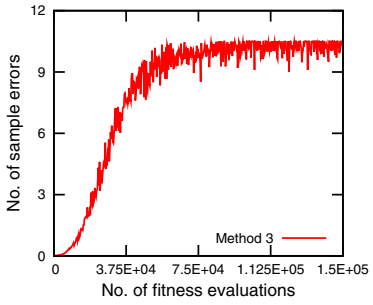


Figure 6: Average number of sample errors generated during the evolution process by Method 3.

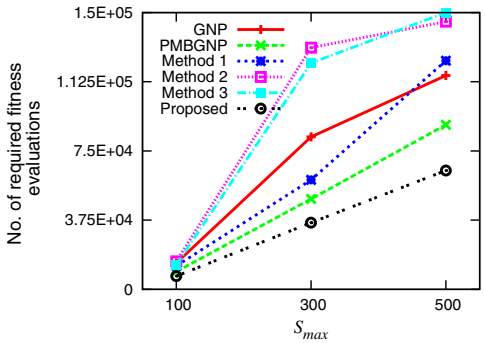


Figure 7: Comparison of the required fitness evaluations.

that it is hard to control the weights of different individuals, which will highly affect the performances. Among three problems, Method 2 used in this paper achieves the worst results. The results show that although Method 2 allows fast convergence speeds, it quickly falls into local optimum.

(3) Method 3: In Method 3, the feasible model P^F is constructed to generate new individuals, while infeasible model P^I is used to filter the sample errors. The fitness results show it achieves the similar performances to PMBGNP. However, one important drawback of Method 3 is that it becomes more and more hard to generate valid individuals during the evolution, since P^I becomes more and more similar to P^F. Fig. 6 shows the average number of sample errors generated during the evolution process by Method 3. It shows that in the later generations, almost 10 sample errors are generated for sampling one valid individual, which takes a long time.

(4) Proposed: Among three problems, the proposed method obtains the similar fitness results to that of PMBGNP. However, we can see that the convergence speed of the proposed method is much faster than PMBGNP especially in the complicated problems like $S_{max} = 300$ and $S_{max} = 500$ as shown in Fig. 5. On the other hand, comparing with the other methods that utilize the infeasible individuals, the proposed method can obtain the best fitness results.

5.2.2 *Required fitness evaluations*

We further compared the average number of required fitness evaluations of different methods to control the robot moving around the wall successfully. The average number of required fitness evaluations is calculated as follows: For each successful simulation run, the exact number of fitness evaluations is counted, while the maximum number of fitness evaluations, i.e., 150,000, is used for each failed run. Then, these numbers of 20 independent runs are averaged. Particularly, the number of sample errors are also counted in Method 3, since they also cost much time.

The average numbers of required fitness evaluations for the three wall following problems are shown in Fig. 7, and the results of the t-test are shown in Table 3. The simulation results indicate that the proposed method shows the fastest convergence to solve the task successfully for the three wall following problems with different complexities among the methods. In addition, the results of the t-test (one-side) show that there are statistically significant differences between the proposed method and the other ones.

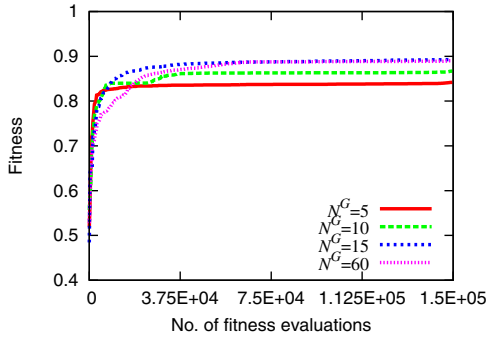


Figure 8: Effect of parameter N^G in the problem of $S_{max} = 100$.

5.2.3 *Effect of parameter N^G*

In the proposed method, one important parameter N^G should be set appropriately to control the number of good actions, as discussed in section 3.

Fig. 8 shows the fitness curves of the proposed method with different settings of N^G in the problem of $S_{max} = 100$. When the value of N^G is small, the actions with higher Q values are highlighted in the probabilistic model. In that case, the proposed method works as a greedy policy which has high possibility to cause the premature convergence. On the other hand, if the value of N^G is set at a large value, many actions with low Q values are also counted in the probabilistic modeling, which will decrease the convergence speed. In the problem of $S_{max} = 100$, the appropriate setting of N^G is 15.

6. CONCLUSIONS

In this paper, a novel method has been proposed in PMBGNP to utilize infeasible individuals. The conventional methods of utilizing infeasible individuals in EDAs community showed some disadvantages, such as premature convergence and difficulty of parameter control, while the proposed method provides another approach to utilize infeasible individuals. That is, the proposed method applies Sarsa learning to identify and extract useful sub-structures of infeasible individuals, while the sub-structures are used in the probabilistic modeling of PMBGNP. The effectiveness of the proposed method is testified in a RL problem, i.e., robot control. The simulation results show that the proposed method can achieve better performances than standard PMBGNP and the other methods of utilizing infeasible individuals.

Table 3: The average number of required fitness evaluations (std. dev.) and t-test results.

		GNP	PMBGNP	Method 1	Method 2	Method 3	Proposed
$S_{max} = 100$	mean	14250	9600	12700	15200	13208	**7200**
	(std. dev.)	(9423)	(4285)	(8688)	(14014)	(10449)	(4175)
	t-test (p value)	1.70×10^{-3}	4.83×10^{-2}	1.20×10^{-2}	1.18×10^{-2}	8.14×10^{-3}	—
$S_{max} = 300$	mean	82650	49000	59350	131000	122625	**36150**
	(std. dev.)	(58695)	(25236)	(34654)	(44710)	(37599)	(19343)
	t-test (p value)	2.85×10^{-3}	5.27×10^{-2}	7.31×10^{-3}	5.56×10^{-8}	6.22×10^{-9}	—
$S_{max} = 500$	mean	115890	89200	123900	145100	150000	**64400**
	(std. dev.)	(42673)	(48815)	(32272)	(16293)	(0)	(38741)
	t-test (p value)	2.43×10^{-4}	4.47×10^{-2}	6.05×10^{-5}	1.79×10^{-8}	3.19×10^{-9}	—

7. REFERENCES

[1] Webots software. http://www.cyberbotics.com/.

[2] S. Brownlee, J. McCall, Q. Zhang, and D. Brown. Approaches to selection and their effect on fitness modelling in an estimation of distribution algorithm. *In Proc. of the IEEE Congress on Evol. Comput.*, pages 2621–2628, 2008.

[3] B. Chen, L. Li, and J. Hu. A novel EDAs based method for HP model protein folding. *In Proc. of the IEEE Congress on Evol. Comput.*, pages 309–315, 2009.

[4] H. Handa. EDA-RL: Estimation of distribution algorithms for reinforcement learning problems. *In Proc. of the Genetic and Evol. Comput. Conf.*, pages 405–412, 2009.

[5] 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*, 38(4):535–550, 2008.

[6] 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 Evolut. Comput.*, pages 1276–1282, 2001.

[7] Y. Hong, G. Zhu, S. Kwong, and Q. Ren. Estimation of distribution algorithms making use of both high quality and low quality individuals. *In Proc. of the IEEE Int'l Conf' on Fuzzy Systems*, pages 1806–1813, 2009.

[8] 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*, pages 3829–3834, 2000.

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

[10] X. Li, S. Mabu, and K. Hirasawa. Towards the maintenance of population diversity: A hybrid probabilistic model building genetic network programming. *Trans. of the Japanese Society for Evol. Comput.*, 1(1):89–101, 2010.

[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. of the IEEE Congress on Evol. Comput.*, pages 2673–2680, 2010.

[12] S. Mabu, K. Hirasawa, and J. Hu. A graph-based evolutionary algorithm: Genetic network programming (GNP) and its extension using reinforcement learning. *Evol. Comput.*, 15(3):369–398, 2007.

[13] J. F. Miller and P. Thomson. Cartesian genetic programming. *In Proc. of the 3rd European Conf. on Genetic Programming*, pages 121–132, 2000.

[14] T. Miquélez, E. Bengoetzea, and P. Larrañaga. Evolutionary computation based on bayesian classifiers. *Int'l J. of Appl. Math. Comput. Sci.*, 14(3):335–349, 2004.

[15] M. Munetomo, N. Murao, and K. Akama. Introducing assignment functions to bayesian optimization algorithms. *Information Sciences*, 178(1):152–163, 2008.

[16] 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.*, pages 712–714, 2004.

[17] P. Nordin, W. Banzhaf, and M. Brameier. Evolution of a world model for a miniature robot using genetic programming. *Robotics and Autonomous Systems*, 25:105–116, 1998.

[18] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz. Linkage problem, distribution estimation, and bayesian networks. *Evol. Comput.*, 8(3):311–341, 2002.

[19] R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Evol. Comput.*, 5(2):123–141, 1997.

[20] Y. Shan, R. I. McKay, D. Essam, and H. A. Abbass. A survey of probabilistic model building genetic programming. *Studies in Computational Intelligence*, 33:121–160, 2006.

[21] K. Shimada, K. Hirasawa, and J. Hu. Genetic network programming with acquisition mechanisms of association rules. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 10(1):102–111, 2006.

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

[23] 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.

[24] K. Yanai and H. Iba. Estimation of distribution programming based on bayesian network. *In Proc. of the IEEE Congress on Evol. Comput.*, pages 1618–1625, 2003.

[25] Y. Yu and Z. H. Zhou. On the usefulness of infeasible solutions in evolutionary search: A theoretical study. *In Proc. of the IEEE Congress on Evol. Comput.*, pages 835–840, 2008.