Waseda University Doctoral Dissertation

Study on Probabilistic Model Building Genetic Network Programming

Xianneng LI

Graduate School of Information, Production and Systems
Waseda University

January, 2013

Acknowledgments

I would like to express my deepest gratitude and thanks to my supervisor, Professor Kotaro Hirasawa, for his invaluable advices and support that guided me to successfully complete this dissertation. Professor Hirasawa gives extraordinary patients to guide me into the world of scientific research. I would express my highest respect of thanks to Professor Hirasawa for helping me ride out the storm of my life. It would not have been possible for me to study in Waseda University and complete this dissertation without the help and support of Professor Hirasawa.

I would like to thank my dissertation committees, Professor Kotaro Hirasawa, Professor Jinglu Hu, Professor Osamu Yoshie and Professor Shigeru Fujimura for their insightful comments and suggestions which improve the quality of this dissertation.

I would also like to thank Dr. Shingo Mabu and Dr. Kaoru Shimada, who provided many valuable comments and discussions to complete the research topics of this dissertation. The colleagues of Hirasawa Lab. and friends of Waseda University are also greatly acknowledged. Thanks for the precious and colorful time we spent together.

This research was partially supported by the Japan Society for the Promotion of Science (JSPS) Grants-in-Aid, No. 243550 and the Waseda University Grant for Special Research Projects, No. 2011A-913, funded to me. Special thanks also go to the JSPS, Waseda University Global COE program, Mitsubishi UFJ Trust Scholarship Foundation and the Okawa Foundation for Information and Telecommunications, which provide the partial scholarships during my student life in Waseda University. They are gratefully acknowledged for the financial supports which make this research possible.

Finally, I would like to dedicate my sincere gratitude to my family. Thank my parents for many years of support and guidance during my life. Thank my wonderful wife for all the support and love that encourage me to make utmost effort to work on the research. Thank them to give me such a wonderful and magnificent life.

Abstract

Estimation of Distribution Algorithm (EDA) is one of the most important branches in Evolutionary Computation (EC). Different from the conventional Evolutionary Algorithms (EAs) which use stochastic ways to simulate the biological genetic operators, i.e., crossover and mutation, for new population generation, EDA constructs a probabilistic model using the techniques of statistics and machine learning to estimate the probability distribution of the current population, and samples the model to generate the new population. By explicitly estimating and recombining the good partial solutions of the population, EDA has been successfully proven to outperform conventional EAs by avoding the premature convergence and speeding up the evolution process in many problems.

The primary objective of this thesis is to propose a novel paradigm of EDA named Probabilistic Model Building Genetic Network Programming (PMBGNP), where the directed graph structure of a novel graph-based EA called Genetic Network Programming (GNP) is used to represent its individuals. Different from most of the current EDAs proposed in string structure based Genetic Algorithm (GA) and tree structure based Genetic Programming (GP), the distinguished graph structure allows PMBGNP to ensure higher expression ability. As a result, a sort of problems can be explored and solved efficiently and effectively comparing with the conventional research in EDA literature.

To achieve this objective, contributions of this thesis are presented on the following two aspects: algorithm part and application part.

From the perspective of algorithm part, first, the thesis proposes the high-level PMBGNP to use Maximum Likelihood Estimation (MLE) to model the probability distribution of the promising individuals. PMBGNP is empirically studied to show the capability of speeding up the evolution efficiency by the estimation of probability distribution. Second, the thesis addresses the issue of population diversity loss by theoretical comparison with classical EDAs, and proposes a hybrid algorithm to maintain the population diversity of PMBGNP. Third, the integration of PMBGNP and Reinforcement Learning (RL) is studied. Inspired by behaviorist psychology, RL concerns with reinforcing the growth of the individuals by learning their experiences. The learning knowledge formulated by Q values can be approximated and incorporated into the probabilistic modeling of PMBGNP to improve the performance by constructing a more accurate model. Finally, PMBGNP is extended from discrete optimization problems to continuous optimization problems.

From the viewpoint of application part, most of the current studies in EDA are carried out in the benchmark problems of GA and GP, such as function optimization and symbolic regression. Therefore, to accomplish one of the essential challenges of EDA for novel applications, the thesis applies PMBGNP to two novel applications of EDA, including data mining and the problems of controlling the agents' behavior. By comparing with the other state-of-the-art algorithms, PMBGNP is testified to be capable of achieving better performances.

Contents

Chapter 1

Introduction

Contents

1.1 Background

In real-worlds, various problems can be classified into the group of optimization problems. Researchers in Computer Science or related fields have focused on solving the optimization problems by proposing numerous methods. Along with the methods derived from the classical techniques of operational research or mathematics which requires much prior knowledge to build the mathematical models of problems, Evolutionary Computation (EC) provides an alternative direction to solve the optimization problems. The class of algorithms from EC is generally called Evolutionary Algorithms (EA or EAs).

Inspired by Darwin's theory of natural evolution, EAs use iterative progress to evolve a population of candidate solutions from the entire search space. The population is evolved by the analog of natural selection and random variation derived from biological evolution. Based on this concept, the population of EAs is evolved towards more promising region of the search space which implies that the qualities of solutions become better and better.

In general, the candidate solutions of the optimization problems are formulated by the individuals of EAs. In each iteration (generally called generation), a population of candidate individuals is subject to evolution by means of three main biological evolution mechanisms, including selection, crossover and mutation. Selection imposes the concept of "Survival-of-the-fittest" on the population to select fitter individuals. Crossover imitates the concept of "Marriage" to generate better offsprings by the gene swap of selected parents. Mutation changes a single individual by perturbing parts of its genes. Selection mainly acts as a force to improve the quality of the current population to the next population by duplicating the fitter individuals. Crossover and mutation simulate the genetic recombination to generate novel sets of individuals with higher quality by exploring the search space and inserting the necessary diversity. The evolution process is executed iteratively until the optimal or near optimal solutions are found.

Comparing with the classical methods of operational research and mathematics which require much knowledge on the modeling of the problems, and the classical deterministic methods in which the same solution is always obtained under the same conditions causing the local convergence, EAs show the advantages of simplicity of implementation and flexibility of different problems. Most importantly, EAs are essentially suitable for solving the problems that the optimal solution is hardly found in acceptable execution time. This means that although they do not ensure finding the global optimum (neither the classical optimization methods), EAs can obtain quite attractive and acceptable solutions near to the optimum in acceptable execution without mathematical modeling of the problems. As a result, EC has been attracted much attention by researchers to propose numerous EAs in the past several years.

1.1.1 Conventional Evolutionary Algorithms (EAs)

There have been several well-known EAs proposed in the past several years, such as Genetic Algorithm (GA) [Holand 1975, Goldberg 1989a], Evolution Strategy (ES) [Beyer 2002], Evolutionary Programming (EP) [Fogel 1994] and Genetic Programming (GP) [Koza 1992, Koza 1994]. The main difference among these algorithms relies on the representation of individual structures. GA encodes its individual by a sequence of bit-strings, ES individuals are coded as vectors of real numbers, EP represents its individuals by Finite State Machines (FSMs) and GP uses tree structures to represent its individuals. Despite such a main difference, these EAs share the same feature that they are population-based metaheuristic optimization algorithms inspired by Darwin's evolution theory.

A) Basic flowchart of EAs

All EAs are carried out in principle based on the flowchart illustrated in Fig. 1.1, including the following 5 steps.

1. *Initialize population*: A population of candidate solutions (individuals) is generated randomly.

2. *Fitness evaluation*: A problem specific fitness function (user-defined objective function) is imposed to calculate the fitness value of each individual.

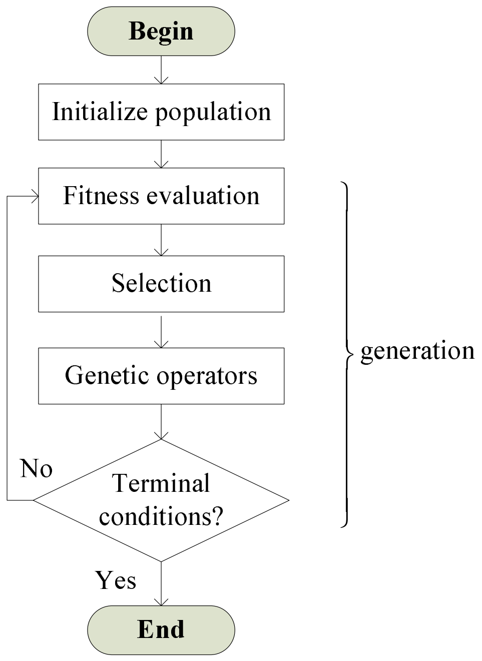


Figure 1.1: Flowchart of EAs.

3. *Selection*: Based on the results of fitness evaluation, selection strategies are applied to select fitter individuals for the generation of the new population.

4. *Genetic operators*: After the selection process, genetic operators, such as crossover and/or mutation, are imitated to generate offspring by exploring the search space of the selected individuals.

5. *Terminal conditions*: If the terminal conditions are satisfied, the evolution will end. Otherwise, evolution continues to step 2. The terminal conditions are generally user-defined, where the typical ones include whether the optimum (or acceptable near optimum) is found, or whether the maximal number of generations is reached.

An execution from step 2 to 5 is generally called one *generation*, where EAs gradually find better individuals through natural evolution generation by generation.

B) Applications of EAs

In principle, EAs are designed to solve the optimization problems. The application domains of different classical EAs are generally different due to the characteristics of their individual representations.

In the brief expression, GA is mainly applied to solve the function optimization problems. Similarly, ES is also used to solve function optimization problems, but with continuous domain of the search space. Both of them use string structures to represent the search/problem space. Despite many benchmark functions introduced and studied in GA and ES, numerous real-world applications could be expanded in these fields. To name a few, these include scheduling problems, parameter optimization, data mining, financial engineering, bioinformatics, evolvable hardware, etc.

GP is another one of the most important EAs. As the name implies, the fundamental basis of GP making it unique from GA is that GP is primarily used to evolve a population of computer programs which are formulated by tree structures. In other words, it is mainly designed as an automatic programming tool for problem solving. More importantly, such tree structures provide more complex ways to represent solutions, which explores EAs to solve a large class of problems different from the ones of GA. The tree structure is actually very close to the natural structures of programs and algorithms, and it can represent various computer elements, such as decision trees, rules and mathematical equations. As a result, GP has been successfully applied to solve many problems, including symbolic regression, electrical circuit design, data mining, classification, financial engineering, image processing, game playing, intelligent agent control and robotics, etc. Moreover, GP has been testified to automatically produce results which equal to or even better than those produced by humans through numerous human-competitive competitions.

EP is another important variant of EAs, however, which has been merged into the other EAs by the recent research due to the similarities in both algorithms and applications. In the previous research, EP has been successfully applied to solve the problems of ES and GP using its own encoding manners.

C) Problems of EAs

As discussed before, the analogs of natural evolution in classical EAs are the ones called genetic operators, including crossover and mutation. Crossover and mutation play the fundamental roles to guide the evolutionary search to optimal solutions. Although the evolution is carried out with bias which means that crossover and mutation are mainly applied to explore the search space towards the promising individuals, these genetic operators actually are the variants of the stochastic search. In other words, although classical EAs reproduce and combine the high-quality partial solutions to form new solutions, the process is generally achieved implicitly through problem-independent, stochastic and fixed crossover and mutation.

The previous research has stated that *Building Block Hypothesis* (BBH) [Holand 1975] provides the fundamental basis that makes the stochastic heuristic search of GA succeed in solving optimization problems.

Definition 1 (Building Block Hypothesis, BBH) *GA is capable of performing adaptation to seek near optimal performance by identifying and recombining the* Building Blocks *(BBs).*

Definition 2 (Building Blocks, BBs) *BBs are the short, low-order, low defining-length partial solutions (also called* schemata*) with fitness higher than the average fitness. In short, BBs are the high-quality partial solutions.*

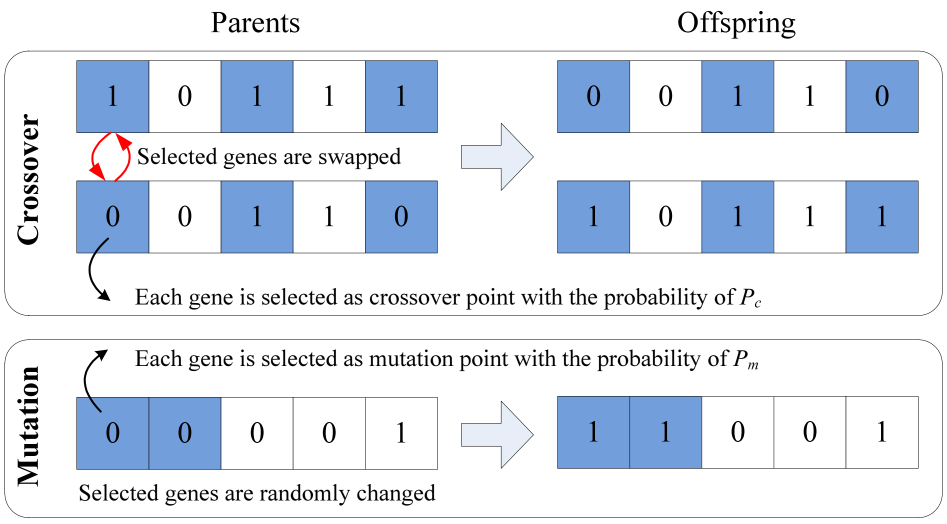

Parents
Offspring
Crossover
1 0 1 1 1
Selected genes are swapped
0 0 1 1 0
0 0 1 1 0
1 0 1 1 1
Each gene is selected as crossover point with the probability of P_c
Mutation
Each gene is selected as mutation point with the probability of P_m
0 0 0 0 1
1 1 0 0 1
Selected genes are randomly changed

Figure 1.2: Example of crossover and mutation in GA.

In other words, GA is capable of identifying the BBs and effectively combining them to produce better solutions through genetic operators, such as crossover. Although BBH is a hypothesis, various studies have addressed that it provides the direction of understanding the success of GA when applied to practical problems in certain degrees. Such theoretical foundation is later explored to the whole EC field for the descriptions of the other EAs.

However, the main control mechanisms to enforce the evolution performance are the two parameters called crossover rate P_c and mutation rate P_m. As an instance shown in Fig. 1.2, in the commonly used genetic operators of GA called uniform crossover and mutation, each gene/allele of GA individuals is selected as the crossover and mutation points with the predefined probabilities P_c and P_m, respectively. This means that after selecting the parents from the current population (this process enforces the search with bias towards the fitter individuals), the evolution is basically carried out by randomly selecting some genes of parents to swap or change. It is generally hard to find the appropriate parameters for all problems, which means that the parameters are problem-dependent. Too small P_c and P_m may cause the low evolution efficiency and genetic drift, while too large P_c and P_m may lead to the premature convergence and the loss of good solutions, respectively.

Such conventional methods have both advantages and disadvantages. First, they show the advantage of simplicity of implementation. Meantime, because the stochastic genetic operators will always reach different final solutions under different trials, the local convergence of classical deterministic methods will be reduced in certain degrees. However, one can easily observe that such kinds of stochastic ways have risks for breaking down the BBs which cause the low evolution efficiency or even make the problems unsolvable.

1.1.2 Estimation of Distribution Algorithm (EDA)

To improve the performance by overcoming the above problems, researchers have developed many improvements by following different directions. Among them, one of the most famous topics is called *Linkage-Learning* (LL) [Goldberg 1989b, Harik 1997]. In LL, the algorithms aim to learn the gene structures by grouping the subset of genes which are dependent with each other. By explicitly grouping and identifying these dependent genes, the BBs can be efficiently preserved and recombined in certain degrees.

Another approach aiming to incorporating the BBs explicitly is called *Estimation of Distribution Algorithm* (EDA) [Larrañaga 2002]. EDA is a variant of EAs that guide the search towards the optimal solutions by building and sampling explicit probabilistic models of promising candidate solutions. In other words, EDA stores the linkage information into the probabilistic models to form the BBs, and as a result, the BBs can be explicitly recombined through sampling the probabilistic models. The fundamental basis of EDA is that it estimates the probabilistic models from the promising solutions, and the models are used to replace the conventional genetic operators, such as crossover and mutation, for the generation of the new solutions. As a result, the control parameters such as P_c and P_m are neglected which lead to the simplicity of the parameter tuning. More importantly, the problem of designing EDA relies on the methods of probabilistic modeling, where estimating the probability distribution from the promising solutions is carried out by the techniques of statistic and machine learning. From the perspective of machine learning, the selected promising solutions can be viewed as a set of samples to be learnt. As a result, it is easy to apply the machine learning techniques to the probabilistic modeling of EDA. To some extent, EDA actually builds the bridge between EC and machine learning.

A) Basic flowchart of EDA

The detailed flowchart of EDA is shown in Fig. 1.3.

Despite the other steps, the generation of the new population in EDA is different from that of conventional EAs. EDA estimates the distribution of selected individuals for the probabilistic model construction, and the model is sampled to generate the new population. Consequently, the conventional genetic operators, such as crossover and mutation, are replaced.

Definition 3 (Truncation selection) *In truncation selection, the individuals of the population are ordered by fitness, while the top N (user-defined integer) individuals are selected.*

In most EDAs, the selection strategy is *truncation selection* [Mühlenbein 1996]. As a result, the probabilistic model represents the probability distribution (probability density in the case of continuous domains) of the promising individuals.

B) Brief introduction of EDA: a case study

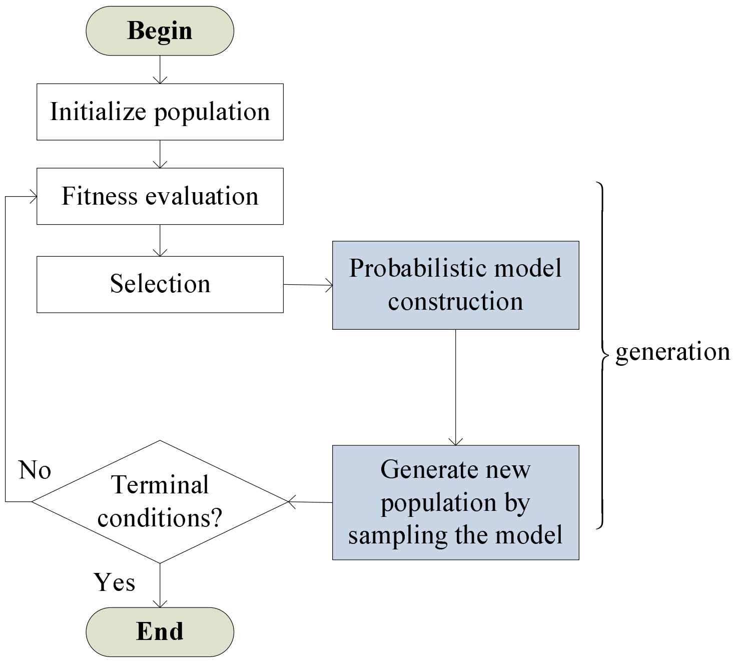


Figure 1.3: Flowchart of EDA.

To make an intuitive explanation of EDA, one of the most earliest EDAs: Univariate Marginal Distribution Algorithm (UMDA) [Mühlenbein 1996] is taken as an example.

In UMDA, the individuals are represented by GA's bit-string structure. In the chromosome of individuals, different genes are assumed to have no relationship. As a result, each gene consists of two probability vectors, $P_i(0)$ and $P_i(1)$, corresponding to the probabilities that the value of gene i is 0 or 1, respectively, where condition $P_i(0) + P_i(1) = 1$ is maintained.

In the example of Fig. 1.4, the problem to be solved is a benchmark function named *OneMax* problem, where the objective is to find the optimal solution which can maximize the following fitness function:

$$f(X) = \sum_{i=1}^{n} x_i, \tag{1.1}$$

where,
n: the number of genes in GA.
X: chromosome of GA;
$x_i \in \{0, 1\}$: value of gene i of chromosome X.

As a result, X can be represented by the formula $X = \{x_i | i = 1, 2, ..., n\}$.

The number of genes is set at 5, and the population size is 4. As a result, the optimal chromosome is "11111" with the fitness value of $f = 5$. The initial population $Pop(0)$ is generated randomly (which is equivalent to $P(0) = P(1) =$

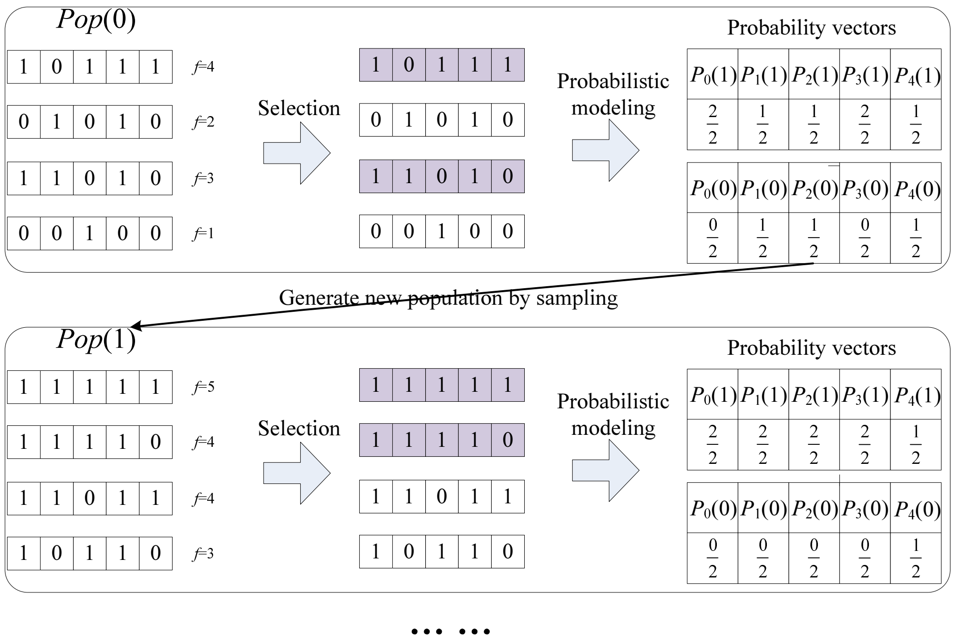


Figure 1.4: Example of UMDA. (Problem: OneMax; Fitness function: $f(X) = \sum_{i=0}^{n-1} X_i$; Selection strategy: truncation)

0.5 for all genes). After the fitness evaluation, truncation selection is applied to select the best 2 individuals. Then, Maximum Likelihood Estimation (MLE) is applied to count the frequencies of 0 and 1 over all genes of the best individuals. These frequencies constitute the probabilistic model of UMDA. The mathematical expression of UMDA can be written as follows:

$$P_i(1) = \frac{\sum\limits_{x=1}^{N} x_i}{N}, \tag{1.2}$$

where,
N: the number of best individuals (truncation selection size).

Accordingly, $P_i(0)$ can be calculated by $1 - P_i(1)$. Since the probability vectors are calculated by only considering the frequencies of genes from the best individuals, the probabilistic model can be thought as a representative which can express the features of the best individuals to bias the search of evolution. The next population $Pop(1)$ is generated by sampling the probabilistic model. This process is repeatedly executed until the terminal conditions. The evolution of UMDA is clearly shown in Fig. 1.4 where better population can be generated by learning and sampling the probabilistic model.

UMDA is one of the simplest EDAs, however, which has been proven to provide competitive, or even better results than GA in various problems. Similar versions of UMDA include Population-based Incremental Learning (PBIL) [Baluja 1994]

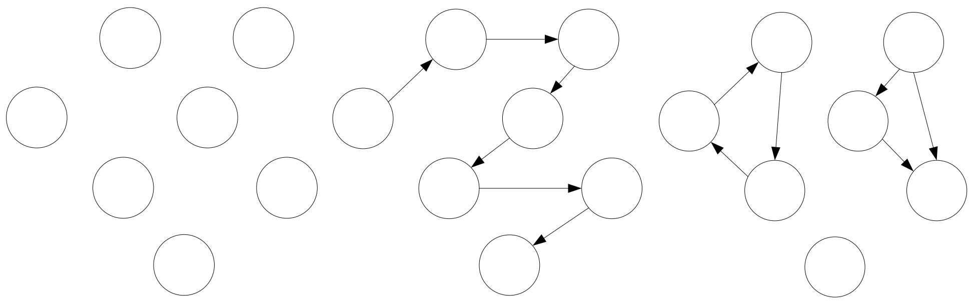


Figure 1.5: The examples of probabilistic models in EDA classified by different dependencies of variables (Univariate model: UMDA; Pairwise model: MIMIC; Multivariate model: BOA). The nodes in the graphical models represent the variables, where the connections denote the interactions between the variables.

and Compact GA (CGA) [Harik 1999].

C) Classification of EDA

Since the proposal of EDA, it has received much attention in the last decade, where many algorithms have been proposed to draw its success. There are many ways to classify the existing EDAs, such as dependency of variables (model complexity), individual representation, problem domains and application aspects.

From the perspective of dependency of variables (model complexity), EDA can mainly be classified into three classes: univariate model, pairwise model and multivariate model.

1. *Univariate model*: The idea of EDA was first introduced in binary GA. In the earliest algorithms so called univariate model-based EDAs, different genes are assumed to have no relationship and each genes consists of two probability vectors, $P(0)$ and $P(1)$, corresponding to the probabilities that the value of this gene is 0 or 1. The representatives of univariate model-based EDAs include PBIL, UMDA and CGA, etc. Later, some researchers extended it from GA's bit-string structure to GP's tree structure to propose a GP version called Probabilistic Incremental Program Evolution (PIPE) [Salustowicz 1997].

   The univariate model-based EDAs have been proven to work fairly well for linear problems and some sorts of real-world applications. However, for many problems where the variables are strongly related (interact with each other), the univariate model generally fails due to the fact violating its assumption of no interactions between variables.

2. *Pairwise model*: Later studies extend EDA to pairwise model. In pairwise model, the genes are assumed to have pairwise interactions, which means that the probabilistic model can represent not only the BBs of order one (single gene) but also the BBs with order two (two genes). In other words, the

pairwise model consists of marginal probabilities and conditional probabilities to cover the genes without and with pairwise interactions. Pairwise model is more complex than univariate model, but it can model more complex BBs.

The classical pairwise model-based EDAs include Mutual Information Maximization for Input Clustering (MIMIC) [Bonet 1997], Combining Optimizers with Mutual Information Trees (COMIT) [Baluja 1997] and Estimation of Distribution Programming (EDP) [Yanai 2003], etc. The current studies investigate that the pairwise model can outperform the univariate model in quadratic problems, however, estimating the distribution also costs more time due to its complexity.

3. *Multivariate model*: Naturally, recent studies care more on extending EDA to cover multivariate interactions. In that sense, the BBs with any order are to be identified, however, which is not an easy task. The key point of the multivariate model is to identify and group multiple variables which are related with each other. To achieve this task, researchers have proposed various algorithms by integrating machine learning techniques.

   In Extended CGA (ECGA) [Harik 2006], the probabilistic model is represented by multiple marginal distributions corresponding to the BBs. The variables are first considered independent, and each variable corresponds to a marginal probability. During evolution, the variables are merged as much as possible by using a measure called Minimum Description Length (MDL). As a result, the variables are grouped into various disjoint sets, where the variables of the same set are considered to be interacted with each other. Since there is no restriction for the size of the disjoint sets, ECGA is capable of representing the BBs with any order. However, since each variable can be only grouped into one set, ECGA cannot solve the problems where the variables of different BBs are overlapped (overlapping BBs). Factorized Distribution Algorithm (FDA) [Mühlenbein 1999b] predefines a probabilistic model based on the factorized distribution given by the problem, where it learns the parameters of the model during evolution to model the multivariate interactions. However, the required factorized distribution is generally unknown in real-world problems. As a result, FDA is mainly applied to solve some theoretical problems, such as additively deceptive functions. A wide study on EDA is to apply Bayesian network to capture the multivariate interactions among variables. The representatives include Bayesian Optimization Algorithm (BOA) [Pelikan 2002a], Estimation of Bayesian Network Algorithm (EBNA) [Etxeberria 1999] and Learning F-DA (LFDA) [Mühlenbein 1999a]. In these EDAs, Bayesian network is used to represent the probabilistic model, in which the connections between the variables denoted by the nodes of Bayesian network actually represent the corresponding multivariate interactions. Bayesian network is generally constructed through learning the promising individuals obtained by truncation selection, and used to sample the new population. However, the construction of the Bayesian network itself in every generation is actually an optimization problem. As a result, even the greedy algorithms are generally employed to construct the Bayesian network, this process is still quite time consuming.

Obviously, modeling the BBs with higher order can represent the real probability distribution of problems more accurately than the simpler models, however, which requires to pay the price for the exponential increase of computation time.

From the perspective of individual representation, the existing EDAs can mainly be classified into two classes: *Probabilistic Model Building Genetic Algorithm* (PMBGA) [Pelikan 2002b] and *Probabilistic Model Building Genetic Programming* (PMBGP) [Shan 2006].

The class of EDAs using GA's bit-string structure for individual representation is generally called PMBGA. Most of the existing EDAs introduced above belong to this class. After the proposal of EDA using GA's bit-string structure, some work extended EDA to tree structure GP to propose a new research topic called PMBGP. As the name implies, PMBGP mainly focuses on learning the tree structure of GP to model the probability distribution and use it for sampling the new population. Extending EDA from the string structure to tree structure provides a more complex way to represent solutions, which explores it to solve a large class of problems, such as program evolution. Divided by model complexity, the existing PMBGPs include univariate model-based PIPE, pairwise model-based EDP, multivariate model-based Extended Compact GP (ECGP) [Sastry 2003] and Program Optimization with Linkage Estimation (POLE) [Hasegawa 2008], which identify and recombine the BBs with different orders in the tree structure of GP.

The class of the above PMBGPs is also called *prototype tree-based PMBGP*, which directly extends EDA to standard GP for learning its tree structure. There is another class of PMBGPs that applies EDA to Grammar guided GP (GGGP) [Whigham 1995, McKay 2010] named *grammar model-based PMBGP* [Shan 2004, Shan 2006]. Another work called N-gram GP [Poli 2008b] explores PMBGP to linear GP.

Considering the classification of EDA by problem domains, the EDAs mentioned above are generally used for the problems of discrete domains, which can be broadly called *discrete EDA*. Another wide range of EDAs extends the existing discrete EDAs to solve problems of continuous domains, which we called *continuous EDA* [Larrañaga 1999, Bosman 2006].

From the application aspects, most of the classical EDAs are studied to clarify their performances in the benchmark problems of GA and GP, i.e., function optimization problems by PMBGAs, symbolic regression and Royal trees problems by PMBGPs. Apart from these, various EDAs and their extensions have also been successfully applied to a sort of applications, such as dynamic problems [Yang 2008], bioinformatics [Santana 2008a], multiobjective optimization problems [Zhang 2008] and reinforcement learning problems [Handa 2009], etc.

1.2 Research objective

The primary objective of this thesis is to propose a novel paradigm of EDA named Probabilistic Model Building Genetic Network Programming (PMBGNP), where the directed graph structure of a novel graph-based EA called Genetic Network Programming (GNP) [Hirasawa 2001, Mabu 2007b] is used to represent its individuals. Different from most of the current EDAs belonging to PMBGA and PMBGP, the distinguished graph structure allows PMBGNP to ensure higher expression ability, where a large number of problems can be explored and solved efficiently and effectively comparing with the conventional research in EDA literature.

In other words, the study of this thesis is mainly to fulfill the following two challenges of EDA:

1. In most of the current bit-string, tree structures-based EDAs, the abilities of representing solutions and the evolution are not enough in terms of the system modeling. Meantime, there is little work on extending EDA to the graph structure based EAs, which have higher expression abilities than that of GA and GP.

2. Due to the restrictions of bit-string and tree structures, most of the current EDAs are carried out in the benchmark problems of GA and GP, where one of the essential challenges in EDA is to explore it to various other problems.

This thesis proposes the graph-based EDA: PMBGNP, which extends the bit-string structure based PMBGA and tree structure based PMBGP to a directed graph structure. Due to the characteristics of its graph-based individual representation, PMBGNP is applied to solve the novel problems of EDA, including data mining and the problems of controlling the agents' behavior. Moreover, another advantage of PMBGNP by combining EDA and GNP is that, it is capable of not just estimating the probabilities of the connections of nodes, but also the contents of the nodes to evolve the optimal directed graph structures, while the conventional EDAs only estimate the probabilities of the contents of genes/nodes.

During this study, various extensions of PMBGNP are proposed to improve its performance based on different sights.

1.3 Organization of the thesis

Besides the first chapter introducing the background of this thesis, the rest of the thesis is divided into six chapters.

1.3.1 Chapter 2: Probabilistic Model Building Genetic Network Programming (PMBGNP)

Chapter 2 describes the algorithm of PMBGNP in details, where the individual representation, probabilistic modeling and methods of sampling the new population are introduced.

To verify its performance, PMBGNP is applied to solve the data mining problems, including the time series traffic dataset and the UCI benchmark datasets. The superiority of PMBGNP is cleared by comparing it with the conventional EAs and classical data mining methods.

1.3.2 Chapter 3: Hybrid PMBGNP

Chapter 3 addresses the issue of population diversity loss in the topic of PMBGNP by theoretical comparison with classical EDAs, including PMBGA and PMBGP. As a result, a hybrid algorithm is proposed in PMBGNP to maintain its population diversity.

In this chapter, PMBGNP is applied to solve a problem of controlling the agents' behavior, i.e., robot control. The effectiveness of hybrid PMBGNP and its theoretical ability to maintain the population diversity are testified through the empirical studies.

1.3.3 Chapter 4: PMBGNP using both of good and bad individuals

Chapter 4 and chapter 5 focus on proposing a new framework of PMBGNP: studying on the integration of PMBGNP and Reinforcement Learning (RL).

Classical EDAs including PMBGNP generally use truncation selection to estimate the distribution of the promising individuals while ignoring the bad ones. However, various studies in conventional EAs have reported that the bad individuals may affect and help the problem solving. In chapter 4, an extended PMBGNP is proposed to accelerate the evolution of PMBGNP by extracting the good substructures from the bad individuals. This target is achieved by designing a RL technique to learn the experiences of individuals. The learning knowledge is formulated by Q values, which can measure the quality of the sub-structures of PMBGNP. By incorporating the learnt Q values, the good sub-structures from the bad individuals can be extracted and combined into the probabilistic modeling of PMBGNP.

1.3.4 Chapter 5: Reinforced PMBGNP

On the other hand, the algorithm of integrating RL can be used from the other sights of EDA. In most of the advanced EDAs, the complex machine learning techniques are used, such as Bayesian network and Markov/Conditional random fields, but they are very time consuming for constructing the probabilistic model. There is limited work on studying the other machine learning techniques to boost the performance of EDA, such as RL. This chapter proposes an algorithm named Reinforced PMBGNP to incorporate the learning knowledge, i.e., Q values, into the probabilistic modeling of PMBGNP to construct a more accurate model.

1.3.5 Chapter 6: Continuous PMBGNP

PMBGNP and its variants mentioned above are all used for solving the discrete optimization problems. Therefore, they cannot deal with (or directly handle) continuous variables. In this chapter, a continuous PMBGNP algorithm is proposed to directly optimize the continuous variables.

To achieve this task, the continuous variables are represented by Gaussian distribution, where the parameters, such as mean value μ and standard deviation σ, are updated by RL techniques. The proposed algorithm is applied to the robot control, where the experimental results show its superiority over the conventional algorithms.

1.3.6 Chapter 7: Conclusions

The last chapter concludes the thesis by drawing the unique features of PMBGNPs and their contributions.

Based on the current studies on PMBGNPs, the future research directions are finally discussed.

Chapter 2

Probabilistic Model Building Genetic Network Programming (PMBGNP)

Contents

2.1 Introduction

In the last few years, there has been a significant development of Estimation of Distribution Algorithm (EDA) in both theory and practice [Baluja 1994, Mühlenbein 1996, Larrañaga 2002, Zhang 2004]. Different from the conventional evolutionary algorithms (EAs) which 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 the new population. Many studies have investigated that EDA can outperform conventional EAs by avoiding the premature convergence and speeding up the evolution process in some problems [Pelikan 2002a, Zhang 2005, Santana 2008a, Hasegawa 2008]. A large number of studies have been conducted on EDA to propose numerous algorithms. Particularly, from the perspective of individual representation, EDA can be mainly classified into two categories, which are Probabilistic Model Building Genetic Algorithm (PMBGA, or Genetic Algorithm based EDA) [Pelikan 2002b] and Probabilistic Model Building Genetic Programming (PMBGP, or Genetic Programming based EDA) [Shan 2006]. 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 chapter, a novel graph-based EDA named **P**robabilistic **M**odel **B**uilding **G**enetic **N**etwork **P**rogramming (PMBGNP) [Li 2010a, Li 2010c] is described. The aim of developing PMBGNP is to extend EDA from the string and tree structures to the graph structure with higher expression ability, where the directed graph (network) structure of Genetic Network Programming (GNP) [Hirasawa 2001, Mabu 2007b] is employed. Some previous studies have shown the superiority of graph-based EAs in terms of higher expression ability than that of conventional GP [Hirasawa 2001, Mabu 2007b, Teller 1995, Poli 1996, Miller 2000]. GNP is one of such graph-based EAs, 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 firstly designed for solving the problem of controlling the agents' behavior, while in recent years, it has been extended to many other problems, such as multi-agent systems [Eguchi 2006], data mining [Shimada 2006b], elevator system control [Hirasawa 2008] and intrusion detection system [Mabu 2011a], etc. Since PMBGNP uses GNP's directed graph structure, it ensures higher expression ability than the conventional EDAs, i.e., PMBGA and PMBGP.

As many other EAs, standard GNP adopts crossover and mutation to generate the offspring and the major drawback of GNP is the heavy computation required. In GNP, the interrelations of the BBs are kept implicitly between different nodes in the individuals of GNP. In other words, the BBs of GNP is represented by the nodes with connections between each other. Generally speaking, reproducing the BBs is the principle of GNP for solving problems, which can be explained by the BBH straightforwardly. However, as the other conventional EAs, crossover and mutation of GNP sometimes may break the BBs, which causes the problems of premature convergence and local optimum. This inspires the proposal of PMBGNP, which integrates the idea of EDA to the framework of GNP.

This chapter starts by describing the directed graph structure of GNP used in PMBGNP. Section 2.3 describes the probabilistic modeling of PMBGNP in details. Section 2.4 provides the details of producing the new population by sampling the probabilistic model and organizes the detailed procedure of PMBGNP. In section 2.5, PMBGNP is applied to solve the data mining problems, including the time series traffic dataset and UCI benchmark datasets. Finally, the summary of this chapter is presented.

2.2 Directed graph (network) structure of PMBGNP

The primary feature of PMBGNP is that it extends EDA from the string and tree structures to a graph structure, i.e., the directed graph structure of GNP, to represent its individuals. This section introduces such distinguished directed graph structure in details.

2.2.1 Basic structure and comparison

PMBGNP uses the directed graph (network) structure of GNP to represent its individuals. Its basic structure is shown in Fig. 2.1, which can be represented by the phenotype and genotype expression. Phenotype shows the directed graph structure in which 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, while the judgment nodes and processing nodes have some functions depending on the concrete problem. Judgment nodes work as "*if-then*" type decision-making functions to judge the environments by dealing with the specific inputs of the problems, such as the returned sensor information/values of the agents. Each judgment node has several conditional branches corresponding to several judgment results, which can be defined flexibly by the problems. Processing nodes preserve the processing functions to the environments, such as determining the agent's actions. 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 of GNP can handle various combinations of judgments and processing. That is, the evolution can efficiently produce 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 graph structure of GNP never causes the bloat problem of GP [Hirasawa 2001]. Such a directed graph structure can perform quite well by realizing the repetitive processes based on the frequent reuse of nodes, which works like 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 GNP, a class of graph-based EAs has been proposed, such as Parallel Algorithm Discovery and Orchestration (PADO) [Teller 1995], Parallel Distributed GP (PDGP) [Poli 1996], Cartesian GP (CGP) [Miller 2000] and EP [Fogel 1994].

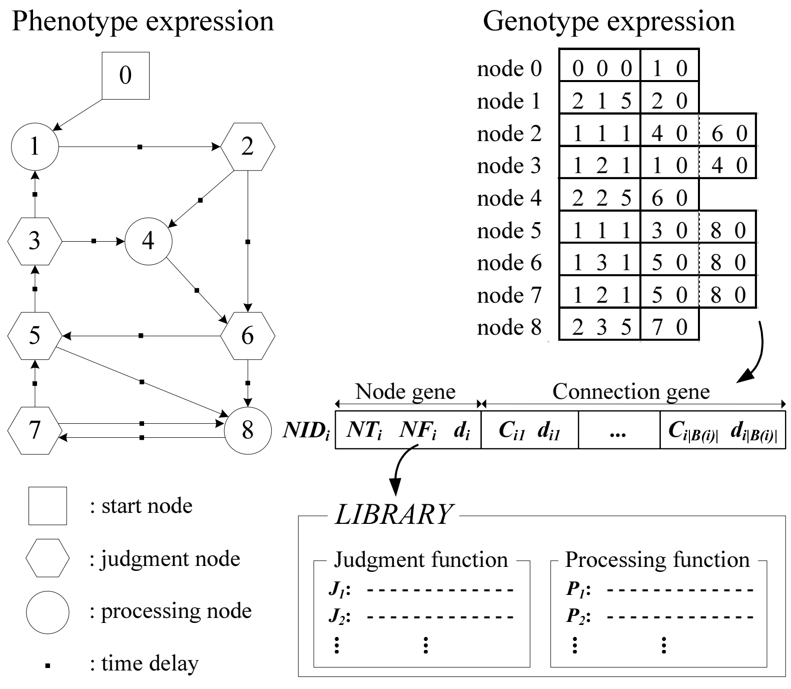


Figure 2.1: Directed graph (network) structure of GNP (and PMBGNP).

Each node of PADO consists of two parts: an action and a branching decision, where the program executes from the start node to the terminal node using stack and explicit indexed memory. In PDGP, the graph is represented as a fixed-sized, 2-dimensional grid, and the running of PDGP programs can be seen as a propagation of the input values from the leaf node to the root node of the graph. Similarly, CGP defines its graph structure as a 2-dimensional grid of nodes. However, its programs are represented by linear chromosomes containing integers, while the executions of the programs are done by the genotype-phenotype mapping from the integers to the grid. EP uses finite state machines (FSM) to form the Markov Decision Process (MDP). However, EP is likely to increase its size of structure for complex problems since all combinations of state and input/output in FSM should be prepared.

There exist fundamental differences between GNP and these graph-based EAs. Firstly, the program of GNP does not consist of the terminal node, and the node transitions end when the task is solved. Secondly, different from PDGP and CGP, where the nodes in the same row or column of the grid are not allowed to be connected to each other, the nodes of GNP's directed graph can be connected arbitrarily, while these node connections are subject to evolution. Thirdly, the nodes of GNP are only connected by necessity, in other words, by evolution. Moreover, by separating judgment and processing functions, GNP program can efficiently generate Partially Observable MDP (POMDP) by selecting only the necessary judgment nodes for the current state of the problems, rather than the MDP of EP [Eguchi 2006].

2.2.2 Time delays

In addition to its distinguished directed graph structure, GNP has time delays in each node and branch. The motivation that introduces time delays in GNP's directed graph is to model the agents with human brain that needs the time for thinking [Eguchi 2006]. Time delays in GNP are designed in three types: time delays spending on judgment nodes, processing nodes and node transitions. These time delays are defined by designers in advance depending on the problems. For examples, for the problems of controlling the agents' behavior, 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 time unit. As a result, the one step of an agent's behavior is defined by the number of time units used in time delays. For example, one step can end when five or more time units are used. That is, the agent can only do fewer than five 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 agents need to solve problems in the constrained time. Another advantage of time delays and steps is to avoid the GNP program to fall into infinite loops [Mabu 2007b]. For example, if processing cannot be executed because of the judgment loops in one GNP individual, the problem cannot be solved by this individual. Since one step ends after five judgments, this individual causing infinite judgment loops will be automatically removed from the population by evolution because of useless steps.

2.2.3 Gene structure and notations

As shown in Fig. 2.1, the nodes of GNP's directed graph are encoded into bit-strings. Each node has a unique identification number NID unchanged during evolution, while the functions of the nodes with the same identification number in different individuals are the same. Let i represent a node number of GNP. NT_i defines the node type, where $NT_i = 0, 1$ or 2 for the start node, judgment node 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 in the remaining part of this chapter is as follows:

N_J: set of suffixes of judgment nodes in one individual.

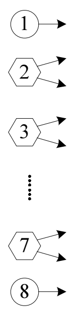
N_P: set of suffixes of processing nodes in one individual.

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 node and processing nodes, while $|B(i)|$ in judgment nodes is determined by judgment function NF_i of node i depending on the concrete problem.

B: set of suffix of branches in one individual.

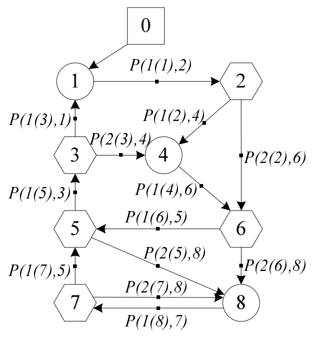
$A(b(i))$: set of suffixes of nodes connected from branch $b(i)$ of node i.



(a). Elements of the directed graph

Branch \ Node	1	2	...	8
1(1)	--	P(1(1),2)	...	P(1(1),8)
1(2)	P(1(2),1)	--	...	P(1(2),8)
2(2)	P(2(2),1)	--	...	P(2(2),8)
1(3)	P(1(3),1)	P(1(3),2)	...	P(1(3),8)
2(3)	P(2(3),1)	P(2(3),2)	...	P(2(3),8)
⋮	⋮	⋮		⋮
1(7)	P(1(7),1)	P(1(7),2)	...	P(1(7),8)
2(7)	P(2(7),1)	P(2(7),2)	...	P(2(7),8)
1(8)	P(1(8),1)	P(1(8),2)	...	--

(b). Probabilistic model of PMBGNP



(c). Example of the individuals sampled by the model

Figure 2.2: Example of the probabilistic model in PMBGNP. "−" in (b) denotes there does not exist connection probability, since the node cannot connect to itself to avoid the infinite loop.

The start node in the directed graph is only used to determine the first node to be executed, where it does not have 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. As a result, we use $G = (N_{\text{node}}, B)$ to denote the directed graph structure of GNP.

The total number of nodes can be calculated by

$$|N_{\text{node}}| = |N_J| + |N_P|, \tag{2.1}$$

and the total number of branches is

$$|B| = \sum_{i \in N_J} |B(i)| + |N_P|. \tag{2.2}$$

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

2.3 Probabilistic modeling of PMBGNP

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

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, as shown as follows

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

Fig. 2.2 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, which can be computed by

$$|P| = |B|(|N_{\text{node}}| - 1). \tag{2.3}$$

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 $P(b(i), j)$, two methods are proposed based on the Maximum Likelihood Estimation (MLE).

2.3.1 Method 1: consider connection information between different nodes

Method 1 considers the connection information between different nodes to calculate the connection probability. This is a straightforward extension of univariate model-based EDAs, such as PBIL and UMDA. The equation of Method 1 is shown as follows:

$$P(b(i), j) = \frac{\sum_{n=1}^{N} \delta_n(b(i), j)}{\sum_{j' \in A(b(i))} \sum_{n=1}^{N} \delta_n(b(i), j')}, \tag{2.4}$$

where,
N: the number of best individuals.
$\delta_n(b(i), j)$: value defined by

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

Using Eq. 2.4, the frequencies of node connections in the promising individuals are counted to constitute the probabilistic model of PMBGNP.

2.3.2 Method 2: consider both connection and transition information between different nodes

In the directed graph structure, generally in one individual, not all the nodes will be used to solve the problems. The node transition will be made by selecting necessary nodes for problem solving. Fig. 2.3 shows an example of the node transitions in an individual. In this instance, the nodes are transited like $1 \to 2 \to 5 \to 6 \to 9$ to solve the problem, which means the information on the connections between these transited nodes is useful for solving the problems. This implies that in the graph structure of PMBGNP, some node connections are frequently used to transit, which

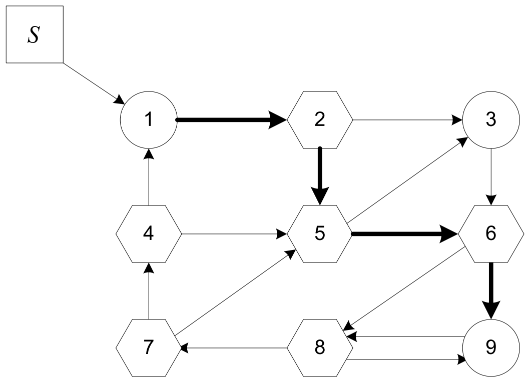


Figure 2.3: An example of node transitions in one individual.

deserves to be considered more, while some are seldom transited. Therefore, Method 2 is further proposed by considering both connection and transition information between different nodes. In this method, the probabilistic model is constructed by:

$$P(b(i),j) = \frac{\sum_{n=1}^{N} \Big(\delta_n(b(i),j) + \eta\sigma_n(b(i),j)\Big)}{\sum_{j' \in A(b(i))} \sum_{n=1}^{N} \Big(\delta_n(b(i),j) + \eta\sigma_n(b(i),j)\Big)}, \tag{2.5}$$

where,
$\sigma_n(b(i),j)$: 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.
η: coefficient.

The probabilities are calculated by considering the connection and transition information between different nodes. The term $\sigma_n(b(i),j)$ is used to consider the reusability of node connections. Parameter η is used to balance the effects of the two factors of connection and transition information. For each branch, we maintain: $\sum_{j \in N_{\text{node}}} P(b(i),j) = 1$. As a result, any individual n can be generated with the following probability

$$P(n) = \prod_{i,j \in N_{\text{node}}} \prod_{b(i) \in B(i)} P(b(i),j). \tag{2.6}$$

After the construction of the probabilistic model in each generation, the following exponential smoothing method is considered to update the current probabilistic model consdering the previous ones:

$$P(b(i),j) = (1-\alpha)P(b(i),j) + \alpha P'(b(i),j), \tag{2.7}$$

Algorithm 1 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 $P(b(i), j)$;
```

Algorithm 2 Algorithm of PMBGNP

```
1: $|Pop| = M$
   $t \leftarrow 0$
2: $Pop(t) \leftarrow$ generate the initial population randomly;
   $Fit(t) \leftarrow$ evaluate the fitness of $Pop(t)$;
3: $Best(t) \leftarrow$ execute truncation selection to select a set of best individuals, where
   $|Best(t)| = N$, $(N \leq M)$;
4: $P \leftarrow$ construct a probabilistic model from $Best(t)$ according to Eq. (2.4) or Eq.
   (2.5);
5: $Pop(t+1) \leftarrow$ generate the new population by Algorithm 1;
   $Fit(t+1) \leftarrow$ evaluate the fitness of $Pop(t+1)$;
6: $t \leftarrow t+1$
   if the termination conditions are not met, go back to 3.
```

where,
α: smoothing rate, and $\alpha \in (0, 1)$.
$P'(b(i), j)$: the probabilistic model of the previous generation.

2.4 Generation of the new population and algorithm of PMBGNP

In this section, the generation of the new population using the constructed probabilistic model is described.

As discussed in section 2.2, the number of nodes and the number of branches in each node are predefined and unchanged during evolution. Therefore, it is easy to generate a new individual by sampling the probabilistic model. The detailed description of the algorithm for generating a new individual is shown in Algorithm 1.

As a result, this process is repeatedly executed M times for generating total M individuals. The time complexity of generating one individual is $O(|B|)$.

The pseudocode of PMBGNP is shown in Algorithm 2, which is similar to conventional EDAs.

It is clearly shown that PMBGNP is a variant of EDA which extends it to the graph-based EAs. The proposed PMBGNP is an extension of conventional univariate model-based EDAs in terms that the probabilistic model is generated by selecting branches.

2.5 PMBGNP for class association rule mining

Most of the current EDAs are mainly applied to solve the benchmark problems of GA and GP, such as function optimization and symbolic regression, where an essential challenge is to explore EDA to novel problems in this research field. On the other hand, due to its distinguished directed graph structure, GNP has been widely applied to various applications, such as data mining [Shimada 2006b], elevator system control [Hirasawa 2008], intelligent agents [Mabu 2007b], stock trading [Mabu 2011b] and multi-agent system [Eguchi 2006], etc.

Particularly, GNP has been successfully applied to data mining field by proposing an algorithm called GNP-based class association rule mining (GNP-CARM). The objective of data mining is to find patterns (generally called *rules*) whose attributes are strongly correlated with each other. CARM is one of the most important branches in data mining, which concentrates on discovering class association rules (CARs) from training data for the prediction of unseen testing data. In GNP-CARM, GNP is to used as a tool to extract the CARs existed in the individuals. The rule extraction is done generation by generation through the power of evolution. Such kind of evolutionary computation based CARM has been applied to various applications, such as UCI benchmark datasets [Shimada 2006b], traffic prediction system [Zhou 2008] and intrusion detection [Mabu 2011a]. Since PMBGNP are the extension of GNP, the aim of this section is to apply PMBGNP to the framework of GNP-CARM to improve the efficiency of rule extraction. To verify the effectiveness of PMBGNP, it is applied to solve a data mining problem using CARM for the prediction of time series traffic dataset, and the experiments on UCI benchmark datasets are also presented.

This section briefly introduces the features of time series traffic dataset, and the way of using PMBGNP in GNP-CARM is introduced. Finally, PMBGNP based CARM is applied to the time series traffic dataset and UCI benchmark datasets for the comparison with the classical algorithms.

2.5.1 Time-related class association rules for traffic prediction

As an significant part of Intelligent Transportation System (ITS), traffic prediction system is capable of helping cities maximize their existing infrastructure to avoid the traffic congestion. Reliable analysis of the historical traffic data is an important part for traffic prediction system. Therefore, data mining is capable of analyzing the real-time traffic data to predict the traffic flow.

Table 2.1 shows an example of the traffic database. The traffic flow of each road section of the traffic network is classified as three cases: Low, Middle or High. At each time point, only one case of the traffic flow in the section could happen. Therefore, the form of attributes saved in the traffic database can be denoted like $A_i(*)$, where A_i represents a road section of the traffic database and $*$ represents the case of traffic flow, i.e., Low, Middle or High. The structure of time-related

Table 2.1: An Example of the Traffic Database

	A_i			A_j		
Time	*Low*	*Mid*	*High*	*Low*	*Mid*	*High*
0001	0	1	0	1	0	0
0002	0	1	0	0	1	0
0003	0	0	1	0	0	1
0004	1	0	0	1	0	0
0005	0	1	0	0	0	1

CARs for traffic prediction is as follows:

$$(A_i(*)(t = t_a) = 1) \wedge ... \wedge (A_j(*)(t = t_b) = 1) \Rightarrow (A_c(*)(t = t_c) = 1). \tag{2.8}$$

Here, t_a, t_b and t_c are the time points which satisfy the condition: $t_a \leq ... \leq t_b \leq t_c$. Meanwhile, the first time t_a always equals to 0, so, $t_b - t_a$ represents the time delay between $A_i(*)$ and $A_j(*)$. Such kind of time-related CARs can express the relationships between different attributes with time points, which could be used to predict the traffic flow. For example, $(A_1(Low)(t = 0) = 1) \wedge (A_2(Mid)(t = 2) = 1) \Rightarrow (A_3(High)(t = 8) = 1)$ means if A_1 is *Low* at time 0 and A_2 is *Mid* at time 2, then at time 8, A_3 is *High*.

2.5.2 Rule extraction

The basic structure and concepts of PMBGNP for CARM are almost the same as the conventional GNP-CARM (the details can be found in [Shimada 2006a]). In GNP-CARM, the functions of judgment nodes and processing nodes are defined as follows.

Judgment Node

The judgment nodes represent the attributes of the traffic database. Therefore, each judgment node function is regarded as one attribute of the database.

Processing node

The processing nodes are used to calculate the measures of CARs.

CARs are represented as the connections of judgment nodes. In this chapter, support, confidence and χ^2 measures are used to define the significance of CARs. Candidate rules that satisfy a minimum support threshold, a minimum confidence threshold and a minimum χ^2 threshold can be extracted from each individual generation by generation.

Extraction Mechanism at Stages (EMS) [Zhou 2008] are used, in order to extract the rules of all of the sections in the traffic database. For each consequent attribute $A_c(*)$, a fixed number of CARs are extracted in order to obtain high classification accuracy. N_f denotes the fixed number of rules. Furthermore, in order to deal with a large number of attributes, Attribute Accumulation Mechanism (AAM) [Zhou 2008] are also used to choose a small attribute set which is used to do the rule extraction and is accumulated round by round.

Meanwhile, since it might occur that there does not exist an enough number of rules for all the consequent attributes at the fixed threshold setting, self-adaptive criteria are also used to adjust the thresholds of the support, confidence and χ^2 values, which works like the following: if any new rule cannot be extracted during the recent generations, the significance level of important rules will decrease using the following equation.

$$Measure_{min} \leftarrow Measure_{min} \times s, \tag{2.9}$$

here, $Measure$ denotes the support, confidence and χ^2 values which are used in this chapter. Furthermore, s belonging to (0,1) represents the step size of the decrease for the threshold of the support, confidence and χ^2 values.

2.5.3 Fitness function

Depending on the importance, complexity, novelty and diversity of CARs which are implicitly saved in the individuals, the fitness function is defined as follows.

$$F = \sum_{r \in R} \{\chi^2(r) + 10(n_a(r) - 1) + \alpha_{new}(r) + \alpha_{multi}(r)\}, \tag{2.10}$$

where,
R: set of suffixes of extracted important CARs in the individuals of PMBGNP.
$\chi^2(r)$: χ^2 value of rule r.
$n_a(r)$: the number of attributes in the antecedent part of rule r.
$\alpha_{new}(r)$: additional constant defined by

$$\alpha_{new}(r) = \begin{cases} \alpha_{new} & \text{if } r \text{ is a new rule,} \\ 0 & \text{otherwise.} \end{cases}$$

$\alpha_{multi}(r)$: additional constant defined by

$$\alpha_{multi}(r) = \begin{cases} \alpha_{multi} & \text{if rule } r \text{ has more than} \\ & \omega \text{ kinds of attributes,} \\ 0 & \text{otherwise.} \end{cases}$$

here, constant ω is the threshold to define the number of kinds of attributes of multiple rules.

2.5.4 Construction of probabilistic model

In CARM, the connections of judgment nodes are represented as CARs [Shimada 2006a, Zhou 2008]. Therefore, the probabilistic model is constructed based on the learning of the connection information between the judgment nodes. The connection information of processing nodes does not have to be considered. The connection probabilities between different judgment nodes are calculated to construct the probabilistic model. Method 1 and Method 2 can be directly used for the construction of the probabilistic model.

2.5.5 Evolution of PMBGNP in class association rule mining

The evolutionary algorithm of PMBGNP in CARM is the same as shown in Algorithm 2, where CARs are extracted from the population of PMBGNP in each generation.

1. First, randomly generate the initial population with M individuals.
2. Extract CARs in the current population.
3. Evaluate the fitness values of each individual.
4. Select N individuals with the higher fitness values ($N \leq M$).
5. Construct the probabilistic model from the selected N individuals.
6. Generate M new individuals by sampling the probabilistic model.

The evolution will be executed generation by generation until enough number of CARs are discovered.

2.5.6 Classification

After the rule extraction, the obtained time-related CARs are used to build a classifier which classifies the testing traffic data to predict the traffic flow.

The time difference between the last attribute of the antecedent part and the consequent attribute is called Prediction Span (PS). In order to predict the traffic flow of road section A_c of L time points later, which is called L-step prediction, the rules whose PS is equal or larger than L ($PS \geq L$) are selected to do the classification.

Let k denote a class of the consequent attribute, i.e., $k \in \{Low, Mid, High\}$. R_k is the set of suffixes of the rules in class k which satisfy $PS \geq L$ and whose antecedent attributes match with the testing traffic data. Num_k denotes the number of rules in class k, which satisfy $PS \geq L$. The average matching degree of data d with the rules in class k, i.e., $Match_k(d)$ is calculated as follows.

$$Match_k(d) = \frac{\sum\limits_{r \in R_k} Confidence(r)}{Num_k}, \tag{2.11}$$

The values of $Match_k(d)$ of three classes are compared, where the class with the highest value of $Match_k(d)$ is the classification result.

2.5.7 Simulations

A comparative study of the proposed algorithm and the conventional GNP is made to solve the traffic prediction problems using CARM in this section. The efficiency of rule extraction and the accuracy of traffic prediction are also shown.

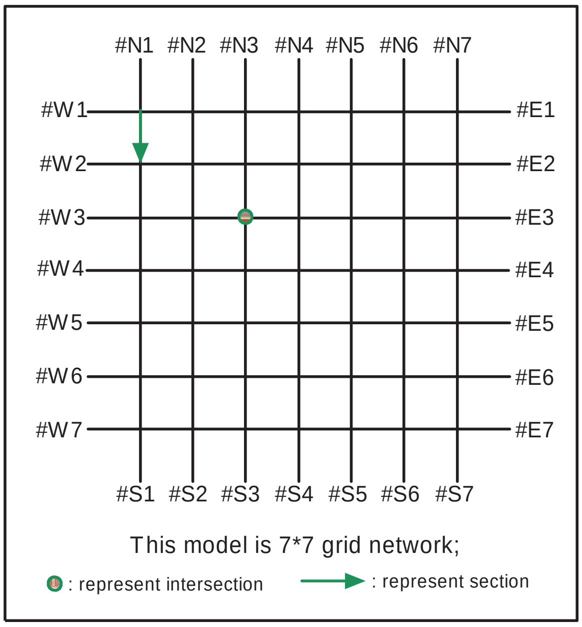

#N1 #N2 #N3 #N4 #N5 #N6 #N7
#W1
#W2
#W3
#W4
#W5
#W6
#W7
#E1
#E2
#E3
#E4
#E5
#E6
#E7
#S1 #S2 #S3 #S4 #S5 #S6 #S7
This model is 7*7 grid network;
: represent intersection
: represent section

Figure 2.4: A simple road map model

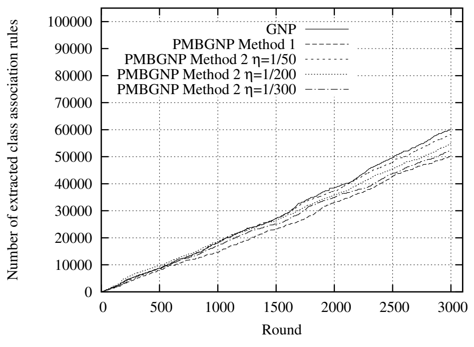
Table 2.2: The traffic database generated by the simulator

Time	$W1N1$,$W2N1$ *Low*	$W1N1$,$W2N1$ *Mid*	$W1N1$,$W2N1$ *High*
0001	0	1	0
0002	0	1	0
0003	0	0	1
0004	1	0	0

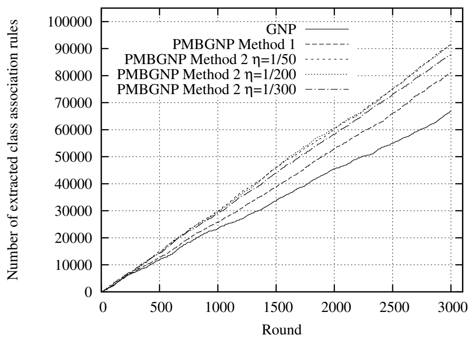
A) Traffic Simulator

A traffic simulator [Zhou 2008] is used to simulate the traffic road in the simulations. The model simulates a 7×7 grid rectangular traffic network as shown in Fig. 2.4, which consists of the edge intersections in the North, South, East and West. The real-world traffic situations, such as traffic lights, traffic jams, speed of cars etc., are all considered in the traffic simulator. Using the simulator, the traffic databases are generated for the simulations. Table 2.2 shows an example. $W1N1$,$W2N1$ represents a section of the road network, and the average traffic flow of each section is discretized to *Low*, *Mid* or *High*. In the road map, there are 112 sections. Furthermore, considering the two directions of each section and 3 kinds of traffic flow of each section, there are total $112 \times 2 \times 3 = 672$ attributes including classes in the traffic database.

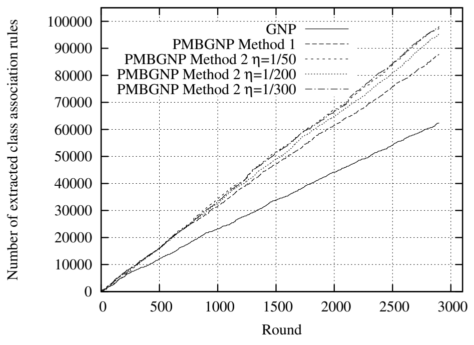
B) Parameter Setting



(a) simulation 1: s=0.9



(b) simulation 2: s=0.85



(c) simulation 3: s=0.8

Figure 2.5: Comparison of the efficiency of rule extraction between PMBGNP and GNP

The parameters of the simulations are set as shown in Table 2.3. In the simulations, the conventional GNP uses four kinds of genetic operators to evolve the individuals: uniform crossover, mutation for functions, mutation for connections and mutation for time delays of judgment nodes [Zhou 2008], while a probabilistic model is adopted to evolve the individuals in the proposed algorithm. In the conventional GNP, the probabilities of crossover and mutation are set to achieve the best results. In the proposed paradigm, Method 1 construct the probabilistic model using the connection information, while coefficient η should be used in Method 2 for considering the transition information. In the databases used in the simulations, we studied that there occur around 200 transitions for the connection between two judgment nodes. Therefore, we set the coefficient η at 1/50, 1/200 and 1/300 to study Method 2. And the smoothing coefficient α in Equation (2.7) is set at 0.1 in order to update the current connection probability by considering the previous connection probabilities. Furthermore, the thresholds of the support, confidence and χ^2 values are set at 0.03, 0.9 and 6.63, respectively, based on many trials.

C) Simulation results and analysis

Table 2.3: Simulation setting for the evolution

Items	Values
Population Size	100
Elite Individuals Size	25
Processing Nodes Size	10
Judgment Nodes Size	100
Time Units Size	800
Attribute Size including classes	672
Generation Size per Round	100
Fixed Number of Rules per Class (N_f)	100

Rule Extraction

In order to study the efficiency of the proposed method, 4 traffic databases generated by the simulator are used to extract the class association rules.

As mentioned earlier, self-adaptive criteria are adopted in order to extract e-nough rules for all sections of the traffic network. Three simulations are carried out, where the coefficient s of self-adaptive criteria is set at 0.9, 0.85 and 0.8, respectively.

The average number of CARs extracted over 20 independent runs are shown in Fig. 2.5 for the 4 databases using the proposed method and conventional method. The results shows that when s is set at 0.9, Method 2 with η=1/50 can get almost the same efficiency as the conventional method, but Method 1 and the others of Method 2 are a little worse. When s is set at a smaller value (i.e., 0.85 or 0.8), both Method 1 and Method 2 are much better than the conventional method.

Furthermore, it is shown that the efficiency of Method 2 is better than Method 1, which means considering both the connections and transitions between nodes can generate a more effective probabilistic model than just considering the nodes connections.

Classification for Traffic Prediction

After the rule extraction stage, the proposed classification mechanism is used to build a classifier for predicting the traffic flow of the simple traffic network. In order to estimate the performance of the proposed method in traffic prediction problems, 4-fold cross-validation is applied. The proposed method and conventional GNP are compared in the L-step traffic prediction. The average classification accuracies for 1-step, 2-step and 3-step traffic prediction are shown in Table 2.4. The simulation results show that the accuracy of GNP-EDAs is almost the same as that of GNP in testing, and in most of the cases even a little better.

Analysis

It is easily understood that the more the value of s decreases, the easier the rules could be extracted, since the threshold of the support, confidence and χ^2 values will decrease more. As a result, the rules with lower significance level will be extracted, leading to the decrease of traffic prediction accuracy. We design the simulations following different settings of s. When s is set at a high value, like 0.9, it might

Table 2.4: Classification accuracy of 4-fold cross-validation. (unit: %)

			1-step prediction			2-step prediction			3-step prediction		
			s=0.9	s=0.85	s=0.8	s=0.9	s=0.85	s=0.8	s=0.9	s=0.85	s=0.8
D 1	GNP		87.8	**87.6**	86.9	86.9	**86.7**	**86.4**	87.1	**86.8**	**86.3**
	Method 1		87.9	87.3	**87.1**	87.2	86.5	**86.4**	87.3	86.4	86.0
	Method 2	$\eta = 1/50$	87.8	86.7	86.5	87.1	86.1	85.8	87.2	86.0	85.5
		$\eta = 1/200$	87.9	86.8	86.6	87.1	86.3	85.7	87.0	86.1	85.4
		$\eta = 1/300$	**88.1**	86.8	86.4	**87.5**	86.3	85.5	**87.9**	86.2	85.4
D 2	GNP		85.8	**85.0**	84.7	84.8	**84.7**	**84.3**	84.7	**84.4**	83.6
	Method 1		85.7	**85.0**	**84.9**	85.0	84.2	84.1	85.2	84.0	**83.8**
	Method 2	$\eta = 1/50$	85.7	84.6	84.5	84.9	84.0	83.7	85.0	83.5	83.3
		$\eta = 1/200$	**85.9**	84.8	84.5	84.9	84.1	83.5	85.0	83.7	83.2
		$\eta = 1/300$	85.8	84.8	84.4	**85.1**	84.2	83.4	**85.6**	83.7	83.0
D 3	GNP		85.6	85.5	84.5	85.0	**84.9**	**84.5**	85.5	**84.7**	84.3
	Method 1		85.8	**86.3**	**84.8**	85.1	84.5	84.3	85.5	84.6	**84.1**
	Method 2	$\eta = 1/50$	85.8	84.8	84.6	85.1	84.1	84.0	85.1	84.2	83.6
		$\eta = 1/200$	**85.9**	84.8	84.5	85.1	84.1	83.6	85.2	84.5	83.4
		$\eta = 1/300$	85.7	84.9	84.5	**85.3**	84.4	83.7	**86.2**	84.4	83.5
D 4	GNP		85.8	**85.6**	84.9	84.6	**84.5**	84.2	84.3	**84.0**	**83.5**
	Method 1		85.5	85.2	**85.1**	84.7	84.4	**84.3**	84.5	83.6	83.2
	Method 2	$\eta = 1/50$	85.6	84.4	84.4	84.7	83.6	83.6	84.2	83.3	83.0
		$\eta = 1/200$	85.6	84.6	84.3	84.6	83.8	83.3	84.1	83.5	82.9
		$\eta = 1/300$	**86.1**	84.7	84.2	**85.0**	83.9	83.3	**85.1**	83.3	82.9

occur that there are just a small number of candidate rules in the searching space for some consequents. Therefore, PMBGNP cannot extract enough mutual interaction information to produce an effective probabilistic model. However, Method 2 can still achieve the same efficiency of the conventional GNP. When the value of s decreases, the number of potential candidate rules increases. As a result, the efficiency of PMBGNP will be significantly improved in terms of extracting many rules by the effective probabilistic model, which leads to the better rule extraction efficiency of the proposed method than the conventional GNP.

In evolutionary algorithm based CARM, the strongly related attributes (corresponding to judgment nodes in GNP and PMBGNP) could be viewed as building blocks, that could be recombined in the next generation. The principle why GNP and PMBGNP could work for finding CARs is decomposing the problems to find strongly associated judgment nodes (building blocks) and recombine them by evolution. Therefore, the simulation results of rule extraction shows that PMBGNP could outperform the conventional GNP when dealing with the problems with a large number of BBs scattered in the search space, such as decomposable problems.

On the other hand, the prediction accuracy is in an acceptable range, even if the value of s decreases (i.e., the accuracy will decrease by less than 1% when s decreases from 0.9 to 0.8), but, the rule extraction efficiency of the proposed method will greatly increase comparing with the conventional GNP by reducing s. Concludingly, when s is set at a relatively small value (i.e., 0.85 or 0.8), the rule extraction efficiency of the proposed method is much better than that of the conventional GNP. Meanwhile, the accuracy of L-step traffic prediction can keep

Table 2.5: Comparison of the experimental results among several methods.(Unit: %)

Method	*Cleveland*	*Breast*
C4.5	78.2	95.0
C4.5 boost	78.3	96.7
CBA	82.8	96.3
CMAR	82.2	96.4
CPAR	81.5	96.0
GNP	83.7	96.9
PMBGNP	84.3	97.6

the same level as the conventional GNP. Therefore, considering the rule extraction efficiency and prediction accuracy as a whole, the proposed algorithm can provide a better trade-off between the efficiency and accuracy to solve traffic prediction problems comparing with the conventional GNP.

D) Complementary simulations on the UCI benchmark datasets

To evaluate the performance of the proposed algorithm, we further compare with the classical CARM methods on the UCI benchmark datasets, where *Cleveland* and *Breast* datasets are used. Several classical methods, i.e., C4.5 [Quinlan 1993], C4.5 boost [Freund 1996], CBA [Liu 1998], CMAR [Li 2001] and CPAR [Yin 2003], are selected for comparison.

For the comparison, it is found from Table 2.5 that PMBGNP can achieve better performance than the classical methods in both *Cleveland* and *Breast* benchmark datasets. On the other hand, the results show that PMBGNP can slightly find better performance than GNP.

2.6 Summary

In this chapter, PMBGNP that extends EDA from bit-string and tree structures to graph structures is proposed. The proposed algorithm collects the mutual interaction information between nodes of individuals to construct the probabilistic model replacing the conventional crossover and mutation for the guidance of evolution. T-wo MLE-based methods are proposed to build the probabilistic model of PMBGNP, which study the connection and transition information between different nodes. A comparative study of PMBGNP and the conventional GNP is made to solve the CARM problems for the prediction of time series traffic dataset. The advantages of using PMBGNP have been demonstrated through the simulations in terms of the efficiency of rule extraction and accuracy of prediction. On the other hand, applying PMBGNP to the UCI benchmark datasets shows its superiority over classical CARM algorithms.

Basically, in this chapter, the high-level framework of PMBGNP is proposed and studied through the data mining problems. In the next chapter, an important issue of EDAs including PMBGNP, i.e., population diversity loss, will be studied. A hybrid PMBGNP is therefore proposed to overcome the diversity loss. Moreover, PMBGNP is applied to solve a novel problem of EDA, controlling the agents' behavior, where a real Khepera robot control is selected.

Note that since this chapter confirms that considering the connection and transition information of different nodes together would achieve better performance, Method 2 of this chapter is used in the remaining chapters of the thesis for the probabilistic modeling of PMBGNP.

Chapter 3

Hybrid PMBGNP (hPMBGNP)

Contents

3.1 Introduction

Most of the current EDAs suffer from the problem that the population diversity of the genetic information will be significantly decreased in the generated population when the population size is not large enough. Many researchers have investigated the diversity loss in PMBGA [Shapiro 2006], especially under the large search space, leading to the premature convergence and local optimum. Therefore, when PMBGA is used to solve problems, its population size is needed to be set at an enough size in order to ensure enough population diversity for global optimum [Pelikan 2002a]. Some studies also investigated that applying mutation operator [Handa 2007] or niching operator [Sastry 2005] to PMBGA could maintain the population diversity.

However, few work has been done on the diversity maintenance in the EDAs with more complex individual structures, such as PMBGP and PMBGNP. For the EDAs with more complex structures, the search space is generally much larger than that of PMBGA's bit-string structure. Consequently, the problem of diversity loss is much more essential than that of PMBGA, since the required sample size in such

EDAs is usually much larger than that of PMBGA. Nevertheless, when solving the real-world optimization problems, the required search space is usually large. As a result, the required population size should be set at a huge number to ensure the enough diversity, which is not the realistic way for problem solving. Therefore, the study on the diversity maintenance becomes much essential in the EDAs with more complex individual structures.

In the tree structure-based PMBGP research, PIPE uses a mutation operator to explore the search space [Salustowicz 1997], while EDP adjusts the calculated probabilistic model by Laplace Correction to avoid the premature convergence [Yanai 2003]. However, most of these algorithms are testified in GP's benchmark problems with not so large search space, such as symbolic regression and boolean function, and the importance of the diversity maintenance is not clarified clearly in PMBGP. As a EDA with graph structures, PMBGNP holds the same problem of the diversity loss as PMBGP, and even more serious. The reason is that, the graph structure and population size of GNP are usually set at a small value to solve problems, which is one of the advantages of GNP [Mabu 2007b], therefore, the sample space becomes much smaller than that of PMBGP, which probably causes the diversity loss.

This chapter focuses on studying the diversity maintenance of PMBGNP to make it work in the problems with large search space than that of classical benchmark problems, such as robot control problems [Li 2010b]. The next section theoretically analyzes the diversity loss of different types of EDAs. Section 3.3, therefore, proposes two techniques to maintain the population diversity of PMBGNP in terms of improving the exploration ability, which are multiple probability vectors and genetic operators. The proposed algorithm is denoted as hybrid PMBGNP (hPMBGNP) and is evaluated theoretically and empirically. Section 3.4 applies the proposed algorithm to a real problem, controlling a kind of autonumous robot, Khepera robot [Cyberbotics , K-Team Corp.]. The empirical study shows the proposed algorithm could significantly maintain the diversity of PMBGNP to solve the problems.

3.2 Diversity loss

The diversity loss of PMBGA has been clarified by Shapiro in [Shapiro 2006] using the trace of the empirical co-variance matrix. However, there is few work on analyzing this issue in PMBGP. This section discuss the significance of the diversity loss in PMBGP and PMBGNP, and we argue that the diversity loss in PMBGP and PMBGNP is more essential than that of PMBGA with binary structures, since the search space in each individual of GP or GNP is larger than that of GA.

Suppose that the set of suffixes of genes in each PMBGA individual is N_{GA}, while the set of suffixes of nodes in each PMBGP individual is also N_{GP}[1]. In GNP,

[1]Generally, in order to avoid the bloating of GP, the structure of GP is designed with some constraints, such as the maximum number of nodes or the maximum tree depth. Here, the maximum number of nodes is selected as the constraint.

the number of nodes and the number of branches are generally predefined and fixed. To simplify the notations, it is supposed that the set of suffixes of nodes in each PMBGNP individual is N_{GNP} (equivalent to N_{node} of Chapter 2) and the set of suffixes of branches in each PMBGNP individual is N_{bra} (equivalent to $B(i)$ of Chapter 2). The size of the search space of these three algorithms can be calculated by

$$\Psi(GA) = 2^{|N_{GA}|}, \tag{3.1}$$

$$\Psi(GP) = \phi^{|N_{GP}|}, \tag{3.2}$$

$$\Psi(GNP) = (|N_{GNP}| - 1)^{|N_{bra}|}. \tag{3.3}$$

Here, ϕ represents the number of functions in each node of GP[2]. There is no self-loop in GNP network structure, since it will cause the infinite loops of GNP programs. Therefore, for each branch of the node in GNP, the candidate nodes to be connected are all the nodes except itself, where $|N_{GNP}|-1$ in Eq. (3.3) represents the number of candidate nodes to be connected. If variables in the search space are treated equally, the probability for each individual to be sampled can be calculated by

$$P_{GA}(ind) = \frac{1}{2^{|N_{GA}|}}, \tag{3.4}$$

$$P_{GP}(ind) = \frac{1}{\phi^{|N_{GP}|}}, \tag{3.5}$$

$$P_{GNP}(ind) = \frac{1}{(|N_{GNP}| - 1)^{|N_{bra}|}}. \tag{3.6}$$

The diversity loss D_{GA}, D_{GP} and D_{GNP} of PMBGA, PMBGP and PMBGNP could be calculated, respectively as follows.

Theorem 1 *If the number of individuals in a population is M, the diversity loss D_{GA} of PMBGA could be represented by the probability that an individual is not sampled in the search space, which is*

$$D_{GA} = [1 - P_{GA}(ind)]^M. \tag{3.7}$$

Proof 1 *When generating one individual, the probability that individual ind is not sampled to the population equals to $1 - P_{GA}(ind)$. Therefore, when generating M individuals, the probability that individual ind is not sampled to the population equals to $[1 - P_{GA}(ind)]^M$. Then, the diversity loss of an individual in GA can be obtained by Eq. (3.7).* $\square$

[2]Although GP consists of function nodes and terminal nodes, this chapter treats them equally.

Theorem 2 *If the number of individuals in a population is M, the diversity loss D_{GP} of PMBGP could be represented by the probability that an individual is not sampled in the search space, which is*

$$D_{GP} = [1 - P_{GP}(ind)]^M. \tag{3.8}$$

Proof 2 *Omitted.* □

Theorem 3 *If the number of individuals in a population is M, the diversity loss D_{GNP} of PMBGNP could be represented by the probability that an individual is not sampled in the search space, which is*

$$D_{GNP} = [1 - P_{GNP}(ind)]^M. \tag{3.9}$$

Proof 3 *Omitted.* □

The proof of **Theorem** 2 and **Theorem** 3 is similar to **Theorem** 1, therefore it would not be shown in this chapter.

From **Theorem** 1, **Theorem** 2 and **Theorem** 3, it is found that, the smaller $P(ind) \in \{P_{GA}(ind), P_{GP}(ind), P_{GNP}(ind)\}$ is, the more serious the diversity loss is. Moreover, there is an inverse relationship between $P(ind)$ and the size of the search space $\Psi \in \{\Psi(GA), \Psi(GP), \Psi(GNP)\}$. Therefore, when the search space becomes larger, the diversity loss is more serious. From Eq. (3.1)-(3.3), it could be found that generally the search space of GP and GNP is much larger than that of GA, since ϕ and $|N_{GNP}|$ is much larger than 2.

On the other hand, when the population size M is small and the probability of each individual to be sampled is small, the diversity loss $D \in \{D_{GA}, D_{GP}, D_{GNP}\}$ will approach near to 1, which means the diversity loss is very serious. Therefore, in the previous research of EDA, the population size M is generally set at large values to obtain the enough population diversity. However, when solving the problems consisting of very large search space, the required population size should be set at huge values, which is almost impossible. GNP is generally designed to solve the complex problems in dynamic environments. Therefore, maintaining the population diversity is much essential in PMBGNP.

3.3 Hybrid PMBGNP

This section focuses on proposing an extension of PMBGNP to maintain its population diversity and to escape from the local optimum of final solutions. Two novel techniques, which are multiple probability vectors and genetic operators, have been proposed to maintain the population diversity of PMBGNP in terms of improving the exploration ability. Since the proposed algorithm is inspired by genetic operators of the conventional evolutionary algorithms, the proposed algorithm is named hybrid PMBGNP (hPMBGNP).

The motivation that makes hPMBGNP innovative is that: firstly, it evolves multiple populations by constructing multiple probability vectors. Secondly, it also applies genetic operators like crossover and mutation to the multiple probability vectors, where the exploration of the search space and population diversity could be improved. One should note that the proposed crossover and mutation are applied to change the probability vectors, not the individual structure which is used in conventional evolutionary algorithms.

In hPMBGNP, there are several populations, i.e., $|R|$. Each population consists of a number of individuals, i.e., M. All the individuals will be initialized by random and evaluated by a predefined fitness function. With their fitness values, the set of promising individuals would be selected. For each population, hPMBGNP constructs a probabilistic model as described in section 2. The probabilistic models are represented by connection probability vectors, therefore, hPMBGNP consists of $|R|$ probability vectors, whose r_{th} one is denoted as P_r $(r \in R)$. Genetic operators such as crossover and mutation are applied to probability vectors P_r to produce new probability vectors $P_r^{'}$.

The new $|R|$ populations will be generated by sampling the probability vectors $P_r^{'}$. In conventional GNP, crossover and mutation are directly used to generate the new population, as a result, the strongly related sub-structures of GNP sometimes will be broken down to produce uninteresting individuals, while the probability model is carried out by learning the structure of promising individuals to guide the evolution of hPMBGNP. Therefore, hPMBGNP inherits the characteristics of EDA that the BBs could be recognized and represented implicitly in the probabilistic model, then the generated population becomes capable of avoiding the breakage of the BBs. On the other hand, crossover and mutation are applied to the constructed model, which means that maintaining the population diversity leads to that PMBGNP can handle the problems consisting of the large search space. The details of the probabilistic model construction and genetic operators in hPMBGNP will be introduced next.

3.3.1 Probabilistic model construction

The probabilistic model of population $r \in R$ is represented as probability vector P_r. P_r^t denotes the probability vector P_r in the t_{th} generation, and $P_r^t(b(i), j)$ represents the connection probability from branch $b(i)$ of node i to node j. The mathematical formulas to calculate the probability vectors in hPMBGNP are the same as the ones used in PMBGNP in section 2. The different point is that in hPMBGNP, the probabilistic model consists of multiple probability vectors, while PMBGNP consists of a single probability vector. Therefore, the probabilistic model could be denoted as follows.

$$P = \{P_r | r \in R\}. \tag{3.10}$$

$$P_r = \{P_r(b(i), j) | i \in N_{GNP}; b(i) \in B(i); j \in A(b(i))\}. \tag{3.11}$$

3.3.2 Genetic operators

Crossover and mutation are designed to produce the new probabilistic model $P^{'}$. The role of genetic operators of the probabilistic model is to explore the probability vectors. In each generation, the constructed multiple probability vector P_r is replaced with the new one generated by crossover and mutation. Tournament selection is used in hPMBGNP to select probability vectors for crossover and mutation. Crossover and mutation operators are carried out subject to the following condition.

$$\sum_{j \in A(b(i))} P_r(b(i), j) = 1, \tag{3.12}$$

for all $i \in N_{GNP}$ and all $b(i) \in B(i)$.

A) Crossover

Crossover is executed between two probability vectors and produces two new probability vectors. Crossover operator exchanges all the probabilities of the selected branches as shown in Algorithm 3. Fig. 3.1 shows an example on how crossover works in hPMBGNP.

Algorithm 3 Crossover of hPMBGNP

```
1: $m, n \in R$
   Select two probability vectors $P_m$ and $P_n$ from $P$ by tournament selection.
2: Each branch $b(i)$ is selected as a crossover branch with the probability of $p_c$.
3: Two probability vectors exchange the probabilities of the corresponding
   crossover branches, i.e., $P_m(b(i), j)$ and $P_n(b(i), j)$ are exchanged.
```

B) Mutation

Mutation is executed in one probability vector to produce a new one. The probabilities of the selected mutation branches are changed randomly by mutation operator, where it should satisfy the condition in Eq. (3.12). The mutation in hPMBGNP is designed as shown in Algorithm 4.

3.3.3 Diversity maintenance of hPMBGNP

In conventional PMBEAs, once the probability[3] in the probabilistic model is equal to zero, the corresponding variable will never be sampled in the future generations.

[3]In PMBGA with no interactions, the probability is represented by $P_{gene}(0)$ or $P_{gene}(1)$, and that of PMBGP is represented by $P_{node}(function)$. In PMBGNP, $P(b(i), j)$ represents the probability.

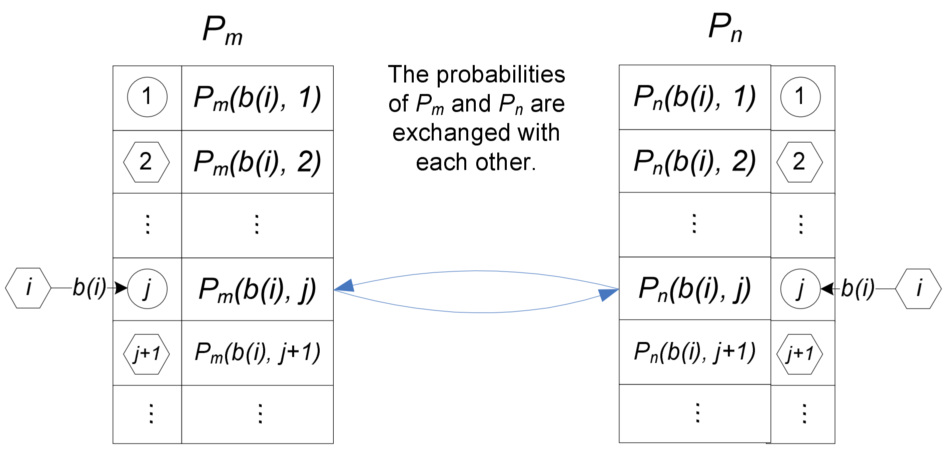


Figure 3.1: Two probability vectors P_m and P_n are selected by tournament selection. In this example, branch $b(i)$ is selected as a crossover branch and their probabilities of P_m and P_n are exchanged with each other to generate the new probability vectors P'_m and P'_n.

Algorithm 4 Mutation of hPMBGNP

```
1: Select one probability vector $P_r$ from $P$.
2: Each branch $b(i)$ is selected as a mutation branch with the probability of $p_m$.
3: For each node $j \in A(b(i))$, randomly generate a positive value $m(j)$.
4: Generate new probability vector $P'_r$ using the following Equation.
```

$$P'_r(b(i), j) = \frac{m(j)}{\sum_{\widehat{j} \in A(b(i))} m(\widehat{j})}$$

This subsection discusses the diversity maintenance of the proposed algorithm comparing with the standard PMBGNP.

Although crossover cannot preserve the population diversity, but it could explore the search space in another way. For example, even though a probability in probability vector P_r is equal to zero, it could be changed to a positive value by exchanging with another probability vector by crossover, which could explore the search space to avoid the local optimum.

Here, the diversity maintenance by the mutation is discussed in the proposed algorithm.

Definition 4 *$o(z)$ represents the number of probabilities equal to zero in branch z.*

Definition 5 *S is the set of suffixes of branches that are not selected as mutation branch. It is very easy to know that $|S| = |N_{bra}|(1 - p_m)$.*

Theorem 4 *For the r_{th} population, the diversity maintenance rate $M(r)$ is defined by Eq. (3.13).*

$$M(r) = \frac{DM(r)}{(|N_{GNP}| - 1)^{|N_{bra}|}}, \tag{3.13}$$

where,
$DM(r)$: the increased size of search space after mutation, and

$$DM(r) = (|N_{GNP}| - 1)^{|N_{bra}|p_m} \prod_{z \in S} [|N_{GNP}| - 1 - o(z)]$$

$$- \prod_{z \in N_{bra}} [|N_{GNP}| - 1 - o(z)]. \tag{3.14}$$

Proof 4 *As described previously, when the probabilities in the probabilistic model are equal to zero, the corresponding variables will never be sampled in the future generations. For branch z, the number of probabilities that is not equal to zero is $[|N_{GNP}| - 1 - o(z)]$. Therefore, the size of the search space of the standard PMBGNP is*

$$\prod_{z \in N_{bra}} [|N_{GNP}| - 1 - o(z)]. \tag{3.15}$$

When mutation is applied to the proposed algorithm, $|N_{bra}|p_m$ branches will be selected as mutation branches. For all mutation branches, their probabilities are always positive values by Algorithm 4. On the other hand, for the branches in set S, their probabilities are still possible to be zero like standard PMBGNP. Therefore, the search space could be calculated by

$$(|N_{GNP}| - 1)^{|N_{bra}|p_m} \prod_{z \in S} [|N_{GNP}| - 1 - o(z)]. \tag{3.16}$$

It is easy to know the size of the search space is increased when mutation is done by comparing Eq. (3.15) and Eq. (3.16), and $DM(r)$ denotes the increased size of the search space as shown in Eq. (3.14). The total search space of PMBGNP can expressed by Eq. (3.3), therefore the diversity maintenance rate can be calculated by Eq. (3.13).

□

It is easy to analyze that $DM(r)$ of Eq. (3.14) is larger than zero, which means the mutation could maintain the population diversity of PMBGNP. Moreover, when the value of $o(z)$ becomes large, $DM(r)$ will also become large when mutation rate is high, therefore the mutation could ensure better diversity maintenance when the population diversity is significantly lost.

3.4 Simulations

The proposed algorithm is evaluated by controlling the movement of Khepera robot and a comparative study among standard GNP, PMBGNP and hPMBGNP is carried out in this section.

3.4.1 Settings of the simulations

A) Settings of the robot

Khepera robot [Cyberbotics , K-Team Corp.] is a small (5.5cm) differential wheeled mobile robot, which includes 8 infrared sensors allowing them to detect the proximity of objects in front of it, behind it, and to the right and left sides of it by reflexion. Each sensor returns a value ranging from 0 to 1023[4]. Two motors corresponding to the left wheel and right wheel can take speed values ranging from -10 to +10. The robot movements are controlled by the speed of the two wheels.

3.4.2 Wall-Following problem and fitness function

Wall-Following problem is used to evaluate the performance of this study. During this task, the robot should avoid the obstacles and find the path to move along the wall quickly. The task ends when a maximum steps ST are reached. The reward and fitness functions are designed based on [Nordin 1998], aiming to obtain the optimal strategy that can control the robot to move along the wall as fast as and as straight as possible.

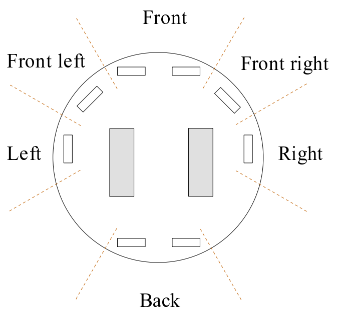
$$Reward = \underbrace{\frac{v_R + v_L}{20}}_{1)} \times \underbrace{(1 - \sqrt{\frac{|v_R - v_L|}{20}})}_{2)} \times \underbrace{C}_{3)}, \tag{3.17}$$

$$Fitness = \frac{\sum_{step=1}^{ST} Reward}{ST}, \tag{3.18}$$

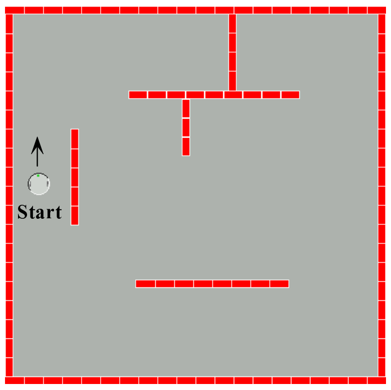
where,
v_R, v_L: the speed of right and left wheels,
ST: user-defined constraint step.
C: value defined by

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

The three marked terms of Eq. (3.17) corresponds to the three factors that should be considered in Wall-Following problem: the robot should be controlled to move 1) as fast as possible, 2) as straight as possible, and 3) along the wall. In every



(a). Khepera robot



(b). Simulation environment

Figure 3.2: Simulation environment.

Table 3.1: Judgment functions for Khepera robot.

Function	Symbol	Description	#Args.	Content of Args.
J_1	JF1	`Judge Front sensor 1`	2	1: $[0, 1000)$
J_2	JF2	`Judge Front sensor 2`		
J_3	JFR	`Judge Front Right sensor`		
J_4	JR	`Judge Right sensor`		
J_5	JB1	`Judge Back sensor 1`		2: $[1000, 1023]$
J_6	JB2	`Judge Back sensor 2`		
J_7	JL	`Judge Left sensor`		
J_8	JFL	`Judge Front Left sensor`		

step, the reward can be calculated by Eq. (3.17), where the final fitness value is the average reward over all ST steps.

In certain respects, the parameter ST can be viewed as a problem size, which can determine the robot's running time. In this chapter, $ST = 500$.

B) Settings of network structures

The node functions used for the Khepera robot are shown in Table 3.1 and 3.2. Each judgment node simulates the corresponding infrared sensor of the Khepera robot, and returns a value probing the position of the robot. The judgment functions of the Khepera robot is designed by its 8 real sensors, as shown in Table 3.1. Discretization process is done in advance to handle the continuous sensor variables. In this chapter, the continuous range $[0, 1023]$ is divided into two intervals, i.e., $[0, 1000)$ and $[1000, 1023]$, to efficiently implement the IFLTE(a, b, c, d) function[5]. Based on the returned sensor information, the robot can consequently determine its

[4]0 means that no object is perceived, while 1023 means that an object is very close to the sensor, almost touching the sensor.

[5]IFLTE(a, b, c, d) function means that if $(a < b)$ then c else d, which is widely used for a range of problems, i.e., robot control.

Table 3.2: Processing functions for Khepera robot.

Function	Symbol	Description	#Args.
P_1, P_2, P_3, P_4, P_5	L(-10), L(-5), L(0), L(5), L(10)	set `Left` motor speed at -10, -5, 0, 5 and 10	0
$P_6, P_7, P_8, P_9, P_{10}$	R(-10), R(-5), R(0), R(5), R(10)	set `Right` motor speed at -10, -5, 0, 5 and 10	0

processing actions by setting the speeds of its two motors. For each motor, the robot can take speeds ranging from the continuous space $[-10, +10]$. In this chapter, the speed that the robot can take is discretized into the set of $-10, -5, 0, 5, 10$. Of course, more potential speeds with many values could be defined, which increases the precision. However, this will cause the exponential increase of the search space. As shown in Table 3.2, the speed of each motor is discretized into 5 candidate values, which has shown to be sufficient in the experiments. The time delay of judgment nodes is set at 1 time unit, that of node transition is set at 0 time unit and that of processing node is set at 5 time units. The robot will take one step of actions when 5 or more time units are used. For example, after executing four judgments, if another one processing is executed, the total time units become 9, which leads to the end of one time step. On the other hand, the simulation ends when the end condition is satisfied, that is, the time step exceeds ST, i.e., 500 in this chapter, which means the task will end when the robot moves 500 time steps.

C) Parameter settings

The parameter settings are shown in Table 3.3. In order to study the proposed algorithm, the number of populations $|R|$ is set at six, not single one as the standard PMBGNP, and the total number of individuals is set at a small value to evaluate the motivation of this work. The crossover and mutation probabilities in GNP and hPMBGNP are set appropriately through the simulations, i.e., $p_c = 0.1$ and $p_m = 0.01$ are used for both GNP and hPMBGNP in this chapter. Each probability vector is constructed for its corresponding population, therefore total six probability vectors exist in hPMBGNP. Two of them are evolved by crossover and the rest four are evolved by mutation.

The number of judgment nodes is set at 40, which means each kind of judgment function has 5 corresponding judgment nodes. Meanwhile, 20 processing nodes exist in an individual, which means each kind of processing function has 2 corresponding processing nodes. The total number of branches $|N_{bra}|$ in an individual is $|N_{bra}| = 40 \times 2 + 20 = 100$, therefore the size of the search space of PMBGNP in the simulations is calculated by

$$\Psi(GNP) = (|N_{GNP}| - 1)^{|N_{bra}|} = 59^{100}. \tag{3.19}$$

The population size is set at relatively small, i.e., 300, in order to evaluate the performance of this study under a small sample size. In each generation, GNP

Table 3.3: Parameter settings.

Parameter	GNP	PMBGNP	hPMBGNP
Number of populations $\lvert R \rvert$	1	1	6
Number of individuals per population M	–	–	50
Total number of individuals N	300	300	300
– Crossover	120	–	–
– Mutation	179	–	–
– Elite	1	1	6
Number of probability vectors	–	1	6
– Crossover	–	–	2
– Mutation	–	–	4
Number of promising individuals	–	150	25
Crossover probability p_c	0.1	–	0.1
Mutation probability p_m	0.01	–	0.01
Number of judgment nodes	40 (5 for each kind of judgment function)		
Number of processing nodes	20 (2 for each kind of processing function)		
η, α	–	0.04, 0.1	

directly preserves the best individual to the next generation, and the remaining individuals are generated by crossover and mutation (120 by crossover and 179 by mutation). For PMBGNP, 299 individuals are generated by sampling the probabilistic model, and combined with the elite individual to form the next new population. In the proposed algorithm, the process of evolving each population is the same as PMBGNP and finally 294 new individuals and 6 elite individuals corresponding to 6 populations are combined to form the new population.

3.4.3 Simulation results

The simulation environment is shown in Fig. 3.2. Fig. 3.3 shows the average fitness curves of the best individuals in each generation over 30 independent simulations.

In the early generations, PMBGNP and hPMBGNP show better fitness than GNP because the probabilistic models have higher evolution ability to find better solutions. However, when the generation goes on, PMBGNP suffers serious diversity loss due to that the sample size (i.e., 300) is much smaller than the required one. Moreover, since PMBGNP does not have any mechanism to preserve the diversity, it quickly converges to a local optimum.

GNP shows better fitness values than PMBGNP, because even if poor individuals are generated in an initial generation, crossover and mutation can explore the search space in each generation, which avoids the premature convergence.

The proposed hPMBGNP shows the best fitness among the three algorithms. Comparing with PMBGNP, the simulation result confirms that the proposed algorithm can maintain the population diversity that avoids the local convergence. On the other hand, the result shows that hPMBGNP has faster convergence than GNP in an early generations, and achieves better fitness value than GNP in the last

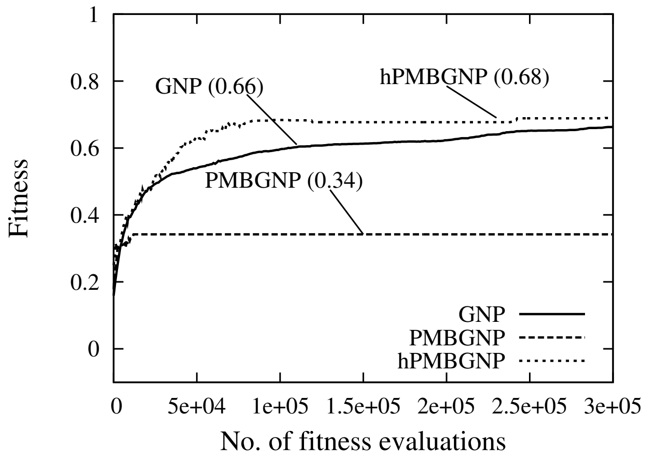


Figure 3.3: Simulation results of GNP, PMBGNP and hPMBGNP in Simulation 1.

Table 3.4: Result of t-test between GNP and hPMBGNP in Simulation 1.

	GNP	hPMBGNP
Mean	0.66	0.68
Standard deviation	0.15	0.06
T-test (p value)	4.72e-01	–

generation.

Table 3.4 shows the results of t-test of the mean fitness values between GNP and hPMBGNP. The p value although shows that although there is not significant difference, hPMBGNP shows competitive results comparing with GNP.

C) Generalization ability

To testify the generalization ability of the proposed algorithm, the robot is e-valuated in an testing environment as shown in Fig. 3.4. The best solutions[6] of GNP and hPMBGNP are used to control the robot. The average fitness values of each algorithm are calculated based on 1000 independent trials. Table 3.5 shows the performance of GNP and hPMBGNP in the testing environment. The average fitness value and standard deviation[7] in Table 3.5 show that the proposed algorithm achieves the best average fitness value, and the t-test result shows that there are significant differences between GNP and hPMBGNP.

[6]30 independent runs can obtain 30 best solutions.

[7]For each best solution, the average fitness value and standard deviation can be calculated from 1000 independent trials. The final average fitness value and standard deviation can be obtained from the 30 average fitness values and standard deviations.

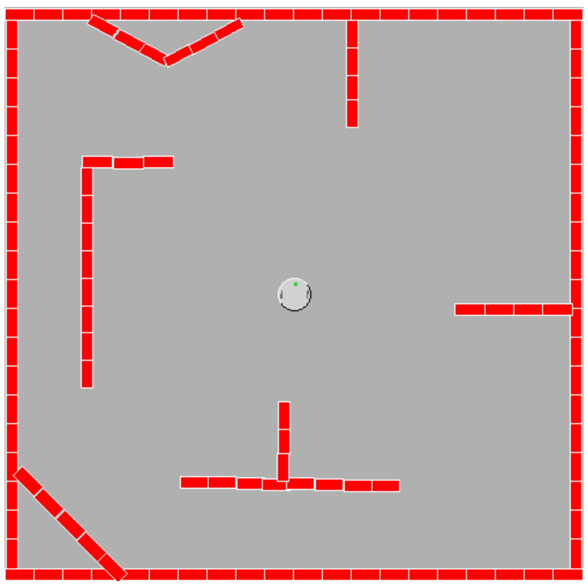


Figure 3.4: Testing environment.

Table 3.5: Result of t-test between GNP and hPMBGNP in the testing environment.

	GNP	hPMBGNP
Mean	0.33	0.37
Standard deviation	0.10	0.07
T-test (p value)	**4.39e-02**	–

3.4.4 Effect of multiple probability vectors

In this subsection, the number of populations $|R|$ and the number of individuals in each population M are set at various values to study the effects of multiple probability vectors. Most simulation settings are the same as Table 3.3, while different settings of $|R|$ and M are used.

The total number of individuals N equals to 300. Since there are so many combinations of $|R|$ and M, we simply define that each population has the same number of individuals, which means $M = N/|R|$. Therefore, four simulations are tested, where the settings of $|R|$ are like $|R| = 4$, $|R| = 5$, $|R| = 6$ and $|R| = 7$[8]. Table 3.6 shows the detailed parameter settings of the 4 simulations. Fig. 3.5 shows the effects of different $|R|$ on the best fitness curves.

For each simulation, at least 2 populations are needed for crossover, and mutation is applied to the remaining populations. Since crossover has little exploration ability, a small number of populations $|R|$ will cause small mutation effects, which can not guarantee enough exploration ability to obtain the best performance. Therefore, in

[8]In the simulation of $|R| = 7$, 6 populations consist of 43 individuals and 1 population consists of 42 individuals.

Table 3.6: Parameter settings of 4 simulations.

Simulation	$\vert R\vert = 4$	$\vert R\vert = 5$	$\vert R\vert = 6$	$\vert R\vert = 7$
Number of individuals per population M	75	60	50	42, 43
Total number of individuals N	300			
Number of probability vectors	4	5	6	7
– Crossover	2	2	2	2
– Mutation	2	3	4	5
Number of promising individuals	35	30	25	20

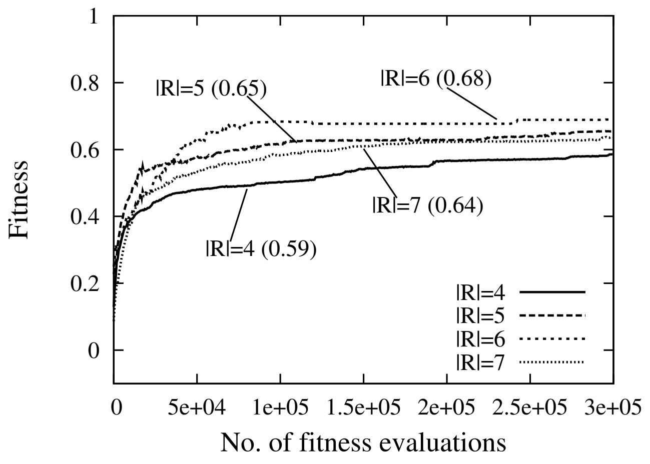


Figure 3.5: Effects of different values of $|R|$.

the case of $|R| = 4$, the simulation achieves the worst performance. On the other hand, too large $|R|$ will make too small population size M, which cannot guarantee enough sample size to estimate an accurate probabilistic model. It is found from Fig. 3.5 that $|R| = 6$ is an appropriate value among the four simulations.

3.4.5 Diversity maintenance comparison between PMBGNP and hPMBGNP

To evaluate the diversity maintenance, we compare the change of $o(z)$ in PMBGNP and hPMBGNP, where $o(z)$ represents the number of probabilities equal to zero in branch z as defined in **Definition** 4.

In PMBGNP, once the probability in the probabilistic model is equal to zero, the corresponding variable will never be sampled in the future generations, while hPMBGNP can explore its search space by mutation to overcome this problem. All the probabilities are considered in the probabilistic models, where the following values are used.

Definition 6 *$SUM(o)$ represents the total number of probabilities equal to zero in the probabilistic model, and we can easily know*

$$SUM(o) = \sum_{z \in N_{bra}} o(z). \tag{3.20}$$

The value $SUM(o)$ can also represent the convergence of the probabilistic model.

However, one should note that exponential smoothing method of Eq. (2.7) causes the probabilities never equal to zero theoretically, once they are not equal to zero in the first generation. The probabilities tend to decrease with respect to the smoothing rate α, if they are smaller than the other probabilities of the same branch. Therefore, in order to calculate the value of $SUM(o)$ to compare the diversity maintenance between PMBGNP and hPMBGNP, the following assumption is used.

Assumption 1 *In the probabilistic model, a probability $P(b(i), j)$ is regarded as zero, if it satisfies*

$$P(b(i), j) < 1.0 \times 10^{-6}. \tag{3.21}$$

Although the probabilities smaller than 1.0×10^{-6} are not actually equal to zero, these probabilities are too small to be sampled. Therefore, **Assumption** 1 is not contrary to the probabilistic model, but makes it possible for $SUM(o)$ to converge.

Simulation 1 is used to compare the diversity maintenance between PMBGNP and hPMBGNP. It can be found from Table 3.3 there are $|N_{GNP}| - 1 = 59$ probabilities in each branch, and each individual consists of $|N_{bra}| = 100$ branches. Therefore, the total number of probabilities in the probabilistic model N_{prob} can be calculated by

$$\begin{aligned} N_{prob} &= |N_{bra}| \times (|N_{GNP}| - 1) \\ &= 100 \times 59 = 5900. \end{aligned} \tag{3.22}$$

Therefore, the probabilistic model of PMBGNP consists of $N_{prob} = 5900$ probabilities. On the other hand, since hPMBGNP consists of $|R| = 6$ probabilistic models, total $6 \times N_{prob}$ probabilities are obtained. In order to make a fair comparison, after counting the number of probabilities equal to zero $SUM(o)$ for each algorithm, $SUM_{\text{PMBGNP}}(o)$ is compared with $SUM_{\text{hPMBGNP}}(o)/6$ in the simulations.

Fig. 3.6 shows the comparison of the diversity between PMBGNP and hPMBGNP. In the figure, the y-axis of PMBGNP is calculated by $SUM_{\text{PMBGNP}}(o)$, while that of hPMBGNP is by $SUM_{\text{hPMBGNP}}(o)/6$. From this simulation, it is shown that the probabilistic model of PMBGNP converges during the evolution process.

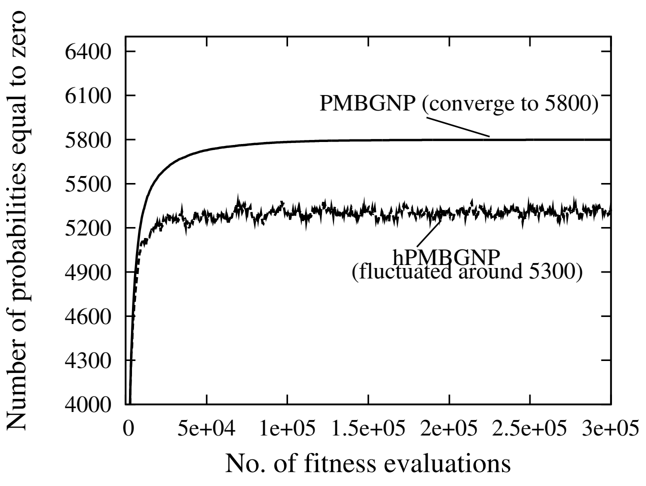


Figure 3.6: Comparison of the diversity between PMBGNP and hPMBGNP.

Theorem 5 *A probabilistic model of PMBGNP converges, if it satisfies*

$$SUM(o) = N_{prob} - |N_{bra}|. \tag{3.23}$$

Proof 5 *If the probabilistic model of PMBGNP converges, the probabilistic model will always produce the same individual, which means every branch is sampled to connect to a specific node with probability 1. In this case, if branch z is sampled to connect to a specific node with probability 1, it means that the number of probabilities equal to zero in branch z is*

$$o(z) = (|N_{GNP}| - 1) - 1 = |N_{GNP}| - 2. \tag{3.24}$$

Then, the total number of probabilities equal to zero in the probabilistic model is

$$\begin{aligned} SUM(o) &= |N_{bra}| \times o(z) = |N_{bra}| \times (|N_{GNP}| - 2) \\ &= |N_{bra}| \times (|N_{GNP}| - 1) - |N_{bra}|. \end{aligned} \tag{3.25}$$

By applying Eq. (3.22) to Eq. (3.25), ***Theorem*** *5 is proven.*

□

Based on **Theorem** 5, it can be easily found that the probabilistic model of PMBGNP has converged in this simulation, since

$$\begin{aligned} SUM_{\text{PMBGNP}}(o) &= N_{prob} - |N_{bra}| \\ &= 5900 - 100 = 5800. \end{aligned} \tag{3.26}$$

On the other, mutation can avoid the convergence of the probabilistic model of hPMBGNP. From Fig. 3.6, it is found that $SUM_{\text{hPMBGNP}}(o)/6$ is fluctuated around 5300, and never converge. Therefore, hPMBGNP can maintain the diversity to find the optimal solution gradually.

3.5 Summary

In this chapter, the issue of the population diversity loss in PMBGNP has been studied by the theoretical comparison with conventional EDAs. The analysis shows that the diversity loss is especially serious in EDAs with more complex structures, such as PMBGP and PMBGNP. Based on the analysis, a hybrid PMBGNP is proposed to maintain the population diversity through multiple probability vectors and genetic operator. This chapter theoretically analyze the diversity maintenance of the proposed algorithm. It is further applied to control the movement of Khepera robot, where the experimental results show the superiority of the proposed algorithm.

Until now, the basic framework of PMBGNP and its diversity maintenance have been studied. The performance of PMBGNP has been verified in both data mining and the problems of controlling the agents' behavior, such as robot control. The next chapter will focus on studying the improvement of PMBGNP by combining it with Reinforcement Learning (RL) techniques. From the perspective of applications, the problems of controlling the agents' behavior will be emphasized in the remaining chapters.

Chapter 4

PMBGNP Using Both of Good and Bad Individuals

Contents

4.1 Introduction

Although any selection strategy [Zhang 2004] can be used, most of the current classical EDAs including PMBGNP generally uses truncation selection to select the promising individuals for the probabilistic modeling [Mühlenbein 1996, Pelikan 2002a, Yanai 2003, Li 2010c], while the bad ones are ignored. In truncation selection, the individuals with the highest fitness values (good individuals) are selected, while the ones with the lowest fitness values (bad individuals) are omitted. However, many studies have reported that utilizing the bad individuals would benefit the evolution process [Yu 2008, Brownlee 2008], which leads to an important research topic in EDAs. This is because they generally contain some useful information that may benefit the further search of EDAs.

In general, there are two ideas to utilize the bad individuals in EDAs. The first idea is directly taking into account the bad individuals to the construction of probabilistic models. In certain aspects, this work is related to the selection 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

[Brownlee 2008], the authors selected a part of the worst individuals for 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 bad individuals will break the useful information of good ones to construct an inaccurate model. Another attempt called EDAs without explicit selection [Yanai 2003, Munetomo 2008] 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 of these methods is that they sometimes do not achieve good performances since it is hard to control the weights of different individuals.

The second idea is to use the bad individuals to filter sample errors [Miquélez 2004, Hong 2009]. [Miquélez 2004] 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, [Hong 2009] proposed a method that estimates two probabilistic models corresponding to the good and bad individuals, where the model of the good individuals is used to sample new individuals and the one of the bad individuals is applied to filter sample errors. These research showed some advantages of convergence speed over standard EDAs.

This chapter proposes a novel method to use the bad individuals [Li 2012]. The point is to extract the useful information of bad 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, 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 [Chen 2010]. However, for most problems whose fitness functions are designed to evaluate the quality of the entire individuals, this task becomes hard to achieve.

To extract the useful sub-structures from the bad individuals, a RL technique named Sarsa learning [Sutton 1998] is used to combine with PMBGNP. In the proposed method, 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 Khepera robot control, and compared with some other methods of utilizing the bad individuals proposed in EDA community. It is expected that a novel viewpoint would be provided to EDA from this method.

The next section describes the proposed method in details. Section 4.3 introduces the related methods of using the bad individuals in PMBGNP for comparison, which are inspired by the other work of this topic in EDA. The experimental study is carried out in Section 4.4. Finally a summary of this chapter is presented.

4.2 The proposed method

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

4.2.1 Preliminaries

In the problems of controlling the agents' behavior, each PMBGNP individual $n \in G$ can be viewed as a *policy* of the agents [Sutton 1998], where the transitions among the nodes of the individual are used to control the agents.

Definition 7 (Episode) *Given a PMBGNP individual $n \in G$, an episode is defined by the sequence of node transitions obtained during the execution of individual n.*

Definition 8 (State) *State s is defined as a branch of a node in G. Therefore, the set of states $\mathcal{S}$ refers to the set of branches in G, represented by B ($\mathcal{S} = B$).*

Definition 9 (Action) *Action a is defined as a node in G. Therefore, the set of actions $\mathcal{A}$ corresponds to the set of nodes N_{node} in G ($\mathcal{A} = N_{node}$).*

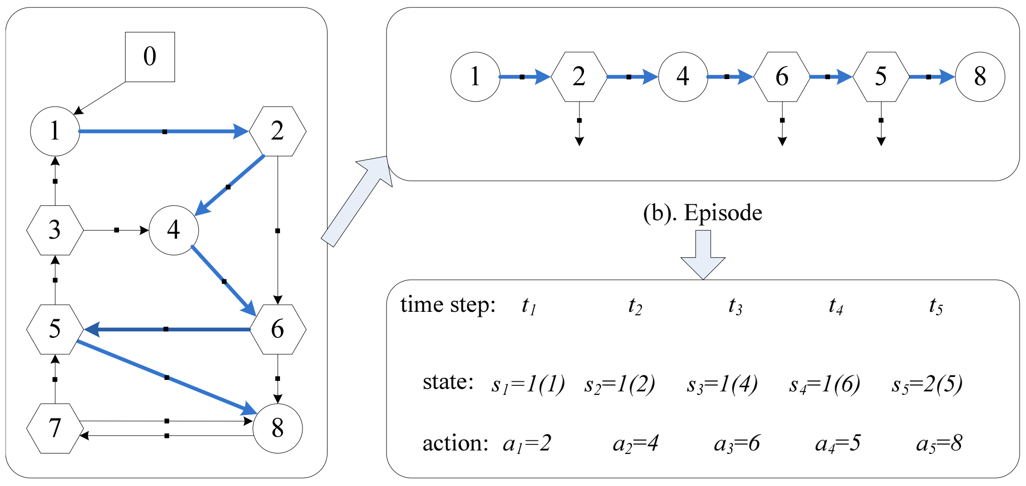
With such definitions, the PMBGNP population with total M individuals can generate M episodes, which can be further factorized to M sequences of state-action pairs. For each episode, the state-action pairs can be represented as follows

$$(\mathcal{S}, \mathcal{A})_n = \{(s_1, a_1)_n, (s_2, a_2)_n, ..., (s_L, a_L)_n\}, \tag{4.1}$$

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

Fig. 4.1 depicts an example of the procedures 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. 4.1 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 10 (State-action pair) *A state-action pair (s, a) corresponds to a node connection from branch s to node a.*


0
1
2
3
4
5
6
7
8
1 2 4 6 5 8
(b). Episode
time step: t_1 t_2 t_3 t_4 t_5
state: s_1=1(1) s_2=1(2) s_3=1(4) s_4=1(6) s_5=2(5)
action: a_1=2 a_2=4 a_3=6 a_4=5 a_5=8

(a). Individual (c). Sequence of state-action pairs (node transitions)

Figure 4.1: An example of the procedures to factorize a PMBGNP individual to the sequence of state-action pairs.

4.2.2 Reinforcement Learning

Reinforcement Learning (RL) is a powerful technique to study the interactions between agents and environments. The aim of RL is to find a policy $\pi(s, a)$ which maximizes the expected future discount sum of rewards received, where the policy $\pi(s, a)$ can be generally represented by the sequences of state-action pairs. In the case of Temporal-Difference (TD) Learning [Sutton 1998], the optimal policy is generally approximated by updating the value functions, i.e., state-action values (Q values) $Q(s, a)$. Among various classical methods, Sarsa Learning [Sutton 1998] is an on-policy method, where the Q values are updated based strictly on the true actions the agent takes, which is different from the off-policy based Q Learning updating the Q values by hypothetical actions (the maximum available action). In other words, Q Learning commits to learn the deterministic optimal policy and the exploration is separated by using the action selection policies, i.e., ε-greedy policy, while Sarsa Learning follows the true experience of the agents to update the Q values. Since the aim of applying RL to this work is to collect the experience of individuals, Sarsa Learning is therefore selected as the fundamental basis for the updating of Q values in this chapter.

At time step t, the main function of Sarsa Learning 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 the following:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big], \tag{4.2}$$

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

4.2.3 Identification of useful sub-structures

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

A) 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:     $n++$
```

In one generation, Sarsa Learning is applied to update the Q table M times. With **Definition** 8 and 9, 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:

$$Q(b(i), j) \leftarrow Q(b(i), j) + \alpha\Big(r_j + \gamma Q(b(j), k) - Q(b(i), j)\Big), \tag{4.3}$$

The immediate reward r_j can be defined flexibly depending on different problems. In this chapter, 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. 4.3 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 judges whether state-action pair $(b(i), j)$ is good or not.

B) Extraction of useful sub-structures

Given state $b(i)$, the set of Q values $Q(b(i), \mathcal{A})$ and the number of good actions N^G, the procedure to extract the set of good actions $A(b_i^G)$ is as follows:

```
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:     $count++$
```

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 section, the bad 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 the bad individuals can be extracted and viewed as useful sub-structures, which are further incorporated into the probabilistic model.

4.2.4 Proposed probabilistic model

The population consisting of M individuals are separated to two classes, which consists of N good individuals and $M - N$ bad individuals. By utilizing useful sub-structures of $M - N$ bad individuals to the probabilistic model, the original probabilistic modeling of PMBGNP Eq. (2.5) can be rewritten as follows:

$$P(b(i), j) = \begin{cases} \frac{1}{Z(b(i))} \left(\sum_{n=1}^{M} \Big(\delta_n(b(i), j) + \eta\sigma_n(b(i), j) \Big) \right) & \text{if } j \in A(b_i^G), \\ \\ \frac{1}{Z(b(i))} \left(\sum_{n=1}^{N} \Big(\delta_n(b(i), j) + \eta\sigma_n(b(i), j) \Big) \right) & \text{otherwise}, \end{cases} \tag{4.4}$$

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

$$\begin{aligned} Z(b(i)) = & \sum_{j' \in A(b(i))} \sum_{n=1}^{N} \Big(\delta_n(b(i), j') + \eta\sigma_n(b(i), j') \Big) + \\ & \sum_{j' \in A(b_i^G)} \sum_{n=N+1}^{M} \Big(\delta_n(b(i), j') + \eta\sigma_n(b(i), j') \Big). \end{aligned}$$

It implies that the good sub-structures of the bad 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 the bad individuals will be directly taken into account for the probabilistic model construction.

4.3 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 the good individuals for the model construction.

Meanwhile, in order to testify the proposed method, the other methods to utilize the bad 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 [Brownlee 2008].
- 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:

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

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

- Method 3: the method is designed based on [Hong 2009]. In this method, two probabilistic models are constructed by standard PMBGNP which are called *good* model P^G and *bad* model P^B. Good model is constructed by estimating the probabilities from the good individuals, while bad model by bad individuals. P^G is used to sample new individuals, which is the same as standard PMBGNP. However, P^B also plays an important role to filter the sample errors. For example, given individual n generated by P^G, two probabilities that individual n can be sampled from P^G and P^B, respectively, denoted as $P^G(n)$ and $P^B(n)$, are calculated. These two probabilities are compared. If $P^G(n) \geq P^B(n)$, individual n is generated successfully, otherwise it is denoted as a sample error.

4.4 Simulations

A comparative study is carried out to solve the Wall-Following problems of Khepera robot control, as shown in Chapter 3. The reward and fitness functions of Wall-Following problems are the same as that of Eq. (3.17) and (3.18). The predefined steps ST is set at $\{100, 300, 500\}$ to study the performance of the proposed method in the problems with different problem sizes.

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 bad individuals due to their low fitness values, it is easily observed that some good state-action

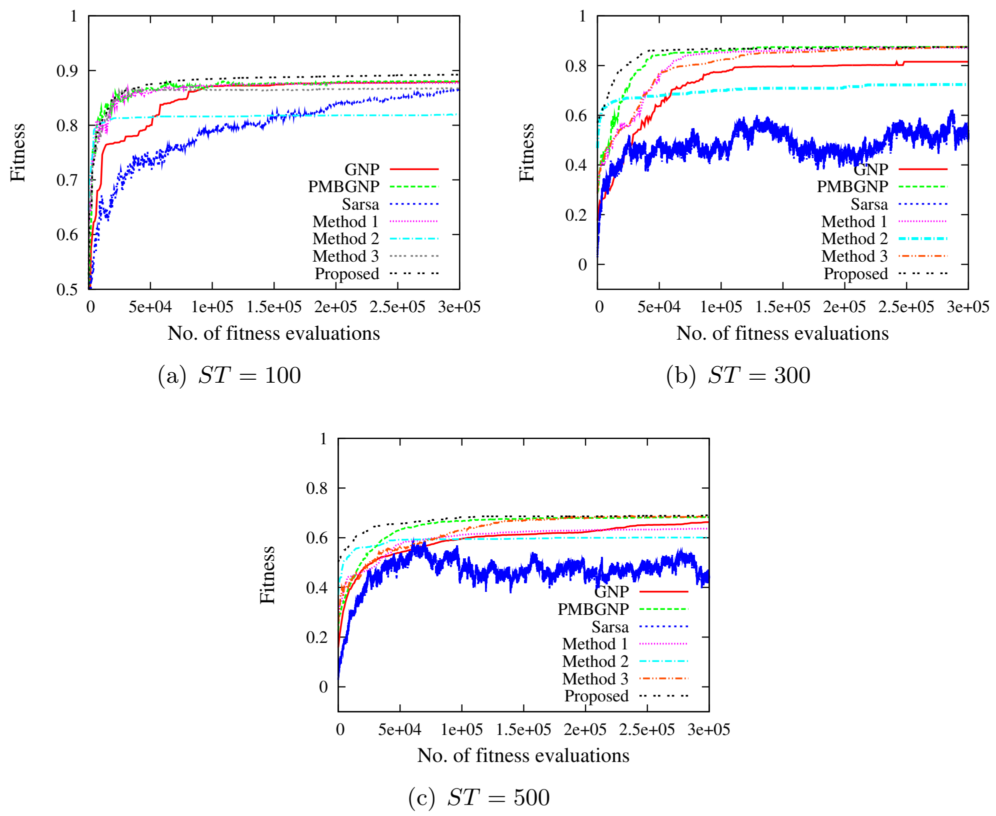


Figure 4.2: Fitness curves in three Wall-Following problems.

pairs still exist in these bad 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 speed up. Accordingly, the objective of this chapter and the effectiveness of the proposed method can be verified by the simulations.

4.4.1 Experimental settings

The settings of the directed graph of PMBGNP are the same as that of Chapter 3. 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.

As discussed in Chapter 3, PMBGNP suffers from the problem of the diversity loss when the population size is not large enough. As a result, to ensure enough population diversity for avoiding premature convergence, 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 good individuals, and η is set at 0.04. PMBGNP directly uses the good individuals to construct the probabilistic model. On the other hand, Method 1 further selects the worst 20%

Table 4.1: The fitness (std. dev.) results.

	$ST = 100$	$ST = 300$	$ST = 500$
GNP	**0.88** (0.02)	0.82 (0.09)	0.66 (0.15)
PMBGNP	0.88 (0.02)	0.86 (0.05)	**0.68** (0.09)
Sarsa	0.87 (0.04)	0.52 (0.21)	0.49 (0.22)
Method 1	0.88 (0.01)	**0.87** (0.02)	0.64 (0.07)
Method 2	0.82 (0.08)	0.72 (0.12)	0.60 (0.10)
Method 3	0.87 (0.01)	**0.87** (0.02)	**0.68** (0.07)
Proposed	**0.89** (0.01)	**0.87** (0.02)	**0.68** (0.09)

individuals to combine with the good individuals for the model construction. In Method 3, the remaining 50% individuals are denoted as bad individuals for the bad 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.

In Sarsa Learning, the state space is defined by the combinations of returned sensor values and the action space is defined by different settings of robot's speed. Therefore, the number of states is $2^8 = 256$, since the domain of each sensor value is divided into 2 intervals ([0,1000) and [1000,1023]). The number of actions is $5 \times 5 = 25$. The learning rate and discount factor of Sarsa are set at 0.1 and 0.9, respectively.

The terminal condition is defined by the maximal number of fitness evaluations, i.e., 300,000 in this chapter.

4.4.2 Simulation results

A) Fitness results

Table 4.1 shows the average fitness and standard deviation over 30 independent runs, and the fitness curves are shown in Fig. 4.2.

The results show that among three problems, PMBGNP achieves better performances than conventional GNP and Sarsa due to its probabilistic modeling. Sarsa shows the worst results among all cases. This is because it has to update a huge size of Q table (256×25 Q values in each step), which causes the slow learning speed to find the optimal solution.

On the other hand, the methods that utilize the bad individuals for the probabilistic modeling are summarized as follows:

(1) Method 1: In simple problems, i.e., $ST = 100$ or 300, Method 1 can achieve

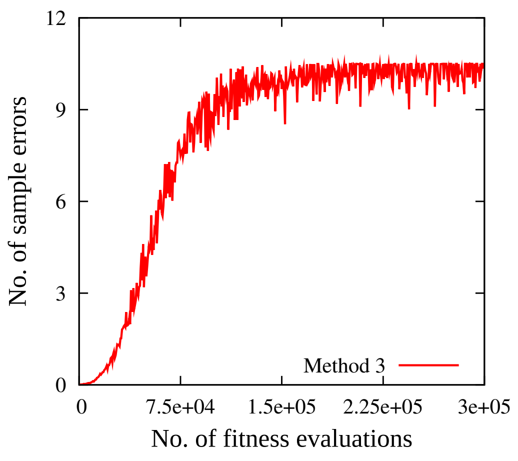


Figure 4.3: Average number of sample errors generated during the evolution process by Method 3.

similar performance with PMBGNP. However, in the case of $ST = 500$, Method 1 cannot obtain good result. This is because the worst individuals are treated equally with the good individuals for the probabilistic modeling, which sometimes provides wrong information and destroy the probabilistic model. It is also mentioned in [Brownlee 2008] that Method 1 can only work well in some problems.

(2) Method 2: In Method 2, one important drawback is 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 chapter achieves the worst results. The results show that although Method 2 allows fast convergence speeds, it quickly falls into local optima.

(3) Method 3: In Method 3, the good model P^G is constructed to generate new individuals, while bad model P^B is used to filter the sample errors. The fitness results show it achieves 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^B becomes more and more similar to P^G. Fig. 4.3 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 similar fitness results to that of PMBGNP. However, it can be seen that the convergence speed of the proposed method is much faster than PMBGNP especially in the complicated problems like $ST = 300$ and $ST = 500$ as shown in Fig. 4.2. On the other hand, comparing with the other methods that utilize the bad individuals, the proposed method can obtain the best fitness results.

B) Required fitness evaluations (RFEs)

In order to compare the effectiveness of this work, the average number of required

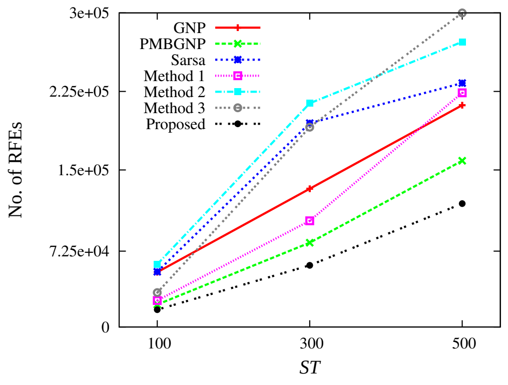


Figure 4.4: Comparison of the required fitness evaluations.

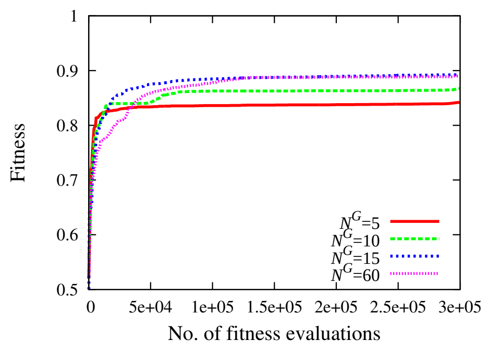
Table 4.2: The average number of required fitness evaluations (std. dev.) and t-test results.

		GNP	PMBGNP	Sarsa	Method 1	Method 2	Method 3	Proposed
$ST = 100$	mean	52300	21467	52800	25347	59867	30905	**16567**
	(std. dev.)	(47379)	(13373)	(51392)	(27585)	(72637)	(26115)	(14916)
	t-test (p value)	**6.1e-04**	7.4e-02	**1.6e-03**	**2.8e-02**	**6.0e-04**	**1.9e-03**	—
$ST = 300$	mean	132000	80500	194707	101400	213633	190850	**58900**
	(std. dev.)	(101807)	(63884)	(99035)	(63904)	(111499)	(92185)	(29222)
	t-test (p value)	**1.0e-03**	**2.4e-02**	**5.5e-08**	**2.8e-03**	**5.9e-08**	**3.6e-08**	—
$ST = 500$	mean	211833	158800	232970	223667	272347	300000	**117907**
	(std. dev.)	(111542)	(88578)	(100870)	(75976)	(62144)	(0)	(58466)
	t-test (p value)	**1.9e-04**	**3.1e-02**	**7.6e-07**	**8.9e-07**	**1.2e-11**	**5.8e-17**	—

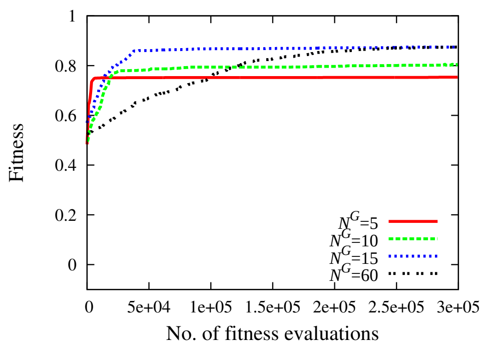
fitness evaluations of different methods is compared to control the robot moving around the wall successfully. The average number of required fitness evaluations is calculated as follows: For each successful simulation run, the exact number of fitness evaluations is counted, while the maximum number of fitness evaluations, i.e., 300,000, is used for each failed run. Then, these numbers of 30 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. 4.4, and the results of the t-test are shown in Table 4.2. The simulation results indicate that the proposed method shows the fastest convergence among the methods to solve the task successfully for the three Wall-Following problems with different complexities. In addition, the results of the t-test show that there are statistically significant differences between the proposed method and the other ones.

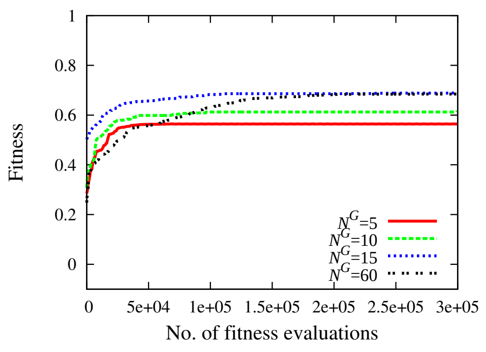
Conclusively, the experiments show that with the correct extraction of useful sub-structures of the bad individuals, the proposed method can speed up the evolution process of PMBGNP, while some other methods proposed in EDAs community show some limitations to achieve good performances.



(a) $ST = 100$



(b) $ST = 300$



(c) $ST = 500$

Figure 4.5: Effect of parameter N^G in the three Wall-Following problems.

C) 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 4.3.

Fig. 4.5 shows the fitness curves of the proposed method with different settings of N^G in the three Wall-Following problems. 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. Among the three problems in this chapter, the appropriate setting of N^G is 15.

4.5 Summary

In this chapter, a novel method has been proposed in PMBGNP to utilize the bad individuals. The conventional methods showed some disadvantages, such as premature convergence and difficulty of parameter control, while the proposed method provides another viewpoint to utilize the bad individuals. That is, the proposed method applies RL to identify and extract useful sub-structures of the bad individuals, while the sub-structures are used in the probabilistic modeling of PMBGNP. The results on Wall-Following problems show that the proposed method can accelerate the evolution of PMBGNP in terms of requiring smaller number of fitness evaluations.

On the other hand, this chapter provides a novel viewpoint to utilize the integration of EDA and RL. The idea of integrating RL can be used from the other sights of EDA. In most of the advanced EDAs, the complex machine learning techniques, such as Bayesian network and Markov/Conditional random fields, they are very time consuming for constructing the probabilistic model. The construction of the probabilistic model itself is an optimization problem. Consequently, there is a trade-off between the probabilistic modeling and time cost in most of the current research on EDA. There is limited work on studying the other machine learning techniques to boost the performance of EDA, such as RL. The next chapter will focus on proposing an algorithm named Reinforced PMBGNP to incorporate the learning knowledge, i.e., Q values, into the probabilistic modeling of PMBGNP to construct a more accurate model.

Chapter 5

Reinforced PMBGNP (RPMBGNP)

Contents

5.1 Introduction

Standard PMBGNP is mainly based on the frequencies of node connections by simple MLE which is inspired by the classical univariate EDAs, such as PBIL and UMDA. In these algorithms, when an element is observed, i.e., variable/function, a weight of 1 is just simply given to it, which may not measure its importance precisely, however. In other words, all the observed elements are treated equally in the probabilistic modeling, regardless of the significance of them which means different individuals have different fitness values. Therefore, to estimate a more precise model, one may consider this matter into the probabilistic modeling. That is, the observed elements are used in the probabilistic modeling w.r.t. the fitness of their individuals [Yanai 2003, Munetomo 2008]. However, as studied in Chapter 4, this may not guarantee the performance of PMBGNP due to the hard control of the weights [Li 2011].

On the other hand, in most of the advanced EDAs, the complex machine learning techniques, such as Bayesian network and Markov/Conditional random fields, are incorporated, which are very time consuming for constructing the probabilistic model. For example, most of the current advanced EDAs apply Bayesian network to capture the probability distribution of promising individuals. These include the famous BOA, EBNA, LFDA, POLE, etc. In these EDAs, Bayesian network is used to represent the probabilistic model, in which the connections between the variables denoted by the nodes of the Bayesian network actually represent the corresponding interactions. As a result, the construction of the probabilistic model in EDA is transferred to the construction of the Bayesian network through learning the promising individuals obtained by truncation selection. However, learning the Bayesian network itself in every generation is actually an optimization problem. Even the greedy algorithms are generally employed to construct the Bayesian network, this process is still quite time consuming. Improving the efficiency of the probabilistic modeling becomes one of the essential challenges in EDA.

Following this viewpoint, Reinforced PMBGNP (RPMBGNP) [Li 2013] is proposed in this chapter, where RL is combined with evolution to overcome the above problem. Inspired by behaviorist psychology, RL concerns 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 which are further incorporated in the probabilistic modeling. By defining the state-action space using PMBGNP's graph structure, RL is capable of studying the sub-structure (node connections) of PMBGNP individuals during their executions. The learning knowledge, i.e., Q values, are used to build the probabilistic model of PMBGNP, which directly benefit the problem solving of controlling the agents' behavior. More importantly, the learning of Q values through RL only requires linear time cost with respect to the problem size. As a result, the construction of the probabilistic model through RL actually requires similar time cost to the original univariate model-based EDAs, which is time saving comparing with the current Bayesian network-based EDAs.

The rest of this chapter is organized as follows: the next section briefly introduces the previous work related to this chapter, and highlights the contribution of this work by comparing with the others. Section 5.3 describes RPMBGNP in details. The experimental study is carried out in section 5.4 and 5.5. Section 5.6 summarize this chapter.

5.2 Related work and comparison

Many previous studies have successfully applied RL to EC for the problems of controlling the agents' behavior [Downing 2001, Kamio 2005, Mabu 2007b].

In certain respects, this topic can be explained as the combination of global search (population evolution) and local search (individual learning), which has been widely discussed by the *Baldwin effect* [Downing 2001]. It relies on the fact of phenotypic plasticity that an individual has ability to adapt to its environment during its lifetime, where the successful adaptation which increases the fitness results will tend to proliferate in the population [Baldwin 1896]. In order to achieve the adaptability, Downing [Downing 2001] proposed Reinforced GP (RGP) method by adding a special form of leaf nodes called choice nodes to GP's tree structure, where the actions/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, where the learning knowledge are represented by Q values of the state-action pairs. Kamio and Iba [Kamio 2005] proposed another method named GP+RL to integrate GP and Q Learning, where evolution and learning are executed step by step. That is, the evolution is first done to find the roughly optimal tree structures of GP, then Q Learning is carried out to adapt the functions of action nodes in GP. Mabu *et al.* [Mabu 2007b] proposed GNP-RL to integrate GNP and Sarsa Learning, 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. GP+RL uses the real states to define its state space, which is similar to the standard RL. Both of them face a problem that the state space may become extremely large causing the difficulty of 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/sub-node in each node corresponds to the actions. In the other words, the size of its state space is only based on the number of nodes in GNP's graph structure, which is generally relatively small.

The key point of these remarkable methods integrating EC and RL is to create the *adaptation* of individuals to find the global optimum. Therefore, they need to change the original tree and graph structures of GP and GNP to create the adaptation ability. In this chapter, we propose a very different algorithm to integrate EC and RL, named Reinforced PMBGNP (RPMBGNP). The basic idea of RPMBGNP is to create the state-action space using GNP's graph structure, and apply Sarsa Learning to update the Q table of state-action pairs based on the individual executions. At the end of each generation, the learning knowledge, i.e., Q values of all state-action pairs, are incorporated in the probabilistic modeling of PMBGNP. Therefore, the primary objective of RPMBGNP is to improve the evolution efficiency in terms of constructing more accurate probabilistic model by RL, rather than to achieve the adaptation ability. The features of RPMBGNP comparing with conventional methods are:

- Conventional methods change the structures of GP/GNP by introducing sub-nodes, while RPMBGNP does not.

- Conventional methods defines its state-action spaces by either the real states or the introduced sub-nodes, while RPMBGNP defines its state and actions by the graph structure of GNP.
- Conventional methods create multiple Q tables corresponding to each individual, while RPMBGNP only create and maintain one Q table which leads to the faster convergence of the Q table.
- Conventional methods require the action selection policies, i.e., ε-greedy policy, to use the Q values, while RPMBGNP only uses them for the probabilistic modeling.

In EDA field, a univariate PMBGA named Reinforcement Learning EDA (RELEDA) [Paul 2003] is the only limited study on integrating EDA and RL techniques. It introduces a number of parameters denoted by $\boldsymbol{\theta} = \{\theta_1, \theta_2, ..., \theta_n\}$, where θ_i is a parameter corresponding to the allele/variable X_i, and the estimation of the probability distribution is based on the update of $\boldsymbol{\theta}$ designed by the analog of RL in iterative process.

On the other hand, RPMBGNP holds the assumption of RL called *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 follows:

$$\mathbb{P}(s_t|s_{t-1}, s_{t-2}, ..., s_0) = \mathbb{P}(s_t|s_{t-1}). \tag{5.1}$$

Although some tasks are non-Markov, the assumption of Markovian provides a theoretical basis of RL for predicting future rewards and for selecting actions to obtain good performance in RL problems [Sutton 1998]. From this point of view, there is a class of EDAs, especially PMBGAs, based on Markov Random Fields/Markov Network [Santana 2005, Shakya 2006, Brownlee 2009]. However, these methods and RELEDA are used to solve function optimization problems rather than to solve the problems of controlling the agents' behavior as RPMBGNP.

There is another recent work called EDA-RL [Handa 2009], which applies conditional random fields (CRFs) to construct its probabilistic model. It has has been reported to successfully solve simple problems of controlling the agents' behavior, but fail on some complex partially observable problems. On the other hand, RL techniques are not used in this algorithm, even the name RL is indicated which mainly means that it is designed to solve RL problems.

5.3 Reinforced PMBGNP (RPMBGNP)

The main feature of RPMBGNP is to apply RL technique to learn the experience of individuals by approximating the knowledge so-called Q values. Consequently, the learnt Q values can measure the quality of sub-structures of PMBGNP, i.e., node connections, which has been discussed in Chapter 4. However, different from the method proposed in Chapter 4, the objective of using RL in RPMBGNP is to model a more accurate model than the MLE-based model through directly incorporating the Q values, rather than using Q values for extracting useful sub-structures from the bad individuals in Chapter 4. As a result, this chapter can be considered as a straightforward extension of Chapter 4 which makes the study of the integration of PMBGNP and RL more comprehensively.

5.3.1 Updating of Q values

The factorization of individuals to the sequences of state-action pairs has been discussed in Chapter 4.

After defining the directed graph structure G of PMBGNP, a Q table which consists of the $Q(\mathcal{S}, \mathcal{A})$ values for all possible combinations of state-action pairs is generated. By **Definition** 10 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 the probabilistic model of PMBGNP, as shown in Fig. 2.2.(b). The initial $Q(\mathcal{S}, \mathcal{A})$ values can be either 0 or positive constants, which are problem specific. In this chapter, we initialize all $Q(\mathcal{S}, \mathcal{A})$ values at 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, as the instance shown in Fig. 4.1. After obtaining the episodes, Sarsa Learning is applied to update the Q value of each state-action pair. Suppose the current state and action at time step t are respectively $b(i)$ and j, which means 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:

$$Q(b(i), j) \leftarrow Q(b(i), j) + \alpha\Big[r_j + \gamma Q(b(j), k) - Q(b(i), j)\Big], \tag{5.2}$$

where,
r_j: 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 5. 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 which consists of all $Q(\mathcal{S}, \mathcal{A})$ values.

5.3.2 Probabilistic model of RPMBGNP

As described above, with **Definition** 10, the state-action pairs of RL is equivalent to the corresponding node connections of PMBGNP. Consequently, the good state-action pairs will be rewarded with higher Q values by RL, and vice-versa, which

Algorithm 5 Algorithm for the updating of 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) $(\mathcal{S}, \mathcal{A})_n$;
4:     update the corresponding $Q$ values of $(\mathcal{S}, \mathcal{A})_n$ using Eq. (5.2);
5:     $n \leftarrow n + 1$;
```

implies that the collected Q values can explicitly express the quality of node connections of PMBGNP. By using these learnt information properly, we can estimate a more accurate probabilistic model for PMBGNP.

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

$$P(b(i), j) = \frac{\exp\left(\frac{Q(b(i),j)}{T}\right)}{Z(b(i))}. \tag{5.3}$$

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

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

where,
T: temperature parameter.

As indicated in Eq. (5.3), Boltzmann distribution is employed in the probabilistic modeling of PMBGNP to correct its convergence. The reason for introducing Boltzmann distribution is because of the issue of the population diversity in PMBGNP proven in Chapter 3, which causes the large required population size M.

A) Required population size

Use a similar expression of the diversity loss discussed in **Theorem** 3, the diversity loss rate of a node connection is defined as follows:

Definition 11 (Diversity loss rate) *In PMBGNP, the diversity loss rate $DL(b(i), j)$ 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.*

Therefore, we can calculate the diversity loss rate of node connection $(b(i), j)$ by:

Lemma 1 *Given the population with total M individuals, the diversity loss rate of node connection $(b(i), j)$ in PMBGNP is calculated by*

$$DL(b(i), j) = \Big(1 - P(b(i), j)\Big)^M. \tag{5.4}$$

Lemma 1 can be easily proven using the same way of **Theorem** 1 and $DL(b(i), j)$ 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 $DL(b(i), j) = 1$.

The required population size M of PMBGNP can be estimated by discussing its diversity loss. In the simplest way, M can be obtained by discussing its lower/upper bound of required sample size N (the number of best individuals in truncation selection) in the initial generation. This is because in the extremely case, the lower/upper bound of M can be represented by that of N, since $N \leq M$. Therefore, we have

$$M \in \left[\frac{1}{P(b(i^*), j^*)}, \infty\right), \tag{5.5}$$

where $P(b(i^*), j^*)$ is the initial value of the probabilistic model, which equals to $1/(|N_{\text{node}}| - 1)$. Theoretically at least $|N_{\text{node}}| - 1$ individuals should be obtained to sample all node connections in the population. However, simply obtaining the individuals with the lower bound cannot estimate an accurate model, and it is impractical to obtain infinite individuals to construct the perfect model. Therefore, the size of M is generally determined by the problems. Nevertheless, since the probability distribution of PMBGNP is actually the multinomial distribution, the required population size of multinomial distribution could be investigated by confidence interval [Hasegawa 2008], in which the lower bound of M can be represented for a given confidence coefficient. Following this point of view, M could be obtained by the diversity loss rate and a given confidence level ε.

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

$$M \geq \frac{\ln \varepsilon}{\ln\left(1 - \frac{1}{|N_{node}|-1}\right)}. \tag{5.6}$$

Proof 6 *With **Lemma** 1, for a given node connection $(b(i), j)$, its diversity loss rate has the relation of $DL(b(i), j) = (1 - P(b(i), j))^M \leq \varepsilon$. After logarithmic process, the inequality can be rewritten as*

$$M \geq \frac{\ln \varepsilon}{\ln\Big(1 - P(b(i), j)\Big)}.$$

Using the initial value of $P(b(i), j)$, Eq. (5.6) can be obtained. □

As a result, the lower bound of M of PMBGNP is:

$$M_{\text{PMBGNP}} = \frac{\ln \varepsilon}{\ln\left(1 - \frac{1}{|N_{\text{node}}|-1}\right)}. \tag{5.7}$$

Similarly, the lower bound of M of univariate model-based PMBGA and PMBGP, i.e., UMDA and PIPE, can be calculated by replacing the initial values of the probabilistic model:

$$M_{\text{PMBGA}} = \frac{\ln \varepsilon}{\ln\left(1-\frac{1}{2}\right)} = \frac{\ln \varepsilon}{\ln \frac{1}{2}}. \tag{5.8}$$

$$M_{\text{PMBGP}} = \frac{\ln \varepsilon}{\ln\left(1-\frac{1}{|\phi|}\right)}. \tag{5.9}$$

B) Boltzmann distribution

By comparing the above three Eq.s, the sensitivities of the diversity loss among different EDAs can be easily drawn, which have also been discussed in Chapter 3. As a result, to estimate an accurate probabilistic model with the same confidence level, PMBGP and PMBGNP need very large M, however, which is impractical in many real-world problems. To overcome this problem and to evolve practical populations, one way is to introduce some exploration methods, i.e., mutation operator [Salustowicz 1997], laplace correction [Yanai 2003] and hybrid mechanisms proposed in Chapter 3. Rather than simply adding the exploration regardless of the probability distribution, Boltzmann distribution is used to relax the sample pressure with respect to the estimated distribution of PMBGNP like Eq. (2.4). The fundamental basis of Boltzmann distribution relies on the *Hammersley-Clifford theorem* [Shakya 2006, Hammersley 1971] that any probability distribution with Markov property can be represented with Boltmann distribution, and the process of GNP program can be viewed as MDP in certain degrees [Mabu 2007b, Eguchi 2006]. Meantime, Boltzmann distribution also makes all probabilities of the probabilistic model positive values, which allows all node connections to be sampled possibly. Another advantage of Boltzmann distribution comparing with hPMBGNP in Chapter 3 is that the number of parameters to be controlled is reduced. In hPMBGNP, there are 3 additional parameters to be controlled for the maintenance of population diversity, such as the number of populations R, the crossover rate p_c and mutation rate p_m, while Boltzmann distribution only requires us to tune the temperature parameter T.

In Boltzmann distribution which 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:

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

In this chapter, the energy function of the probabilistic model is the Q values, which is $E(b(i), j) = Q(b(i), j)$.

Algorithm 6 Algorithm of RPMBGNP

```
1: $|Pop| = M$
   $t \leftarrow 0$
2: $Pop(t) \leftarrow$ generate the initial population randomly;
   $Q(t) \leftarrow$ initialize the $Q$ table;
   $Fit(t) \leftarrow$ evaluate the fitness of $Pop(t)$;
3: $Best(t) \leftarrow$ execute truncation selection to select a set of best individuals, where
   $|Best(t)| = N$, $(N \leq M)$;
4: $Q(t+1) \leftarrow$ update $Q(t)$ by Algorithm 5;
5: $P \leftarrow$ construct the probabilistic model using Eq. (5.3);
6: $Pop(t+1) \leftarrow$ generate the new population according to Algorithm 1;
   $Fit(t+1) \leftarrow$ evaluate the fitness of $Pop(t+1)$;
7: $t \leftarrow t+1$
   if the termination conditions are not met, go back to 3.
```

Therefore, the temperature parameter T can control the convergence of the probabilistic model, where the appropriate value of T can be either fixed or changed during the evolution process. In this chapter, we use an adaptive way to determine the value of T, where large values of the temperature 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 trade-off between the exploration and exploitation. A monotonically decreasing function is used in this chapter to determine the temperature:

$$T = \frac{\tau}{t+1}, \tag{5.10}$$

where,
τ: coefficient.
t: the generation number.

The value of τ is determined by the energy function $E(b(i), j)$, which can explicitly control the shape of Eq. (5.10).

5.3.3 Algorithm of RPMBGNP and comparison with PMBGNP

The pseudocode of RPMBGNP (Algorithm 6) is similar to PMBGNP, where the process of constructing the probabilistic model is based on the Sarsa Learning technique.

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 Eq. (2.5)) with the learnt Q values according to RL. For instance, in the PMBGNP, when a state-action pair, i.e., $(b(i), j)$, is observed $\bar{n}$ times among best N individuals, a weight $\bar{n}$ and its additional count of node transitions are just simply given to it. However, in the RPMBGNP, the weight of the observed pair $(b(i), j)$ is determined by the $Q(b(i), j)$ value, which is captured by RL.

By calculating the frequencies of node connections and transitions using simple MLE, PMBGNP aims to gather these information of the elite individuals for constructing its probabilistic model. However, RPMBGNP can implicitly collect these two factors simultaneously. That is, the updated Q values can convey both of the node connections and transitions information. Firstly, Sarsa Learning allows us to capture the information of node connections since a Q value will be updated iff its corresponding state-action pair is observed in the best N individuals. Secondly, RPMBGNP records the node transitions as the episode of RL (as **Definition** 7), 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 indivieduals by MLE to construct its probabilistic model, which requires time $O(N|B|)$. Meantime, RPMBGNP spends only the time on the updating of Q values, which needs time $O(NL)$. This describes that even RPMBGNP incorporates the additional technique of Sarsa Learning, it does not increase the time complexity than that of conventional MLE based PMBGNP. This shows the advantage of computation cost comparing with the conventional Bayesian network-based advanced EDAs.

5.4 Experimental analysis on Tileworld system

In order to testify the effectiveness in the problem of controlling the agents' behavior, PMBGNP and RPMBGNP are firstly applied to a benchmark testbed - Tileworld system [Pollack 1990] for comparison with traditional algorithms. And its performance on robot control is presented in the next section.

5.4.1 Tileworld system

The Tileworld system [Pollack 1990] 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. Due to its parameterization and high flexibility, Tileworld system has been widely used for the development of intelligent agents and multi-agents [Mabu 2007b, Eguchi 2006, Iba 1996, Pollack 1994].

The Tileworld system used in this chapter is designed by a 12×12 2-dimensional grid world, as an example shown in Fig. 5.1. The world consists of multiple objects, where each object occupies one cell of the world. 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 or avoid the obstacles unless reaching the world's boundaries. The agent can push a tile to its forward cell by the action of move forward when there is not obstacle in the forward cell

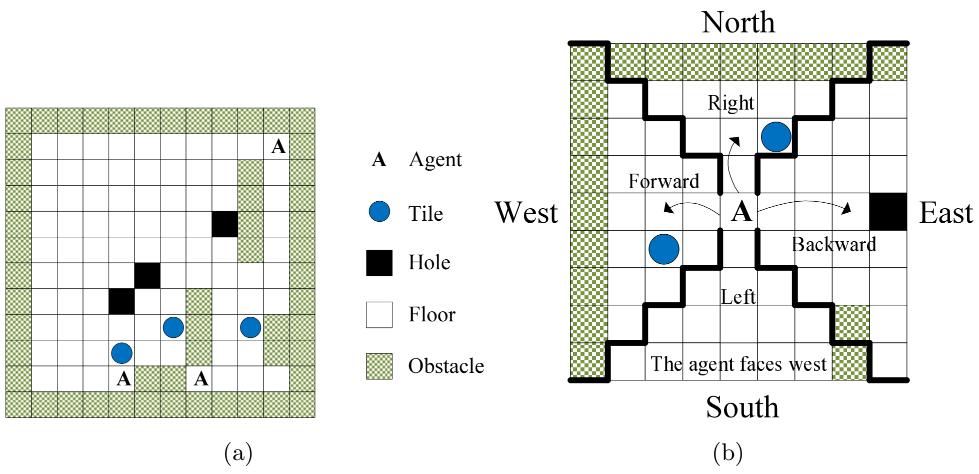


Figure 5.1: (a) Tileworld system (12×12 2-dimensional grid of cells). (b) Directions that the agent can recognize in the Tileworld system. The figure shows that when the agent faces west, the two tiles are located in its forward and right directions, while the hole is in its backward direction.

of the tile. Once a tile is pushed into a hole, this tile and hole disappear and the corresponding cell becomes a floor. The primary objective of the Tileworld system is to find the optimal strategy that could control the agents to push tiles into holes as many and fast as possible by given constrained steps.

In this chapter, the 12×12 2-dimensional environment of the Tileworld system consists of 3 agents, tiles and holes which are positioned in the Tileworld. In every step, 3 agents are controlled for movement simultaneously. The agent is designed to have 8 sensor abilities to judge the environment around it, as shown in Table 5.1. By the judgment functions $J_1 \sim J_4$, the agent can perceive the contents of its neighboring cells. The judgment functions $J_5 \sim J_8$ 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 the example shown in Fig. 5.1.(b).

Based on the returned arguments, the agent can take the following four processing actions without arguments for movements, `Move Forward`, `Turn Left`, `Turn Right` and `Stay` (Table 5.2). As a result, the programs of controlling the agents can be formulated by the combinations of these judgment sensors and processing actions.

5.4.2 Fitness function and experimental environments

To evaluate the agents' behavior, the following three factors are taken into account: in given constrained steps: The tiles should be pushed into the holes 1) as many as possible, 2) using as less steps as possible, or 3) the tiles should be pushed towards the holes as close as possible if there are remaining tiles that cannot be pushed into the holes in the given steps. As a result, the evaluation function of the solutions for

Table 5.1: Judgment functions for Tileworld system.

Function	Symbol	Description	#Args.	Content of Args.
J_1	JF	`Judge Forward`	5	1: Floor 2: Obstacle 3: Tile 4: Hole 5: Agent
J_2	JB	`Judge Backward`		
J_3	JL	`Judge Left`		
J_4	JR	`Judge Right`		
J_5	DT	Judge the `Direction` of the nearest `Tile` from the agent	5	1: Forward 2: Backward 3: Left 4: Right 5: Cannot find
J_6	DH	Judge the `Direction` of the nearest `Hole` from the agent		
J_7	DHT	Judge the `Direction` of the nearest `Hole` from the nearest `Tile`		
J_8	DST	Judge the `Direction` of the `Second` nearest `Tile` from the agent		

Table 5.2: Processing functions for Tileworld system.

Function	Symbol	Description	#Args.
P_1	MF	`Move Forward`	0
P_2	TL	`Turn Left`	
P_3	TR	`Turn Right`	
P_4	ST	`Stay`	

the Tileworld system is formulated as follows

$$f = c_t \times \underbrace{DT}_{1)} + c_s \times \underbrace{\Delta ST}_{2)} + c_d \times \underbrace{\left[\sum_{t \in Tile} \Big(D(t) - d(t) \Big) \right]}_{3)}, \tag{5.11}$$

and ΔST denotes the remaining steps calculated by

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

where,
DT: the number of tiles that have been pushed into the holes.
ST: user-defined constraint step.
S_{used}: the 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 Eq. (5.11), the three marked terms correspond to the three factors 1), 2) and 3) discussed above. In this chapter, ST is set at 60. To balance the effects of factor 1), 2) and 3), the parameters are set at $c_t = 100$, $c_s = 3$ and $c_d = 20$.

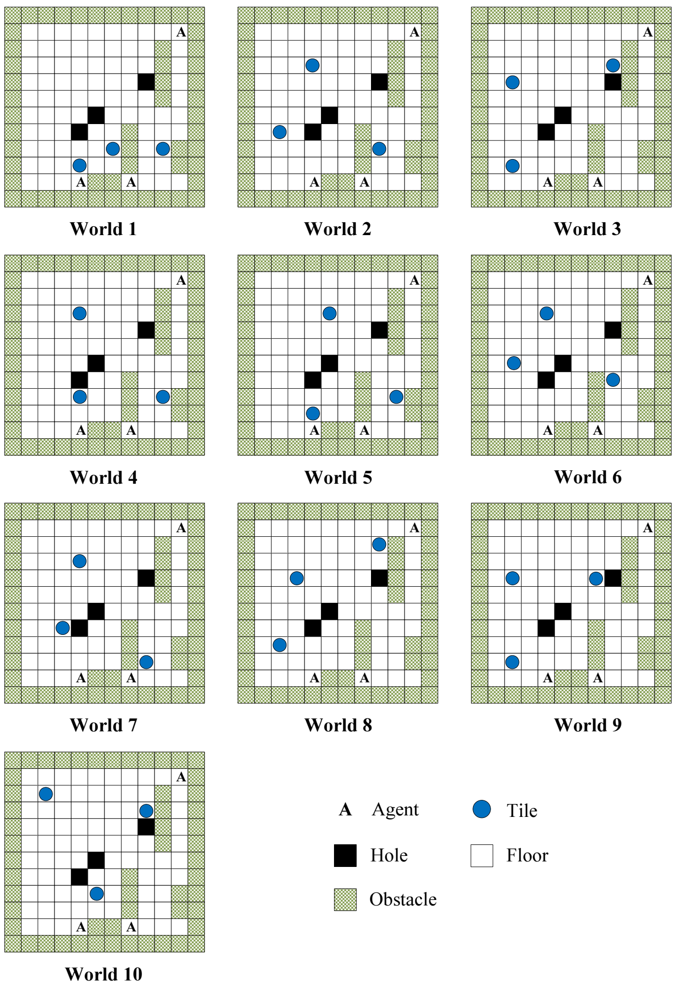


Figure 5.2: Ten worlds of Tileworld system used in this chapter.

Ten worlds of the Tileworld system are used to perform the experimental environments in this chapter, as shown in Fig. 5.2. In these cases, the numbers of agents, tiles and holes are set at 3. The positions of agents, obstacles and holes are the same, while the positions of tiles vary case by case in order to obtain a general strategy that could control the agents' behavior for handling 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 ten environments. As a result, the fitness function of the Tileworld system used in this chapter is as follows

$$Fitness = \sum_{w=1}^{10} f(w), \tag{5.12}$$

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

Table 5.3: Parameter settings for Tileworld system and Khepera robot control. ([] presents the settings for Khepera robot control.)

	GNP	GP	PIPE	EDP	Sarsa	PMBGNP	RPMBGNP
Population size M	300 [500]	300 [500]	200 [300]	1800 [2000]	—	1800 [2000]	1800 [2000]
– elite ind.	1	1	1	100	—	100	100
– crossover ind.	120 [200]	120 [200]	—	—	—	—	—
– mutation ind.	179 [299]	179 [299]	—	—	—	—	—
– promising ind. N	—	—	1	900 [1000]	—	900 [1000]	900 [1000]
Program size $\|N_{\text{node}}\|$	60	156-781 [127]	156-781 [127]	156-781 [127]	—	60	60
Maximum Tree Depth (D_M)	—	3-4 [6]	3-4 [6]	3-4 [6]	—	—	—
Tournament size	2 [5]	2 [5]	—	—	—	—	—
Crossover rate	0.1	0.9	—	—	—	—	—
Mutation rate	0.01 [0.02]	0.1 [0.2]	—	—	—	—	—
Other parameters							
— PIPE	(learning rate) $= 0.02$ [0.03] (fitness constant) $= 1$ [0.1] (elitist update probability) $= 0$						
— EDP	(learning rate) $= 0.1$ [0.05]						
— Sarsa	(learning rate) $\alpha = 0.2$ [0.1] (discount factor) $\gamma = 0.9$ (ε-greedy policy) $\varepsilon = 0.01, 0.1, 0.5$						
— PMBGNP	$\eta = 0.02$ $[0.2, 0.1, 0.07, 0.05, 0.04]$ $\tau = 200$ [100]						
— RPMBGNP	(learning rate) $\alpha = 0.2$ [0.1] (discount factor) $\gamma = 0.9$						
Terminal condition	300000 fitness evaluations						

5.4.3 Compared algorithms and experimental settings

Note that although there exist multiple agents in the Tileworld system, i.e., 3 in this chapter, we use the *homogeneous breeding* [Iba 1996] to control the agents' behavior. That is, all agents use the same strategy to evaluate the performance of the Tileworld system. 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 the comparison in the Tileworld system:

Genetic Network Programming (GNP) [Hirasawa 2001, Mabu 2007b]: This is the standard implementation of original GNP. The directed graph structure described in chapter 2 is used to represent GNP programs. GNP consists of 8 judgment functions and 4 processing functions in its LIBRARY. The number of each kind of judgment and processing nodes is set at 5. As a result, there are total $|N_J| = 8 \times 5 = 40$ judgment nodes and $|N_P| = 4 \times 5 = 20$ processing nodes. For the case of judgment nodes, each judgment node has 5 branches since each judgment function has 5 arguments. The number of branches for each processing node is set at 1 since each processing function has no arguments. Therefore, the directed graph structure $G = (N_{\text{node}}, B)$ of GNP consists of $|N_{\text{node}}| = 40 + 20 = 60$ nodes and $|B| = 40 \times 5 + 20 \times 1 = 220$ branches. The time delays of GNP are set as chapter 2. That is one time unit for judgment nodes, five time units for processing nodes and zero for node transitions. One step of the agents ends when five or more time units are used. Each GNP individual ends when the predefined steps ST, i.e., 60, are reached.

The evolution of GNP is achieved by standard genetic operators, including uniform crossover and mutation, which are similar to the other EAs (Details of the uniform crossover and mutation in GNP can be referred to [Hirasawa 2001, Mabu 2007b]). Elite selection (with elite size 1) is used to directly reproduce the current best GNP individual into the next generation. Tournament selection (with size 2) is adopted to select GNP individuals for crossover and mutation.

Genetic Programming (GP) [Koza 1992]: The function set of GP is composed of 8 judgment functions, and its terminal set is defined by the 4 processing functions. Since all the functions have 5 arguments, each function node in GP's tree structure consists of 5 arities. Terminals have no arguments and can be only assigned to the leaf nodes. Therefore, the GP programs are encoded in *complete 5-ary trees*. Maximum tree depth D_M is defined to determine the size of GP program. Since the GNP program consists of 60 nodes,D_M is set to 3 and 4 to create its program with similar size for comparison[1]. In the case of $D_M = 3$, GP tree consists of 156 nodes, while 781 nodes are existed when $D_M = 4$. `FULL` method is used to initialize the GP individuals[2].

To evolve GP programs, one-point crossover and point mutation [Poli 2008a] are used. In one-point crossover, the crossover point is selected with biased, while function nodes are selected as crossover point 90% of the time and terminal nodes are selected 10% of the time.

Probabilistic Incremental Program Evolution (PIPE) [Salustowicz 1997]: It is a univariate PMBGP, which uses *Probabilistic Prototype Tree* (PPT) to represent its probabilistic model. Since the GP programs are encoded in complete 5-ary trees and D_M is defined, we can easily create the PPT of PIPE which is a complete 5-ary tree with depth D_M according to [Salustowicz 1997]. The function nodes of PPT can be only picked from the function set, while the terminal nodes can only select instructions from the terminals. As a result, the PPT of PIPE consists of $31 \times 8 + 125 \times 4 = 748$ probabilities for $D_M = 3$, and $156 \times 8 + 625 \times 4 = 3748$ probabilities for $D_M = 4$. It can be easily found that the probabilities in the function and terminal nodes of PPT are initialized to 1/8 and 1/4, respectively.

PPT is used to replace crossover and mutation for the generation of new populations in PIPE. The construction of PPT is similar to that of PBIL, where an incremental learning method called *Generation-Based Learning* (GBL) is designed to update the probabilities of PPT towards the best GP program found so far. Mutation of PPT reported in [Salustowicz 1997] is not used in the experiments in order to present the pure performance of EDA. Since the GP tree for the Tileworld system in this chapter is complete 5-ary trees, growing and pruning of PPT are not necessary, and many parameters of PIPE can be omitted. For the remaining parameters, the learning rate is set at 0.02, and the fitness constant and elitist update probability are set at 1 and 0, respectively.

Estimation of Distribution Programming (EDP) [Yanai 2003]: EDP is an

[1]Larger tree depth can be also set, however, which generally increases the search space, decreasing the evolution efficiency for finding optimal solutions. For example, $D_M = 5$ and 6 are also testified in this simulation, however, which have shown worse results than that of $D_M = 4$.

[2]Although `GROW` and `Ramped half-and-half` can create trees of more varied sizes and shapes, we tend to fully use the trees with the maximum tree depth. This allows the agents to obtain more precise environmental information surrounding them by taking more function nodes. According to our experiments, `FULL` method can obtain the best performance.

extension of PIPE whose probabilistic model is constructed by conditional probabilities between parent node and child node. It can model pairwise interactions of GP tree. For the Tileworld system, its probabilistic model consists of 8 marginal probabilities for the root node and $(5+5^2)\times 8^2+5^3\times 8\times 4 = 5920$ conditional probabilities for the other nodes when $D_M = 3$. In the case of $D_M = 4$, EDP should maintain 8 marginal probabilities for the root node and $(5+5^2+5^3)\times 8^2+5^4\times 8\times 4 = 29920$ conditional probabilities for the other nodes. The initial probability values for function nodes are 1/8, and that for terminal nodes are 1/4.

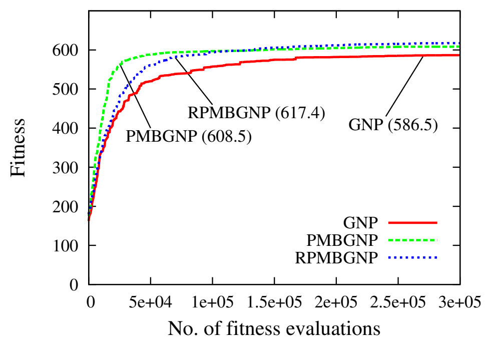
The population size for EDP is set at 1800, and top 50% individuals are selected for the probabilistic modeling. The learning rate is set at 0.1 to perform the best results.

Sarsa Learning (Sarsa) [Sutton 1998]: This is a standard implementation of Sarsa, which is selected as a state-of-the-art RL algorithm for the comparison with this work. The states are defined by the observations that the agents can possibly see in its four adjacent cells of the grid world[3]. This means that there are $5^4 = 625$ possible observations. The actions are the processing functions of the Tileworld system. Therefore, there are total 625 states, 4 actions and $625 \times 4 = 2500$ state-action pairs for Sarsa. Sarsa updates its Q values by its updating function. When a tile has been pushed into the hole, a reward ($r = 1$) is assigned for the updating of Q values. The agents are controlled to move until all tiles are pushed into the holes or the maximum steps ST is reached. The learning rate α and discount factor γ are set at 0.2 and 0.9, respectively, and ε-greedy policy (with $\varepsilon = 0.01$, 0.1 and 0.5) is used for the selection of actions to study the balance of the exploitation-exploration.

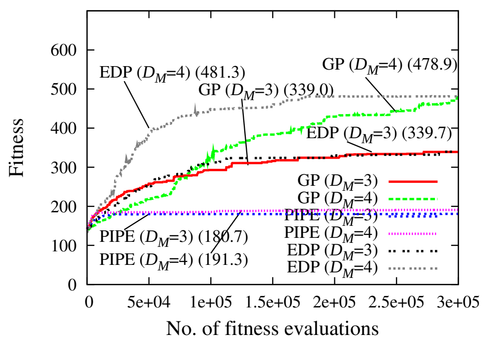
PMBGNP: PMBGNP uses the same graph structure $G = (N_{\text{node}}, B)$ of GNP to represent its individual. However, a probabilistic model is used to replace crossover and mutation of GNP for the generation of new populations. The probabilistic model of PMBGNP is constructed by MLE method described in this chapter. However, different from the ones of standard MLE-based probabilistic modeling used in chapter 2, 3, and 4, PMBGNP used in this chapter also applies Boltzmann distribution to relax the sample pressure to avoid the premature convergence. The size of PMBGNP's probabilistic model can be calculated by $|P| = 220 \times (60 - 1) = 12980$. The population size is set at 1800 to model an accurate probability distribution. For the parameters of the probabilistic modeling in PMBGNP, η is set at 0.02 to balance the effects of the connection information and transition information. τ is set at 200 to control the shape of Boltzmann distribution for the trade-off between the exploration and exploitation.

RPMBGNP: This is the extension of PMBGNP by combining the technique of RL, i.e., Sarsa in this chapter. The settings of graph structure G, population size and parameter τ are the same as that of PMBGNP. As described in chapter 4, the states and actions of Sarsa in RPMBGNP are defined by the branches and nodes of G, since we aim to learn the graph structure. As a result, there are total 220 states,

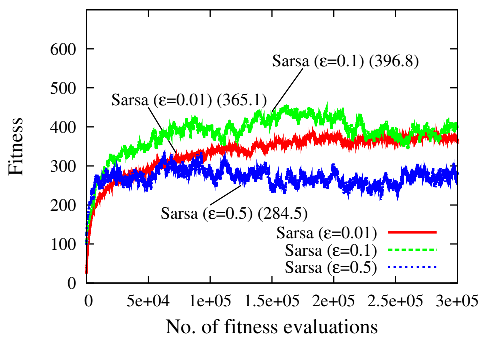
[3]Although we can use the combinations of judgment functions to represent more realistic states that the agents can observe in the grid world, we do not consider this way, since it will create extremely large state space (with size of $5^8 = 390625$) which is almost impossible to be learnt.



(a) GNP and its variants



(b) GP and its variants



(c) Sarsa

Figure 5.3: Fitness curves of a single Tileworld system (Simulation I).

60 actions and $220 \times 60 = 13200$ state-action pairs in RPMBGNP. The reward is defined similarly as standard Sarsa, where the reward ($r = 1$) will be assigned when a tile has been pushed into the hole by a processing node. The learning rate α is set at 0.2, and discount rate γ is set at 0.9.

All the settings are listed in Table 5.3, which are determined by hand-tuning to perform the best results of each algorithm.

5.4.4 Experimental results and analysis

A) 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. 5.2 is used to perform the experiment. In this case, the fitness is calculated 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

$$\begin{aligned} f_{max}(1) &= 100 \times 3 + 3 \times (60 - 15) + 20 \times \\ &\quad \Big[(2-0) + (3-0) + (5-0)\Big] \\ &= 635. \end{aligned} \tag{5.13}$$

Table 5.4: Results of a single Tileworld system (Simulation I).

	Fitness (std. dev.)	suc%[1]	Rank	t-test 1[2]	RFEs (std. dev.)	t-test 2
GNP	586.5 ± 69.8	93.3	3	2.06e-01 **4.04e-02**	56160 ± 73983	**3.69e-03** **4.86e-02**
GP ($D_M = 3$)	339.0 ± 121.1	13.3	8	**6.34e-12** **8.53e-13**	275790 ± 63075	**7.18e-20** **7.84e-18**
GP ($D_M = 4$)	478.9 ± 151.6	40.0	5	**4.03e-05** **2.07e-05**	238090 ± 90072	**5.89e-14** **9.64e-13**
PIPE ($D_M = 3$)	180.7 ± 3.7	0.0	10	**7.95e-28** **2.38e-40**	300000 ± 0	**1.34e-59** **1.12e-35**
PIPE ($D_M = 4$)	191.3 ± 18.0	0.0	9	**1.34e-26** **2.96e-36**	300000 ± 0	**1.34e-59** **1.12e-35**
EDP ($D_M = 3$)	339.7 ± 128.3	16.7	7	**8.24e-12** **1.98e-12**	269560 ± 76582	**2.57e-17** **5.24e-16**
EDP ($D_M = 4$)	481.3 ± 99.9	76.7	4	**2.58e-06** **3.39e-08**	172720 ± 101916	**2.74e-09** **4.73e-08**
Sarsa	396.8 ± 114.3	26.7	6	**5.17e-10** **1.85e-11**	268761 ± 66519	**5.52e-19** **3.09e-18**
PMBGNP	608.5 ± 54.4	100.0	2	— 4.32e-01	12600 ± 2865	— **4.28e-05**
RPMBGNP	617.4 ± 20.2	100.0	1	— —	28320 ± 18137	— —

t-test 1 analyzes the statistical difference of fitness results between PMBGNP/RPMBGNP and the other algorithms, while t-test 2 is studied by Required Fitness Evaluations (RFEs).

[1] suc%: percentage that all 3 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.

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 300000 is used. All the experimental results are the average of 30 independent trials. These settings are also used in the rest experiments of this chapter.

Fitness results: The fitness curves of the compared algorithms are shown in Fig. 5.3, and the detailed fitness results are described in Table 5.4. The results show that:

Graph structures of GNP achieve better performance than that of tree structures of GP. It is found from Fig. 5.3 that even if GNP has a smaller number of program size (60 for GNP, and 156/781 for GP with depth 3/4), it can achieve much better fitness results than that of GP. On the other hand, it is natural that GP programs can have higher expression ability as D_M increases, however, GP programs will also increase the computation time and memory exponentially. 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 univariate model, which cannot cover the interactions between judgment and processing functions of the Tileworld system.

Sarsa achieves worse fitness results than the other algorithms except GP/EDP with $D_M = 3$ and PIPE. This is because the most important lack of RL is that the optimal strategy is hardly learnt when the state-action space (Q table size) becomes large. In this problem, the Q table size is 2500, where 300000 episodes cannot find the optimal ones. Meantime, the value of ε should be determined appropriately to balance the exploitation and exploration. The results show that 0.1 is the best setting.

From the perspective of the proposed algorithms, both of PMBGNP and RPMBGNP can achieve better results than the others. By incorporating RL, RPMBGNP can find slightly better result 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 are 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 learnt 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, it is shown 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 the 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 5.4, t-test (two tailed, paired) results show the statistically significant differences between the proposed algorithms and the other algorithms. Meantime, although RPMBGNP can obtain slightly better result than PMBGNP, it is found that there is no statistical difference between them. This is because both of PMBGNP and RPMBGNP can successfully push all the tiles into the holes using as less steps as possible for the simple problem of Simulation I.

Required fitness evaluations (RFEs) and Reliability: The required fitness evaluations (RFEs) is further counted to testify the search speed. The RFEs is calculated in the following way: for the successful trials that the agents can push all 3 tiles into the holes within ST (=60) steps, the exact RFEs is used; for the failed trials that the agents cannot push all 3 tiles into the holes even after ST steps, the maximum fitness evaluations, i.e., 300000, is counted. Afterwards, the average value of 30 independent trials is calculated to form the final RFEs.

The results of RFEs of Simulation I is shown in Table 5.4. The results show that on average PMBGNP can successfully push all the tiles into the holes with the

Table 5.5: 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 ($D_M = 3$)	3042.5 ± 524.5	15.9 ± 3.0	52.9	8	**4.36e-11** **8.39e-14**
GP ($D_M = 4$)	3356.3 ± 679.2	17.6 ± 3.5	58.7	6	**2.44e-06** **2.04e-12**
PIPE ($D_M = 3$)	1968.7 ± 303.3	9.9 ± 1.9	33.0	10	**1.56e-16** **2.28e-21**
PIPE ($D_M = 4$)	2267.4 ± 338.0	11.5 ± 2.4	38.4	9	**3.05e-15** **1.12e-21**
EDP ($D_M = 3$)	3144.8 ± 463.5	16.1 ± 2.5	53.6	7	**9.78e-09** **4.06e-14**
EDP ($D_M = 4$)	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.%: percentage that the tiles has been pushed into the holes within ST steps in 30 independent trials. (For each trial, perc.%=$100 \times \sum_{w=1}^{10} DT/30$.)

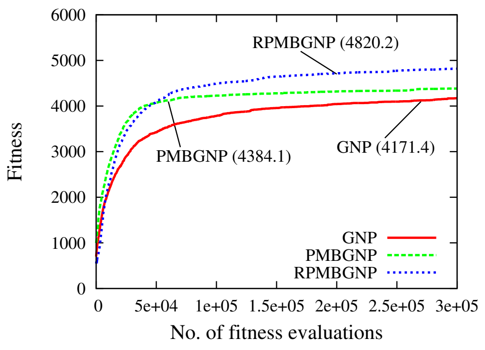
fastest speed, while RPMBGNP stands in the second rank which requires slightly larger FEs. The t-test 2 results describe the statistically significant difference between PMBGNP/RPMBGNP and the other algorithms from the perspective of RFEs. Overall, from the perspective of reliability, the two proposed algorithms can succeed in all 30 independent trials, while GNP fails in 2 trials. EDP with $D_M = 4$ achieves the best result among all GP variants, while PIPE fails in all trials.

B) Simulation II: Case study in ten worlds

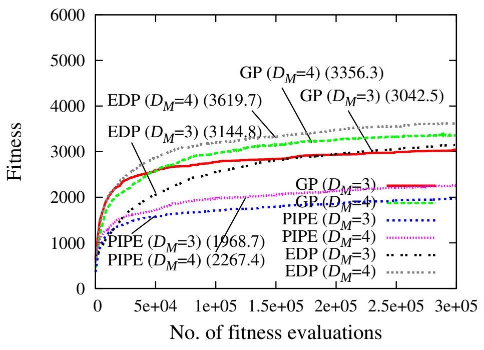
In this experiment, ten worlds shown in Fig. 5.2 are used to perform the comparison. The fitness is calculated by Eq. (5.12), which are the sum of the evaluation results of ten worlds. This experiment is much more complex than that of Simulation I, since it is needed to find a strategy that can handle all the ten worlds well. Consequently, 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. 5.4, and the detailed fitness results are described in Table 5.5. The results show similar conclusions to Simulation I, where the proposed algorithms, i.e., PMBGNP and RPMBGNP, can obtain the highest fitness after 300000 fitness evaluations.

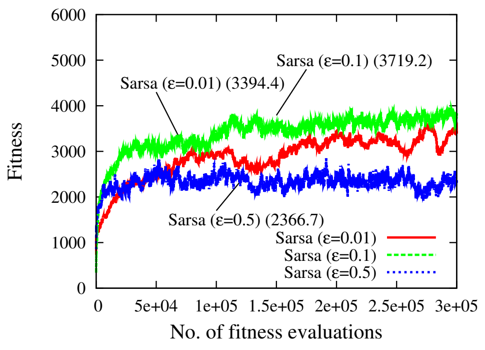
From the fitness curves of Fig. 5.4.(a) it is explicitly found that the probabilistic



(a) GNP and its variants



(b) GP and its variants



(c) Sarsa

Figure 5.4: Fitness curves of ten Tileworld systems (Simulation II).

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 that Simulation II uses ten worlds to perform the results, where the fitness landscape becomes more smooth. The results report that EDP with $D_M = 4$ can obtain the best fitness values than the other GP variants, where PIPE performs the worst result. The t-test results show the significant difference between the proposed algorithms and the others.

From the perspective of dropped tiles (DT), the results report that the proposed algorithms can push as more 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 the Simulation II reports the results in 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 that 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.

5.4.5 Generalization ability

To testify the generalization ability of each algorithm, we applied the 30 best solutions found by the independent trials of Simulation II to the new environments. Two experiments were carried out, where one randomly changes the tile positions of the ten worlds in Fig. 5.2, and another randomly changes both of the tile and hole positions. For each best solution, we applied it to the ten worlds and ran 1000 times to calculate the average results. The results of each algorithm are reported in Table 5.6.

As shown in Fig. 5.2, the best solutions are trained and obtained by ten worlds with different tile positions. As a result, 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, the proposed algorithms PMBGNP and RPMBGNP show higher generalization ability than the other algorithms in both testing experiments.

5.4.6 Additional experimental results on Tileworld system

This section studied the *computation time* of each algorithm to confirm the efficiency of the proposed algorithms, and the *effects of parameters* needed to be configured in the proposed algorithms, i.e., α and γ in RPMBGNP.

A) Computation time

Table 5.6: Fitness results (std. dev.) in new environments of 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**

Table 5.7: Computation time for 300000 fitness evaluations in ten Tileworlds. In the variants of GP, $D_M = 4$ is used since it performs better results than that of $D_M = 3$.

	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**

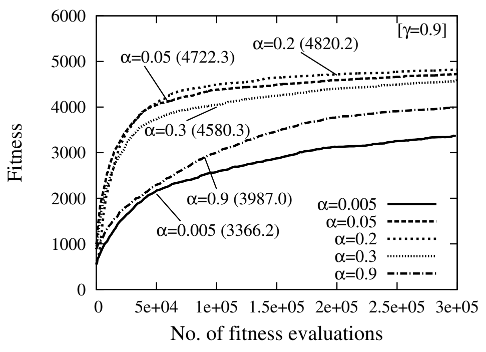
Table 5.7 shows the computation time of each algorithm in Simulation II of Tileworld system. All experiments are carried out on a PC with Intel Core i5 running at 2.80 GHz with 4 GB of RAM. The used complier is Visual Studio 2010 under the OS of Windows 7. For each algorithm, the experiments end when 300000 fitness evaluations are achieved.

Sarsa requires the smallest computation time for problem solving, since the updating of the Q values only needs linear time cost with respect to the maximum steps, i.e., ST in this chapter. On the other hand, GNP is the faster than GP, since GP (with $D_M = 4$) has much more nodes which takes more time than that of GNP with compact program size. EDA based algorithms need more time than conventional EAs since the estimation of the probability distribution generally requires additional time cost. Meantime, the computation time is proportional to the accuracy of the probabilistic model. This means that learning a probabilistic model that can capture more complex variable interactions requires more computation time. As a result, EDP requires more time than PIPE, since it needs to maintain more probabilities in every generation. As for the proposed algorithms, the results show that even integrating RL into the probabilistic modeling, RPMBGNP only requires similar computation time to PMBGNP. This could be explained by the discussion presented in section 5.3.3, where the time complexities of PMBGNP and RPMBGNP are in the same level.

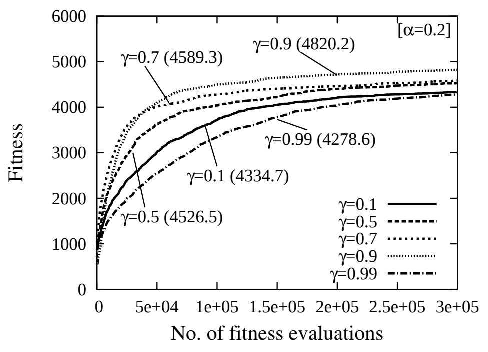
B) Effects of parameters

This part discusses the effect of parameters in the proposed RPMBGNP. All the results are carried out based on Simulation II of the Tileworld system.

As discussed in the previous section, η is removed from the probabilistic modeling



(a) Fitness results with different α



(b) Fitness results with different γ

Figure 5.5: Effect of α and γ of the Tileworld system in Simulation II.

of RPMBGNP, where the Q values play the role of automatically counting the information of both node connections and transitions. As a result, the parameter of RL, i.e., learning rate α and discount factor γ should be configured in RPMBGNP.

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 of Q values. The appropriate α should be set to update Q values gradually. As shown in Fig. 5.5.(a), α=0.2 can achieve the highest fitness values comparing with the other values in Simulation II.

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 current reward obtained by the 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 to 1, the Q values may diverge. Fig. 5.5.(b) reports the fitness curves with different γ, where γ=0.9 shows the best result.

5.5 Experimental analysis on robot control

Besides solving the benchmark problem of the Tileworld system, the proposed algorithms is further applied to the Wall-Following problem of Khepera robot control.

The experimental environment is defined by a map used in Chapter 3 and 4. The settings of node functions and Wall-Following problem are remaining the same as the ones used in the previous chapters. In this chapter, the predefined steps is $ST \in \{100, 200, 300, 400, 500\}$ to study the performance of the proposed algorithm under different problem sizes.

5.5.1 Compared algorithms and experimental settings

The same algorithms shown in the Tileworld system will be used to present a comparative study of the proposed algorithms and the classical ones. The experimental settings are listed in Table 5.3, which have been testified to perform the best results.

The judgment and processing functions of GNP is designed as the ones presented in the previous chapters, which consists of 8 judgment functions and 10 processing functions. The number of each kind of judgment nodes is set at 5, and that of processing node is equal to 2. As a result, the directed graph structure $G = (N_{\text{node}}, B)$ of GNP has $|N_{\text{node}}| = 40 + 20 = 60$ nodes and $|B| = 40 \times 2 + 20 \times 1 = 100$ branches. The function and terminal set of GP consists of 8 judgment functions and 10 actions. The GP programs are represented by complete 2-ary trees. D_M is set at 6 (with 127 nodes) for relative fair comparison with GNP. `FULL` method is used for the initialization of GP.

The setting of tree structure in PIPE is the same as that of GP. The PPT is a complete 2-ary tree with depth D_M. The size of PPT is 1144. The parameters of PIPE, including learning rate, fitness constant and elitist update probability, are set at 0.03, 0.1 and 0, respectively. In the probabilistic model of EDP, there exists 8 marginal probabilities and 9088 conditional probabilities. The learning rate is set at 0.05.

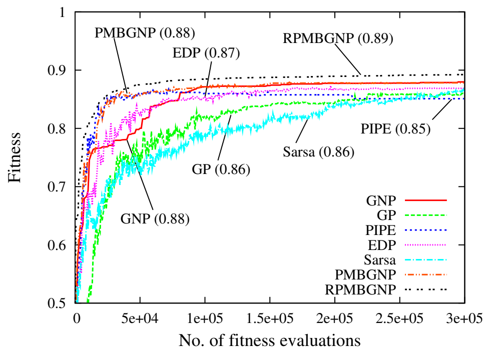
The states of Sarsa for this problem is defined by the combinations of judgment functions. Since each judgment function has 2 arguments, the total number of states for Sarsa is $2^8 = 256$. The actions are defined by the settings of different speeds for left and right motors, which are $5 \times 5 = 25$. Therefore, there are total 6400 state-action pairs. In every step, the reward is calculated by Eq. (3.17) for the updating of Q values. The task ends until ST is reached. α, γ and ε are set at 0.1, 0.9 and 0.1, respectively.

The setting of graph structure G in PMBGNP remains the same as GNP. The size of its probabilistic model is $|P| = 100 \times (60 - 1) = 5900$. In the probabilistic modeling of PMBGNP, the value of η varies w.r.t. ST in order to balance the effects of the connection and transition information. For the five settings of ST in this chapter, η is set at 0.2, 0.1, 0.07, 0.05 and 0.04, respectively. τ is set at 100. In RPMBGNP, the sizes of states and actions for Sarsa are 100 and 60, respectively. The updating of Q values is done by Sarsa, where the reward of Eq. (3.17) is assigned in every execution of processing nodes. α and γ is set at 0.1 and 0.9.

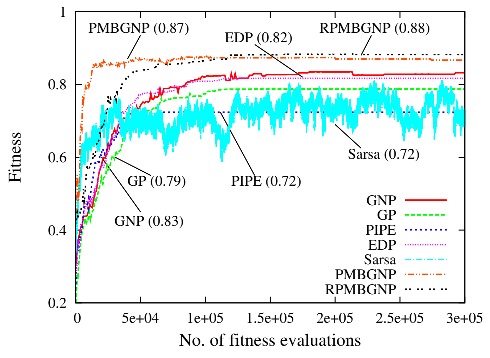
5.5.2 Experimental results and analysis

The environment of Fig. 3.2 used in the previous chapters is used to conduct the experiments. All the results are the averaged of 30 independent trials.

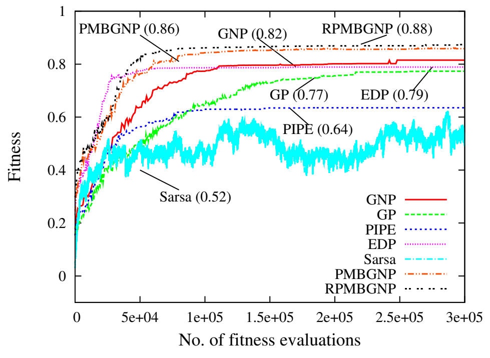
Fitness results: The fitness curves for the five studied Wall-Following problems are presented in Fig. 5.6, and the detailed fitness results are described in Table 5.8. The reporting results draw similar conclusions to that of Tileworld system. The proposed algorithm PMBGNP outperforms the classical algorithms, and by



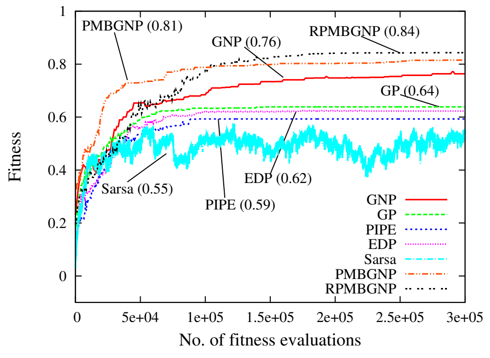
(a) $ST = 100$



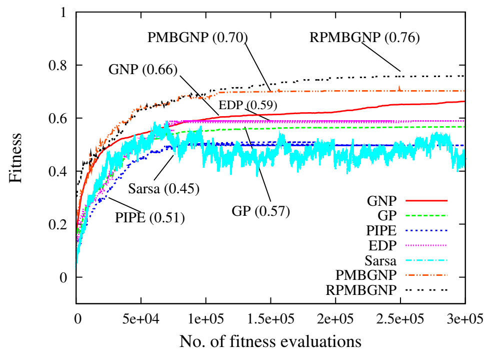
(b) $ST = 200$



(c) $ST = 300$



(d) $ST = 400$



(e) $ST = 500$

Figure 5.6: Fitness curves in the Wall-Following problems with different settings of ST.

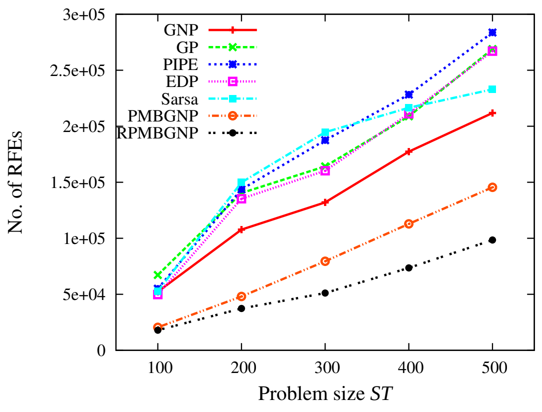


Figure 5.7: The required fitness evaluations for Wall-Following problems.

incorporating RL to propose RPMBGNP, the performance can be further improved. Considering the five Wall-Following problems as a whole, the results clarify the scalability of the proposed algorithms that they can obtain higher fitness values than the others. For simple problems, i.e., ST=100, all the algorithm can solve the problem successfully. However, with the increase of the problem size, the proposed algorithms can significantly outperform the others. The results of t-test 1 report the statistically significant difference between PMBGNP/RPMBGNP and the compared algorithms.

RFEs and reliability: The results of RFEs for the five Wall-Following problems are shown in Fig. 5.7 in order to study the search speed. The results of detailed RFEs are shown in Table 5.8. The t-test 2 is reported to present the statistical analysis of different algorithms with respect to RFEs. According to the results, most algorithms can solve the simple Wall-Following problem, i.e., $ST = 100$, successfully. However, the number of RFEs are different. Overall, the results indicate that the proposed algorithms need the smallest RFEs to solve the tasks, while the others need larger ones. Meantime, with the increase of the problem size, the gaps among different algorithms become more significant. In the large problem size, Sarsa and PIPE almost cannot find the acceptable trajectories under the constrained FEs, where GNP, GP and EDP can only find a small number of successful trials. t-test results show that there are statistically significant differences between the proposed algorithms and the others. The results of suc% in Table 5.8 indicate the reliability of each algorithm in this problem. It shows that PMBGNP has much higher probability than the classical algorithms to find the acceptable trajectory to solve the Wall-Following problems. On the other hand, by integrating RL, RPMBGNP can almost achieve the 100% probability to solve all the five problems.

Table 5.8: The fitness results and required fitness evaluations for Wall-Following problems.

Problem size ST		GNP	GP	PIPE	EDP	Sarsa	PMBGNP	RPMBGNP
100	Fitness	0.88±0.02	0.86±0.05	0.85±0.06	0.87±0.05	0.87±0.04	0.88±0.02	0.89±0.01
	t-test 1	7.40e-01	1.31e-01	**1.02e-02**	3.96e-01	**7.39e-02**	—	**5.94e-03**
		2.71e-05	**2.31e-03**	**3.78e-04**	**2.91e-02**	**4.82e-04**	—	—
	RFEs	52300±47379	67200±56564	54967±82806	49800±68639	52800±51392	20467±27883	18000±14325
	suc%	100	100	90	93.3	100	100	100
	t-test 2	**6.28e-03**	**3.61e-04**	**4.95e-02**	**4.64e-02**	**7.40e-03**	—	6.81e-01
		2.64e-04	**9.32e-05**	**2.78e-02**	**1.99e-02**	**2.88e-03**	—	—
200	Fitness	0.83±0.11	0.79±0.15	0.72±0.10	0.82±0.11	0.72±0.11	0.87±0.05	0.88±0.06
	t-test 1	1.29e-01	**2.25e-03**	**1.33e-07**	**4.94e-02**	**2.02e-04**	—	2.64e-01
		3.91e-02	**2.68e-03**	**1.15e-07**	**1.04e-02**	**1.78e-04**	—	—
	RFEs	107733±76205	140427±97119	143713±109848	135347±97818	149850±97569	48033±20993	37377±14584
	suc%	90	76.7	56.7	83.3	60	100	100
	t-test 2	**5.63e-04**	**4.88e-05**	**7.06e-05**	**9.412e-05**	**1.14e-05**	—	**2.87e-02**
		4.82e-05	**7.79e-06**	**2.67e-05**	**5.62e-06**	**1.07e-06**	—	—
300	Fitness	0.82±0.09	0.77±0.15	0.64±0.16	0.79±0.12	0.52±0.21	0.86±0.05	0.88±0.03
	t-test 1	**3.99e-02**	**1.20e-02**	**5.11e-08**	**1.05e-02**	**3.59e-09**	—	1.21e-01
		2.28e-03	**9.29e-03**	**5.02e-09**	**1.19e-03**	**5.38e-10**	—	—
	RFEs	132000±101807	164180±82590	187520±106868	160340±116551	194707±99035	79500±64111	51167±32248
	suc%	76.7	63.3	40	70	36.7	93.3	100
	t-test 2	**1.50e-02**	**5.79e-06**	**1.38e-04**	**1.49e-03**	**4.40e-06**	—	**2.88e-02**
		4.56e-04	**8.00e-09**	**1.15e-07**	**5.42e-05**	**1.70e-08**	—	—
400	Fitness	0.76±0.08	0.64±0.08	0.59±0.17	0.62±0.14	0.55±0.19	0.81±0.09	0.84±0.10
	t-test 1	**5.17e-02**	**5.41e-08**	**5.80e-07**	**9.92e-08**	**2.33e-07**	—	2.04e-01
		9.02e-04	**9.43e-10**	**5.85e-08**	**2.65e-08**	**6.11e-08**	—	—
	RFEs	177300±87890	208893±121929	228220±112741	211093±102287	216529±106187	112867±95416	73593±21606
	suc%	70	36.7	30	36.7	23.3	83.3	100
	t-test 2	**8.71e-03**	**7.56e-03**	**2.53e-04**	**8.20e-04**	**2.48e-04**	—	**3.57e-02**
		5.7e-07	**1.00e-06**	**3.68e-08**	**4.71e-08**	**6.78e-08**	—	—
500	Fitness	0.66±0.15	0.57±0.10	0.51±0.06	0.59±0.08	0.45±0.22	0.70±0.08	0.76±0.08
	t-test 1	1.93e-01	**1.01e-05**	**1.33e-09**	**4.00e-05**	**2.26e-06**	—	**2.57e-02**
		6.01e-03	**2.02e-08**	**6.67e-14**	**9.79e-10**	**3.48e-08**	—	—
	RFEs	211833±111542	268827±70595	283700±62032	267040±85622	232970±100870	145467±87187	98500±75761
	suc%	43.3	20	6.7	13.3	16.7	80	93.3
	t-test 2	**1.55e-02**	**1.30e-06**	**7.01e-08**	**5.26e-06**	**1.71e-03**	—	**1.42e-02**
		2.51e-05	**2.20e-08**	**4.12e-10**	**5.11e-08**	**1.00e-05**	—	—

the upper value of t-test 1/2 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.

5.6 Summary

RPMBGNP has been proposed by integrating PMBGNP and RL in this chapter. PMBGNP factorizes the individuals to the sequences of state-action pairs/node connections, and uses RL, i.e., Sarsa Learning, to learn knowledge/experience during the individual executions. The collected knowledge formulated by Q values can correctly measure the quality of node connections. As a result, RPMGNP incorporates the learnt Q values into its probabilistic modeling, which has shown better performance than the MLE-based PMBGNP.

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 this 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. Meantime, the computation cost is remaining in the same level as PMBGNP.

Until now, the proposed variants of PMBGNP are applied to solve the discrete optimization problems. In the next chapter, PMBGNP will be extended to solve the continuous optimization optimization problems.

Chapter 6

Continuous PMBGNP

Contents

6.1 Introduction

Standard PMBGNP and its variants proposed in the previous chapters are mainly designed for discrete optimization problems. Therefore, it cannot deal with (or directly handle) continuous variables. To solve this problem, the discretization of continuous variables should be employed, which will cause the loss of solution precision. In this chapter, a new PMBGNP algorithm is proposed to directly handle the continuous variables.

On the other hand, much progress has been made in the development of EDA for continuous domains [Sebag 1998, Larrañaga 1999, Bosman 2006]. This starts with the early attempt of continuous Population-based incremental learning (PBILc) [Sebag 1998] and continuous Univariate marginal distribution algorithm (UMDAc) [Larrañaga 1999], in which the probability density function is modeled by unidimensional Gaussian distribution and variable independencies assumption is employed. Naturally, some later research extended continuous EDA to cover multivariate interactions by factorization of multivariate Gaussian distribution [Larrañaga 1999, Bosman 2000a]. Some other continuous EDAs include Gaussian kernels, mixtures of Gaussian distribution [Bosman 2000b], binary encoding by histograms [Tsutsui 2001], and variants using clustering [Lu 2005] and niching [Dong 2008] techniques.

This chapter proposed a continuous PMBGNP named PMBGNP with Actor-Critic (PMBGNP-AC). In PMBGNP-AC, Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ is used for the distribution of continuous variables of nodes. The mean value μ and standard deviation σ are constructed like those of classical PBILc [Sebag 1998]. However, a RL technique, i.e., Actor-Critic (AC) [Sutton 1998, Mabu 2007a], is designed to update the parameters (μ and σ). AC is applied to calculate the Temporal-Difference (TD) error to evaluate whether the selection of the continuous value (action) is better or worse than expected. The evaluation result is formulated as a scalar reinforcement signal which can decide whether the tendency to select this continuous value should be strengthened or weakened, allowing us to determine the shape of the probability density functions (μ and σ) of the Gaussian distribution.

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

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

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

Chapter 6 is organized as follows: Section 6.2 presents the proposed algorithm in details. The experimental study is carried out in section 6.3. Section 6.4 presents the summary of this chapter.

6.2 PMBGNP with Actor-Critic (PMBGNP-AC)

The individual representation of PMBGNP-AC is the same as that of GNP and PMBGNP. However, in the previous work of GNP and PMBGNP, the variables of each node is discretized in advance and fixed during the evolution process. As a result, the evolution mainly enforces the changes of node connections among different nodes. For discrete problems in which the variables of nodes do not need to be evolved, PMBGNP can work well. However, one can easily observe that PMBGNP cannot evolve the continuous variables since its probabilistic modeling is designed for the determination of node connections. To solve this problem, PMBGNP-AC is proposed in this section. Besides modeling the probability distribution of node connections using the existing PMBGNP, this section proposes another probabilistic model to estimate the probability density of the continuous variables in each node.

6.2.1 Probabilistic modeling

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

A) Probabilistic modeling of node connections

The construction of P_{nc} is actually the ones introduced in the previous chapters. It estimates the probabilities of connections between the nodes, where $P_{nc}(b(i), j)$ represents the connection probability from branch $b(i)$ of node i to node j. For each branch in the graph structure, the probabilities to connect to the next node is calculated to represent the probabilistic model P_{nc}.

To calculate the probabilities of node connections, the RPMBGNP proposed in Chapter 5 is used. In RPMBGNP, Sarsa Learning is employed to learn the experience of individuals to formulate the Q values. The Q values are used for the construction of P_{nc}, as shown in Eq. (5.3). The factorization of individuals to the sequences of state-action pairs are shown in Fig. 6.1-Middle.

As discussed in Chapter 5, this algorithm has been confirmed to outperform the MLE-based method since Q values play roles as weights to quantify the importance of node connections.

B) Probabilistic modeling of continuous variables

Most attempts on extending EDA to continuous domains are to use Gaussian distribution [Sebag 1998, Larrañaga 1999, Bosman 2006]. The advantages of utilizing Gaussian distribution is its simplicity of implementation and without loss of generality for solutions which ensures the performance in continuous domains. Most of the current continuous EDAs are based on Gaussian distribution or related variants. PMBGNP-AC proposed in this chapter is a straightforward extension of classical univariate continuous EDAs. That is, unidimensional Gaussian distribution is used for the distribution of continuous variables in each nodes of PMBGNP-AC.

Inspired by classical PBILc [Sebag 1998] and UMDAc [Larrañaga 1999], PMBGNP-AC optimizes Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ of each node. That is, the mean value μ and standard deviation σ of each continuous variable in each node are to be evolved. The probabilistic model P_{cv} of continuous variables in PMBGNP-AC consists of a set of probabilities $P^i_{cv}(x; \mu, \sigma)$, where $P^i_{cv}(x; \mu, \sigma)$ represents the probability density function (pdf) of the continuous variable of node i, described by:

$$P^i_{cv}(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right], \tag{6.1}$$

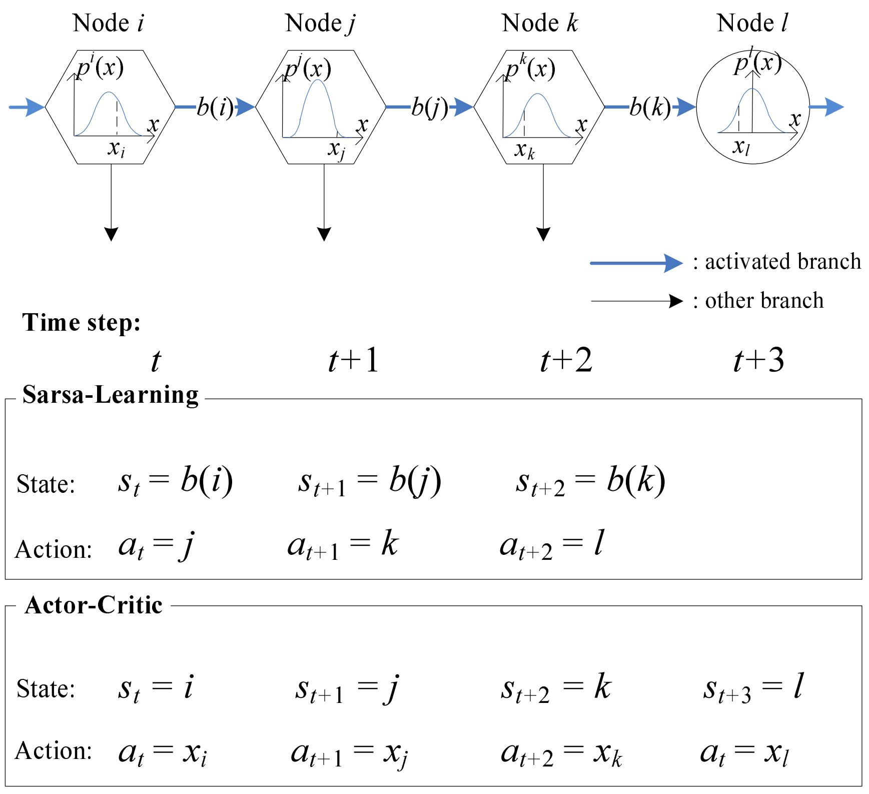

Node i
Node j
Node k
Node l
$p^i(x)$
$p^j(x)$
$p^k(x)$
$p^l(x)$
x
x_i
x_j
x_k
x_l
$b(i)$
$b(j)$
$b(k)$
: activated branch
: other branch
Time step:
t
$t+1$
$t+2$
$t+3$
Sarsa-Learning
State: $s_t = b(i)$ $s_{t+1} = b(j)$ $s_{t+2} = b(k)$
Action: $a_t = j$ $a_{t+1} = k$ $a_{t+2} = l$
Actor-Critic
State: $s_t = i$ $s_{t+1} = j$ $s_{t+2} = k$ $s_{t+3} = l$
Action: $a_t = x_i$ $a_{t+1} = x_j$ $a_{t+2} = x_k$ $a_t = x_l$

Figure 6.1: (Top): Example of node transitions of PMBGNP-AC individuals; (Middle): Factorization of node transitions to state-action pairs in Sarsa Learning of P_{nc}; (Bottom): Factorization of node transitions to state-action pairs in Actor-Critic of P_{cv}.

where,
x: continuous variables of node i.

In order to update the mean value μ and standard deviation σ of the continuous variable in each node, one may concern with the incremental learning [Sebag 1998] or Maximum Likelihood Estimation [Larrañaga 1999]. In this chapter, a novel method is used to update the Gaussian distribution by AC. The partial derivative of parameters μ and σ is firstly calculated as follows:

$$\frac{\partial P_{cv}^i(x;\mu,\sigma)}{\partial \mu} = \underbrace{\frac{2}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]}_{>0} \cdot (x-\mu), \tag{6.2}$$

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

$$\cdot \left[\frac{(x-\mu)^2}{\sigma^2} - 1\right] \tag{6.3}$$

As shown above, the front terms in the right side of Eq. (6.2) and (6.3) are always greater than 0. Therefore, inspired by the idea of gradient descent the updating directions of μ and σ are obtained given a sampled value x of this Gaussian distribution as follows:

$$\nabla(\mu; x) = x - \mu, \tag{6.4}$$

$$\nabla(\sigma; x) = \frac{(x-\mu)^2}{\sigma^2} - 1. \tag{6.5}$$

These two equations are the simplified versions of Eq. (6.2) and (6.3). Then, the following updating rules of μ and σ are used:

$$\mu \leftarrow \mu + \alpha_\mu \nabla(\mu; x), \tag{6.6}$$

$$\sigma \leftarrow \sigma + \alpha_\sigma \nabla(\sigma; x), \tag{6.7}$$

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

Using the sampled values x from the promising individuals, Eq. (6.6) and (6.7) become similar to those of PBILc and UMDAc. Based on this foundation, an extended version is proposed using a RL technique, i.e., Actor-Critic (AC) [Sutton 1998, Mabu 2007a].

In RL, an agent is interacted with its environment through observations and actions. At every step, the agent observes the current state of the environment, then chooses an action to change the state of the environment. At every step of choosing actions, a scalar reinforcement value is formulated according to the reward the agent obtains. This value is backwardly sent to the agent which allows the modification of its actions to maximize the reinforcement value. Based on this idea, this chapter proposes an algorithm PMBGNP with Actor-Critic (PMBGNP-AC) to update the Gaussian distribution. To incorporate AC, the state and action are defined according to the structure of PMBGNP-AC as follows (Fig. 6.1-Bottom):

Definition 12 (State) *State s is defined as a node in the directed graph of PMBGNP-AC.*

Definition 13 (Action) *Action a is defined as the selection of continuous variables in each node.*

Therefore, the set of states refers to the set of nodes in the directed graph of PMBGNP-AC. Consequently, the set of actions is infinite and bounded according to the range of the continuous variables. Note that these two definitions (**Definition** 8 and 9) are different from that of P_{nc} (see Fig. 6.1). With such definitions, AC is incorporated to the proposed algorithm, where the Gaussian distribution is known as the *actor* since it is used to select actions (sample continuous variables), and the *critic* is formulated as the state-value function to criticize the actions made by the actor. At each action selection, the critic evaluates the new state to determine whether this selection is better or worse than expected. The evaluation is formulated by the Temporal-Difference (TD) error δ as follows:

$$\delta_t = r_t + \gamma_{ac} V(s_{t+1}) - V(s_t), \tag{6.8}$$

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

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

$$V(s_t) \leftarrow V(s_t) + \alpha_{ac}\delta_t, \tag{6.9}$$

where,
α_{ac}: learning rate of AC.

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

$$\theta_t = \begin{cases} -1, & \text{for } \delta_t < 0 \\ 0, & \text{for } \delta_t = 0 \\ 1, & \text{for } \delta_t > 0 \end{cases} \tag{6.10}$$

Inserting this scalar reinforcement signal into Eq. (6.6) and (6.7), the final updating rules of the Gaussian distribution of PMBGNP-AC is obtained as follows:

$$\mu \leftarrow \mu + \alpha_\mu \nabla(\mu; x)\theta_t, \tag{6.11}$$

$$\sigma \leftarrow \sigma + \alpha_\sigma \nabla(\sigma; x)\theta_t. \tag{6.12}$$

6.2.2 Algorithm of PMBGNP-AC

The algorithm of PMBGNP-AC is a combination of the probabilistic model P_{nc} of node connections and P_{cv} of continuous variables of each node, as shown in Algorithm 7. As a result, the final probability of generating individual n by PMBGNP-AC is:

$$P(n) = \prod_{i \in N_{\text{node}}} \left[P_{cv}^i(x_i; \mu_i, \sigma_i) \prod_{b(i) \in B(i)} P_{nc}(b(i), j) \right]. \tag{6.13}$$

The initial Q values and state-values V are prepared in advance, which are set at zero in this chapter. P_{nc} is initialized to uniform distribution, while initial values of μ and σ in P_{cv} is determined problem-specifically.

Algorithm 7 : PMBGNP-AC

```
1:  $t \leftarrow 0$;
    Initialize population $Pop(t)$ with the size of $M$ and two probabilistic models $P_{nc}$
    and $P_{cv}$;
2:  Evaluate the fitness of $Pop(t)$;
3:  Select a set of promising individuals $Best(t)$ in which $(|Best(t)| = N < M)$;
4:  for $i \in Best(t)$ do
5:      Update the $Q$ values according to Sarsa Learning;
6:      Update the TD error $\delta$ and state-value function $V$ according to Eq. (6.8) and
        (6.9), respectively;
7:      Calculate $P_{cv}$ by updating $\mu$ and $\sigma$ according to Eq. (6.11) and (6.12), where
        $\nabla(\mu; x)$, $\nabla(\sigma; x)$, $\theta_t$ is obtained by Eq. (6.4), (6.5) and (6.10), respectively;
8:  Calculate $P_{nc}$ according to Eq. (5.3);
9:  Generate $P(t+1)$ by sampling $P_{nc}$ and $P_{cv}$;
    $t \leftarrow t + 1$;
10: Go back to step 2 until the terminal criteria is met.
```

6.3 Simulations

To evaluate the performance of the proposed algorithm, the Wall-Following problem of Khepera robot control is applied for the experimental study.

The following algorithms are selected for comparisons:

1) ***Standard GNP***: the discretization is used to transfer the continuous variables into discrete values, while the node connections are evolved by crossover and mutation.

2) ***RPMBGNP***: the discretization is used to transfer the continuous variables into discrete values, where the value of each node remains unchanged during evolution. In this case, only P_{nc} is constructed to evolve the node connections by Sarsa-Learning introduced in Chapter 5.

3) ***PBILc***: μ and σ of continuous PMBGNP are evolved like those of PBILc [Sebag 1998].

4) ***Sarsa Learning (Sarsa)*** [Sutton 1998]: A classical RL metod, Sarsa Learning in which the continuous state and action space is defined as the discrete space. ε-greedy policy is used for the action selection.

5) ***PMBGNP-AC***: the proposed algorithm.

6.3.1 Node functions and continuous variables

As shown in Table 6.1, the node functions used in PMBGNP-AC are set similarly as the ones in the previous chapters. However, the discretization process is not used to transfer the continuous domain to discrete values in PMBGNP-AC. Instead, it directly evolve the continuous variables by updating the Gaussian distribution. The roles of continuous variables in judgment/processing nodes to be optimized are as follows:

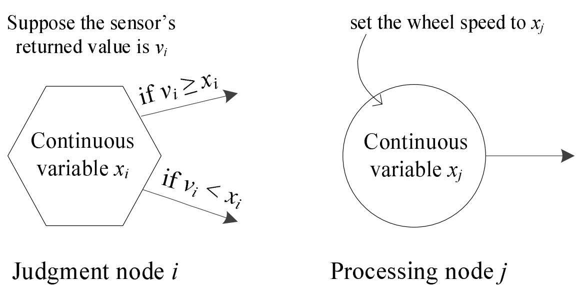

Suppose the sensor's returned value is v_i
Continuous variable x_i
if $v_i \geq x_i$
if $v_i < x_i$
Judgment node i
set the wheel speed to x_j
Continuous variable x_j
Processing node j

Figure 6.2: Roles of continuous variables in judgment/processing nodes.

Judgment node (i.e., node i): continuous variable x_i divides the domain of its sensor value into two intervals ([0, x_i) and [x_i, 1023]), where the selection of branches is determined by the comparison of real returned value and x_i.
Processing node (i.e., node j): continuous variable x_j ([-10, 10]) formulates the speed of its corresponding wheel motor.

These two kinds of continuous variables are to be evolved to determine the final solutions, as an example shown in Fig. 6.2. Meanwhile, in standard RPMBGNP, these two types of variables are discretized in advance. For judgment nodes, the domain is divided into two fixed intervals, [0, 1000) and [1000, 1023]. In processing nodes, the domain is discretized into the set of {-10, -5, 0, 5, 10} to determine the robot's speed (Details can be found in chapter 3).

The discretization of Sarsa is done similarly as RPMBGNP, where the state space is defined by the combinations of sensor values and the action space is defined by different settings of the robot's speed. Therefore, there are total $2^8 = 256$ states and $5 \times 5 = 25$ actions in Sarsa.

6.3.2 Parameter settings

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

The number of judgment nodes for each judgment function is set at 5. The number of processing nodes for each processing function is set at 10. Therefore, the program size[1] of each individual is $5 \times 8 + 10 \times 2 = 60$.

The simulation conditions are defined as shown in Table 6.2. All these settings are the appropriate ones defined by hand-tuning.

6.3.3 Simulation results and analysis

Five simulations of $ST \in \{100, 200, 300, 400, 500\}$ are carried out to testify the effectiveness and scalability of the proposed algorithm. The simulation results are

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

Table 6.1: Node functions used for Khepera robot.

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

Table 6.2: Simulation conditions.

	GNP	RPMBGNP	PBILc	PMBGNP-AC
Population size M	300	2000	2000	2000
– elite ind.	1	100	100	100
– crossover ind.	120	–	–	–
– mutation ind.	179	–	–	–
– promising ind. N	–	1000	1000	1000
Program size $\|N_{\text{node}}\|$	60	60	60	60
Crossover rate	0.1	–	–	–
Mutation rate	0.01	–	–	–
Other parameters	(— Sarsa-Learning) $\alpha_s = 0.1$, $\gamma_s = 0.9$ (— Actor-Critic) $\alpha_{ac} = 0.1$, $\gamma_{ac} = 0.9$ $\alpha_\mu = 0.05$, $\alpha_\sigma = 0.05$ (— ε-greedy policy) $\varepsilon = 0.1$			
Terminal condition	300,000 fitness evaluations			

the average over 30 independent runs.

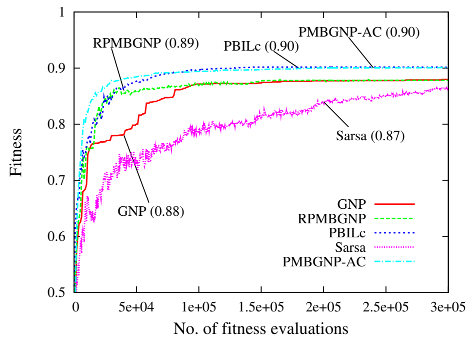
A) Fitness results

The detailed fitness values and curves of the compared methods are shown in Table 6.3 and Fig. 6.3. Each value of Table 6.3 indicates the average fitness with standard deviation, where the bold ones denote the best results of the problems. The analysis of different methods is performed as follows:

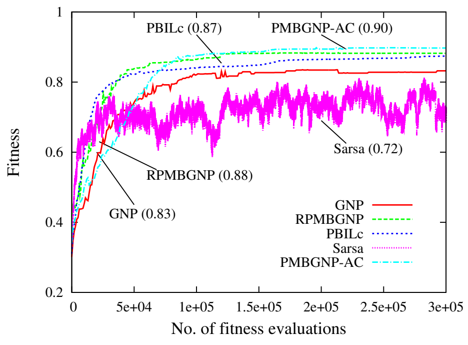
GNP: In most cases of the five problems, GNP performs worse results than the variants of PMBGNP, and only outperforms Sarsa. This is due to the lack of evolution ability by standard genetic operators, i.e., crossover and mutation, which has been verified in the previous chapters. The results show that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

RPMBGNP: RPMBGNP achieves better performance than that of Sarsa and GNP. However, it has worse performance than PMBGNP-AC. This is due to the loss of the solution precision by the discretization. On the other hand, RPMBGNP actually obtains very stable performances among five problems (has the smallest standard deviation), which is because of the balance of exploitation and exploration by Boltzmann distribution.

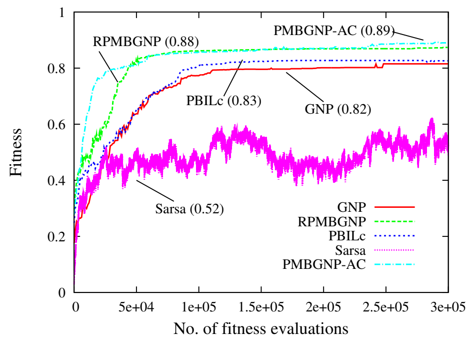
PBILc: In this method, only the information of three individuals (the best two and the worst one) are used to update the pdf of Gaussian distribution [Sebag 1998]. It might cause that the search space is explored in a too restricted region and



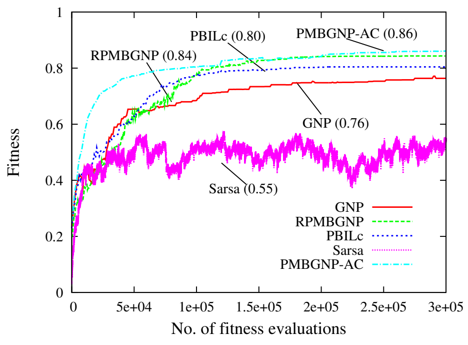
(a) $ST = 100$



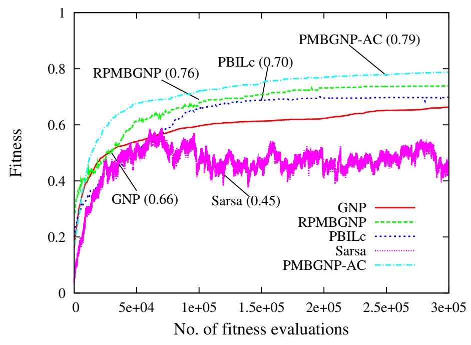
(b) $ST = 200$



(c) $ST = 300$



(d) $ST = 400$



(e) $ST = 500$

Figure 6.3: Fitness curves in five wall-following problems.

Table 6.3: The fitness results over 30 independent runs.

	Fitness (std. dev.)				
ST	100	200	300	400	500
GNP	0.88 ± 0.02	0.83 ± 0.11	0.82 ± 0.09	0.76 ± 0.08	0.66 ± 0.15
RPMBGNP	0.89 ± 0.01	0.88 ± 0.06	0.88 ± 0.03	0.84 ± 0.10	0.76 ± 0.09
PBILc	$\mathbf{0.90 \pm 0.01}$	0.87 ± 0.09	0.83 ± 0.09	0.80 ± 0.15	0.70 ± 0.12
Sarsa	0.87 ± 0.04	0.72 ± 0.11	0.52 ± 0.21	0.55 ± 0.19	0.45 ± 0.22
PMBGNP-AC	$\mathbf{0.90 \pm 0.02}$	$\mathbf{0.90 \pm 0.03}$	$\mathbf{0.89 \pm 0.10}$	$\mathbf{0.86 \pm 0.14}$	$\mathbf{0.79 \pm 0.11}$

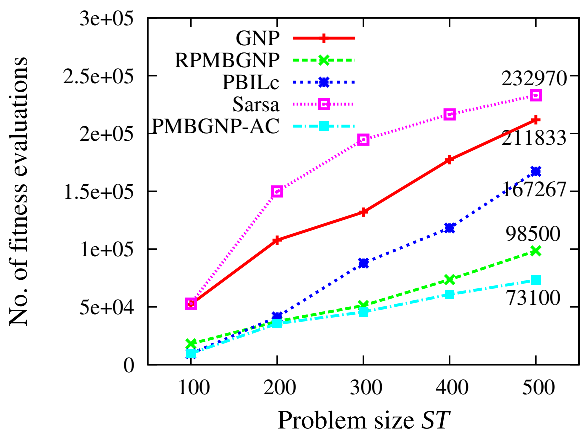


Figure 6.4: Average fitness evaluation for five wall-following problems.

slow speed. Therefore, this method can ensure quite good performance in simple problems ($ST = 100, 200$), however, obtain poor results when the problem size increases ($ST = 500$).

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

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

Fig. 6.4 plots the average number of required fitness evaluations to solve the wall-following problems. Results confirm the higher evolution ability of PMBGNP-AC over the other methods.

B) Statistical analysis

In order to further analyze the simulation results, the statistical analysis of

Table 6.4: The t-test results of PMBGNP-AC and the other methods (The bold ones denote statistically significant difference).

	t-test (p value)				
ST	100	200	300	400	500
PMBGNP-AC vs. GNP	**6.67e-05**	**9.50e-05**	**1.19e-04**	**5.71e-05**	**2.53e-05**
PMBGNP-AC vs. RPMBGNP	**1.59e-02**	7.57e-01	7.18e-01	**3.37e-02**	**1.40e-02**
PMBGNP-AC vs. PBILc	9.52e-01	**2.79e-02**	**2.20e-02**	**1.17e-02**	**1.39e-03**
PMBGNP-AC vs. Sarsa	**4.68e-05**	**2.05e-06**	**5.12e-07**	**6.46e-08**	**2.86e-05**

different methods by t-test of the number of required fitness evaluations is shown in Table 6.4. This shows that PMBGNP-AC statistically outperforms GNP and Sarsa among all the problems, and outperforms RPMBGNP and PBILc in some problems. Particularly, in the case of the largest problem size ($ST = 500$), PMBGNP-AC is statistically superior than the other methods.

6.4 Summary

This chapter extends PMBGNP from the discrete domain to continuous cases. The conventional research on the topic of continuous EDAs was followed and a novel method was formulated to learn Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ by a Reinforcement Learning method, i.e., Actor-Critic (AC). The resulting PMBGNP-AC method can be thought as an extension of PBILc, where AC can implicitly update the pdf of Gaussian distribution by considering multivariate interactions. The results show that in the Wall-Following problems of autonomous robots, PMBGNP-AC outperforms the conventional methods, including conventional genetic operators based EAs, discretization based EDA, PBILc based variant and classical RL method. Moreover, the scalability of PMBGNP-AC is confirmed by different settings of the problem size.

Chapter 7

Conclusions

Contents

This thesis proposes a novel Estimation of Distribution Algorithm (EDA) named Probabilistic Model Building Genetic Network Programming (PMBGNP), and systematically studied PMBGNP by proposing several enhanced algorithms and applying to several problems. This chapter first presents the conclusions of the thesis and highlight the contributions. Finally, some future directions of this topic are addressed.

The conclusions of the thesis are drawn through two aspects: algorithm aspect and application aspect.

7.1 Algorithm aspect

In Chapter 2, PMBGNP that extends EDA from bit-string and tree structures to graph structures is proposed. Inspired by classical EDAs, such as PBIL and UMDA, PMBGNP constructs a probabilistic model from the promising individuals by Maximum Likelihood Estimation (MLE) to generate the new population. As a result, it inherits the advantages of EDA, where the frequent breakage of the BBs in conventional genetic operator-based EAs can be avoid in certain respects. On the other hand, due to its distinguished graph structure, PMBGNP can achieve higher expression ability, where a large number of problems can be explored and solved efficiently and effectively comparing with the conventional EDAs.

Chapter 3 discusses the population diversity loss of PMBGNP. First, this chapter theoretically compares the sensitivity of the diversity loss among PMBGA, PMBGP and PMBGNP. Based on the discussion, a hybrid PMBGNP (hPMBGNP) is proposed to maintain the population diversity of PMBGNP, where two methods named multiple probability vectors and genetic operator are introduced. This chapter finally verifies the effectiveness of hPMBGNP theoretically and empirically.

Chapter 4 and chapter 5 mainly focus on the improvement of PMBGNP by studying the integration of PMBGNP and Reinforcement Learning (RL). Chapter 4 applies RL to identify and extract useful sub-structures of the bad individuals, while the sub-structures are used in the probabilistic modeling of PMBGNP. The simulation results show that the proposed method can accelerate the evolution of PMBGNP in terms of requiring smaller number of fitness evaluations.

Chapter 5 provides another method to utilize the integration of EDA and RL called Reinforced PMBGNP (RPMBGNP). In RPMBGNP, RL is used to learn knowledge/experience during the individual executions. The collected knowledge formulated by Q values can correctly measure the quality of node connections. As a result, RPMGNP incorporates the learnt Q values into its probabilistic modeling, which has shown better performance than the MLE-based PMBGNP and various classical algorithms from the literature of EA, EDA and RL. On the other hand, another advantage of RPMBGNP is that although RL is applied to the framework of PMBGNP, the computation time still remained in the same level as PMBGNP, while classical Bayesian network-based EDAs generally requires much time cost for the probabilistic modeling.

From Chapter 2 to Chapter 5, PMBGNP and its variants are applied to solve the discrete optimization problems. Chapter 6 extends PMBGNP to continuous domains, where an algorithm named PMBGNP with Actor-Critic (PMBGNP-AC) was proposed. In PMBGNP-AC, the continuous variables of nodes are formulated by Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. Actor-Critic (AC), is designed to update the parameters (μ and σ) of the Gaussian distribution. This chapter fulfills the topics of integrating EDA and RL techniques, and extends PMBGNP to solve continuous optimization problems.

Concretely speaking, the contributions of the thesis can be mainly summarized in the following two points from the perspective of algorithms:

1. Extending EDA from GA's bit-string structure and GP's tree structure to a more complex individual structure: directed graph structure.

2. Studying on the integration of EDA and RL.

The thesis focuses on studying these two points by proposing the algorithm of PMBGNP and its several variants.

7.2 Application aspect

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 [Zhang 2008], dynamic problems [Yang 2008] and bioinformatics [Santana 2008a], etc. Despite many different implementations, one important challenge of EDA is to explore it to some other applications [Santana 2008b].

From the perspective of applications, the thesis contributes to explore the problem solving of EDA to two classes of problems, including data mining problems and the problems of controlling the agents' behavior.

Chapter 2 studies on applying PMBGNP to solve one of the data mining problems, class association rule mining (CARM), where the empirical studies show the superiority of PMBGNP over classical algorithms in terms of the efficiency of rule extraction and the accuracy of prediction.

From Chapter 3 to Chapter 6, PMBGNP is mainly studied on the problems of controlling the agents' behavior. There is a wide range of work on studying EAs, such as GP and Evolutionary Programming (EP) [Fogel 1994], to successfully solve the problems of controlling the agents' behavior, while until now, there is only a few work on studying EDA to solve such kind of problems. [Salustowicz 1997] and [Salustowicz 1998] were the limited work that applies PIPE, a univariate PMBGP, to solve the problems of controlling the agents' behavior. Another work on applying EDA to control the agents is a recent one named EDA-RL [Handa 2009], which has been reported to successfully solve simple problems but fail on some complex partially observable problems. Meanwhile, it is meaningful to study the problems of controlling the agents' behavior, since many real-world applications can be solved by such kind of agent systems, i.e., stock trading, foreign exchange prediction and real robot control, etc.

Through the studies on both benchmark testbed the Tileworld system and the robot control problems, it has been confirmed that PMBGNP and its variants can achieve better results than the classical state-of-the-art algorithms in the fields of EA, EDA and RL.

Further studies will include the extensions of PMBGNP to the other problems, such as stock trading, foreign exchange prediction and dynamic optimization problems, etc. More in-depth theoretical analysis of PMBGNP will also be done in the later research. On the other hand, improving the method of integrating EDA and RL, and extending it to the other EDAs will be another future research.

Bibliography

[Baldwin 1896] J. M. Baldwin. *A new factor in evolution.* American Naturalist, vol. 30, no. 354, pages 441–451, 1896. (Cited on page 67.)

[Baluja 1994] S. Baluja. *Population-based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Leaning.* Tech. Report CMU-CS-94-163, Carnegie Mellon University, 1994. (Cited on pages 8 and 15.)

[Baluja 1997] S. Baluja and S. Davies. *Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space.* In Proc. of the Int'l Conf. on Machine Learning, pages 30–38, 1997. (Cited on page 10.)

[Beyer 2002] H.-G. Beyer and H.-P. Schwefel. *Evolution Strategies: A Comprehensive Introduction.* J. Natural Computing, vol. 1, no. 1, pages 3–52, 2002. (Cited on page 2.)

[Bonet 1997] J. S. De Bonet, C. L. Isbell and P. Viola. *MIMIC: Finding Optima by Estimating Probability Densities.* In Advances in Neural Information Processing Systems, pages 424–430, 1997. (Cited on page 10.)

[Bosman 2000a] P. A. N. Bosman and D. Thierens. *Expanding from discrete to continuous estimation of distribution algorithms: The IDEA.* In Proc. of the 5th Conf. on Parallel Problem Solving from Nature, pages 767–776, 2000. (Cited on page 94.)

[Bosman 2000b] P. A. N. Bosman and D. Thierens. *Mixed IDEAs.* Tech. Report UUCS- 2000-45, Utrecht University, 2000. (Cited on page 94.)

[Bosman 2006] P. A. N. Bosman and D. Thierens. *Numerical Optimization with Real-Valued Estimation-of-Distribution Algorithms.* In M. Pelikan, K. Sastry and E. Cantú-Paz, editeurs, Scalable Optimization via Probabilistic Modeling, chapitre 5, pages 91–120. Springer-Verlag, Berlin, Germany, 2006. (Cited on pages 11, 94 and 96.)

[Brownlee 2008] 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. IEEE Congress on Evol. Comput., pages 2621–2628, 2008. (Cited on pages 52, 53, 58 and 61.)

[Brownlee 2009] A. Brownlee, J. McCall, S. Shakya and Q. Zhang. *Structure learning and optimisation in a Markov-network based estimation of distribution algorithm.* In Proc. of the IEEE Congress on Evol. Comput., pages 447–454, 2009. (Cited on page 68.)

[Chen 2010] B. Chen and J. Hu. *A hybrid EDA for protein folding based on HP model.* IEEJ Trans. on Electrical and Electronic Engineering, vol. 5, no. 4, pages 459–466, 2010. (Cited on page 53.)

[Cyberbotics] Cyberbotics. *Webots software.* In http://www.cyberbotics.com/. (Cited on pages 35, 42 and 95.)

[Dong 2008] W. Dong and X. Yao. *NichingEDA: Utilizing the diversity inside a population of EDAs for continuous optimization.* In Proc. of the IEEE Congress on Evol. Comput., pages 1260–1267, 2008. (Cited on page 94.)

[Downing 2001] K. L. Downing. *Reinforced genetic programming.* Genetic Programming and Evolvable Machines, vol. 2, no. 3, pages 259–288, 2001. (Cited on pages 66 and 67.)

[Eguchi 2006] 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, pages 179–193, 2006. (Cited on pages 16, 18, 19, 24, 72 and 74.)

[Etxeberria 1999] R. Etxeberria and P. Larrañaga. *Global optimization using Bayesian networks.* In Proc. of the Symp. on Artif. Intell., pages 332–339, 1999. (Cited on page 10.)

[Fogel 1994] D. B. Fogel. *An Introduction to Simulated Evolutionary Optimization.* IEEE Trans. Neural Netw., vol. 5, no. 1, pages 3–14, 1994. (Cited on pages 2, 17 and 108.)

[Freund 1996] Y. Freund and R. Schapire. *Experiments with a New Boosting Algorithm.* In Proc. of the Int'l Conf. on Machine Learning, pages 148–156, 1996. (Cited on page 32.)

[Goldberg 1989a] D. E. Goldberg. Genetic algorithm in search, optimization and machine learning. Addison-Wesley, 1989. (Cited on page 2.)

[Goldberg 1989b] D.E. Goldberg, B. Korb and K. Deb. *Messy genetic algorithms: Motivation, analysis, and first results.* Complex Systems, vol. 5, no. 3, page 493¨C530, 1989. (Cited on page 6.)

[Hammersley 1971] J. M. Hammersley and P. Clifford. Markov fields on finite graphs and lattices. Unpublished, 1971. (Cited on page 72.)

[Handa 2007] H. Handa. *The Effectiveness of Mutation Operation in the case of Estimation of Distribution Algorithms.* BioSystems, vol. 87, pages 243–251, 2007. (Cited on page 34.)

[Handa 2009] 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. (Cited on pages 11, 68 and 108.)

[Harik 1997] G. Harik. *Learning linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* PhD thesis, Dept. Computer Science, University of Michigan, Ann Arbour, Michigan, Ann Arbour, 1997. (Cited on page 6.)

[Harik 1999] G. R. Harik, F. G. Lobo and D. E. Goldberg. *The Compact Genetic Algorithm.* IEEE Trans. Evol. Comput., vol. 3, no. 4, pages 287–297, 1999. (Cited on page 9.)

[Harik 2006] G. R. Harik, F. G. Lobo and K. Sastry. *Linkage Learning via Probabilistic Modeling in the Extended Compact Genetic Algorithm (ECGA).* In M. Pelikan, K. Sastry and E. Cantú-Paz, editeurs, Scalable Optimization via Probabilistic Modeling, chapitre 3, pages 39–61. Springer-Verlag, Berlin, Germany, 2006. (Cited on page 10.)

[Hasegawa 2008] Y. Hasegawa and H. Iba. *A Bayesian Network Approach to Program Generation.* IEEE Trans. Evol. Comput., vol. 12, no. 6, pages 750–764, 2008. (Cited on pages 11, 16 and 71.)

[Hirasawa 2001] K. Hirasawa, M. Okubo, H. Katagiri, J. Hu and J. Murata. *Comparison between Genetic Network Programming (GNP) and Genetic Programming (GP).* In Proc. of the IEEE Congress on Evol. Comput., pages 1276–1282, 2001. (Cited on pages 12, 16, 17 and 78.)

[Hirasawa 2008] 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, pages 535–550, 2008. (Cited on pages 16 and 24.)

[Holand 1975] J. H. Holand. Andaptation in natural and artificial systems. Ann-Arbor, University of Michigan Press, 1975. (Cited on pages 2 and 4.)

[Hong 2009] 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. IEEE Int'l Conf' on Fuzzy Systems, pages 1806–1813, 2009. (Cited on pages 53 and 58.)

[Iba 1996] H. Iba. *Emergent cooperation for multiple agents using genetic programming.* In Proc. of the Conf. on Parallel Problem Solving from Nature, pages 32–41, 1996. (Cited on pages 74 and 78.)

[K-Team Corp.] K-Team Corp. In http://www.k-team.com/. (Cited on pages 35, 42 and 95.)

[Kamio 2005] 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, pages 318–333, 2005. (Cited on pages 66 and 67.)

[Koza 1992] J. R. Koza. Genetic programming, on the programming of computers by means of natural selection. MIT Press, 1992. (Cited on pages 2 and 79.)

[Koza 1994] J. R. Koza. Genetic programming ii, automatic discovery of reusable programs. MIT Press, 1994. (Cited on page 2.)

[Larrañaga 1999] P. Larrañaga, R. Etxeberria, J. A. Lozano and J. M. Peña. *Optimization by learning and simulation of Bayesian and Gaussian networks.* Tech. Report EHU-KZAA-IK-4-99, Intelligent Systems Group, Dept. of Comput. Sci. and Artif. Intell., University of the Basque Country, 1999. (Cited on pages 11, 94, 96 and 97.)

[Larrañaga 2002] P. Larrañaga and J. A. Lozano. Estimation of distribution algorithms. a new tool for evolutionary computation. Kluwer Academic Publishers, 2002. (Cited on pages 6 and 15.)

[Li 2001] W. Li, J. Han, and J. Pei. *CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules.* In Proc. of the IEEE Int'l Conf. on Data Mining, pages 369–376, 2001. (Cited on page 32.)

[Li 2010a] 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.* J. Advanced Computational Intelligence and Intelligent Informatics, vol. 14, no. 5, pages 497–509, 2010. (Cited on page 16.)

[Li 2010b] X. Li, S. Mabu and K. Hirasawa. *Towards the maintenance of population diversity: A hybrid probabilistic model building genetic network programming.* Trans. Japanese Society for Evol. Comput., vol. 1, no. 1, pages 89–101, 2010. (Cited on page 35.)

[Li 2010c] 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. (Cited on pages 16 and 52.)

[Li 2011] X. Li, S. Mabu and K. Hirasawa. *Use of infeasible individuals in probabilistic model building genetic network programming.* In Proc. of the Genetic and Evol. Comput. Conf., pages 601–608, 2011. (Cited on page 66.)

[Li 2012] X. Li, S. Mabu and K. Hirasawa. *An extended Probabilistic model building genetic network programming using both of good and bad individuals.* IEEJ Trans. on Electrical and Electronic Engineering, 2012. accepted. (Cited on page 53.)

[Li 2013] X. Li, S. Mabu and K. Hirasawa. *A Novel Graph-Based Estimation of Distribution Algorithm and Its Extension Using Reinforcement Learning.* IEEE Trans. Evol. Comput., 2013. accepted. (Cited on page 66.)

[Liu 1998] B. Liu, W. Hsu, and Y. Ma. *Integrating Classification and Association Rule Mining.* In Proc. of the ACM Int'l Conf. on Knowledge Discovery and Data Mining, pages 80–86, 1998. (Cited on page 32.)

[Lu 2005] Q. Lu and X. Yao. *Clustering and learning Gaussian distribution for continuous optimization.* IEEE Trans. on Syst. Man & Cyber. - C, vol. 35, no. 2, pages 195–204, 2005. (Cited on page 94.)

[Mabu 2007a] S. Mabu, Y. Chen, K. Hirasawa and J. Hu. *Stock trading rules using genetic network programming with actor-critic.* In Proc. of the IEEE Congress on Evol. Comput., pages 508–515, 2007. (Cited on pages 95 and 98.)

[Mabu 2007b] 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, pages 369–398, 2007. (Cited on pages 12, 16, 19, 24, 35, 66, 67, 72, 74 and 78.)

[Mabu 2011a] 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, pages 130–139, 2011. (Cited on pages 16 and 24.)

[Mabu 2011b] S. Mabu and K. Hirasawa. *Enhanced Rule Extraction and Classification Mechanism of Genetic Network Programming for Stock Trading Signal Generation.* In Proc. of the Genetic and Evol. Comput. Conf., pages 1659–1666, 2011. (Cited on page 24.)

[McKay 2010] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan and M. O'Neill. *Grammar-based Genetic Programming: a survey.* Genetic Programming and Evolvable Machine, vol. 11, no. 3-4, pages 365–396, 2010. (Cited on page 11.)

[Miller 2000] J. F. Miller and P. Thomson. *Cartesian Genetic Programming.* In Proc. of the Eur. Conf. on Genetic Programming, pages 121–132, 2000. (Cited on pages 16 and 17.)

[Miquélez 2004] T. Miquélez, E. Bengoetzea and P. Larrañaga. *Evolutionary computation based on Bayesian classifiers.* Int'l J. Appl. Math. Comput. Sci., vol. 14, no. 3, pages 335–349, 2004. (Cited on page 53.)

[Mühlenbein 1996] H. Mühlenbein and G. Paaß. *From Recombination of Genes to the Estimation of Distributions I. Binary Parameters.* In Proc. of the Conf. on Parallel Problem Solving from Nature, pages 178–187, 1996. (Cited on pages 6, 7, 15 and 52.)

[Mühlenbein 1999a] H. Mühlenbein and T. Mahnig. *FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions.* Evol. Comput., vol. 7, no. 4, pages 353–376, 1999. (Cited on page 10.)

[Mühlenbein 1999b] H. Mühlenbein, T. Mahnig and A. O. Rodriguez. *Schemata, distributions and graphical models in evolutionary optimization.* Journal of Heuristics, vol. 5, no. 2, pages 215–247, 1999. (Cited on page 10.)

[Munetomo 2008] M. Munetomo, N. Murao and K. Akama. *Introducing Assignment Functions to Bayesian Optimization Algorithms.* Information Sciences, vol. 178, no. 1, pages 152–163, 2008. (Cited on pages 53 and 66.)

[Nordin 1998] P. Nordin, W. Banzhaf and M. Brameier. *Evolution of a world model for a miniature robot using genetic programming.* Robotics and Autonomous Systems, vol. 25, pages 105–116, 1998. (Cited on page 42.)

[Paul 2003] T. K. Paul and H. Iba. *Reinforcement learning estimation of distribution algorithm.* In Proc. of the Genetic and Evol. Comput. Conf., pages 1259–1270, 2003. (Cited on page 68.)

[Pelikan 2002a] M. Pelikan, D. E. Goldberg and E. Cantú-Paz. *Linkage Problem, Distribution Estimation, and Bayesian Networks.* Evol. Comput., vol. 8, no. 3, pages 311–341, 2002. (Cited on pages 10, 16, 34 and 52.)

[Pelikan 2002b] M. Pelikan, D. E. Goldberg and F. G. Lobo. *A Survey of Optimization by Building and Using Probabilistic Models.* Computational Optimization and Applications, vol. 21, no. 1, pages 5–20, 2002. (Cited on pages 11 and 16.)

[Poli 1996] R. Poli. *Parallel distributed genetic programming.* Tech. Report CSRP-96-15, School of Computer Science, The University of Birmingham, 1996. (Cited on pages 16 and 17.)

[Poli 2008a] R. Poli, W. B. Langdon and N. F. McPhee. A field guide to genetic programming. Published via http://lulu.com/ and freely available at http://www.gp-field-guide.org.uk/, 2008. (With contributions by J. R. Koza). (Cited on page 79.)

[Poli 2008b] R. Poli and N. F. McPhee. *A linear Estimation-of-Distribution GP System.* In Proc. of the Eur. Conf. on Genetic Programming, pages 206–217, 2008. (Cited on page 11.)

[Pollack 1990] M. E. Pollack and M. Ringuette. *Introducing the tile-world: Experimentally evaluating agent architectures.* In Proc. of the Conf. of the American Association for Artificial Intelligence, pages 183–189, 1990. (Cited on page 74.)

[Pollack 1994] M. E. Pollack, D. Joslin, A. Nunes, S. Ur and E. Ephrati. *Experimental investigation of an agent commitment strategy.* Tech. Report 94-31, Dept. of Computer Science, Univ. of Pittsburgh, 1994. (Cited on page 74.)

[Quinlan 1993] J. Quinlan. C4.5: Programs for machine learning. Morgan Kaufmann, 1993. (Cited on page 32.)

[Salustowicz 1997] R. P. Salustowicz and J. Schmidhuber. *Probabilistic Incremental Program Evolution.* Evol. Comput., vol. 5, no. 2, pages 123–141, 1997. (Cited on pages 9, 35, 72, 79 and 108.)

[Salustowicz 1998] R. P. Salustowicz, M. A. Wiering and J. Schmidhuber. *Learning Team Strategies: Soccer Case Studies.* Machine Learning, vol. 33, no. 2-3, pages 263–282, 1998. (Cited on page 108.)

[Santana 2005] R. Santana. *Estimation of distribution algorithm with kikuchi approximations.* Evol. Comput., vol. 13, no. 1, pages 67–97, 2005. (Cited on page 68.)

[Santana 2008a] 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, pages 418–438, 2008. (Cited on pages 11, 16 and 107.)

[Santana 2008b] R. Santana, P. Larrañaga and J.A. Lozano. *Research topics in discrete estimation of distribution algorithms.* Memetic Computing, vol. 1, no. 1, pages 35–54, 2008. (Cited on page 107.)

[Sastry 2003] K. Sastry and D. E. Goldberg. *Probabilistic Model Building and Competent Genetic Programming.* In R. L. Riolo and B. Worzel, editeurs, Genetic Programming Theory and Practice, volume 13, pages 205–220. 2003. (Cited on page 11.)

[Sastry 2005] K. Sastry, H. A. Abbass, D. E. Goldberg and D. D. Johnson. *Substructural Niching in Estimation of Distribution Algorithms.* In Proc. of the Genetic and Evol. Comput. Conf., pages 671–678, 2005. (Cited on page 34.)

[Sebag 1998] M. Sebag and A. Ducoulombier. *Extending Population-based Incremental Learning to Continuous Search Spaces.* In Proc. of the Conf. on Parallel Problem Solving from Nature, pages 418–427, 1998. (Cited on pages 94, 95, 96, 97, 100 and 102.)

[Shakya 2006] S. K. Shakya. *DEUM: A framework for an Estimation of Distribution Algorithm based on Markov Random Fields.* PhD thesis, School of Computing, The Robert Gordon University, Aberdeen, UK, April 2006. (Cited on pages 68 and 72.)

[Shan 2004] Y. Shan, R. I. McKay, R. Baxter, H. Abbass, D. Essam and N. X. Hoai. *Grammar model-based Program Evolution.* In Proc. of the IEEE Congress on Evol. Comput., pages 478–485, 2004. (Cited on page 11.)

[Shan 2006] Y. Shan, R. I. McKay, D. Essam and H. A. Abbass. *A Survey of Probabilistic Model Building Genetic Programming.* In M. Pelikan, K. Sastry and E. Cantú-Paz, editeurs, Scalable Optimization via Probabilistic Modeling, chapitre 6, pages 121–154. Springer-Verlag, Berlin, Germany, 2006. (Cited on pages 11 and 16.)

[Shapiro 2006] J. L. Shapiro. *Diversity Loss in General Estimation of Distribution Algorithms.* In Proc. of the 9th Conf. on Parallel Problem Solving from Nature, pages 92–101, 2006. (Cited on pages 34 and 35.)

[Shimada 2006a] K. Shimada, K. Hirasawa and J. Hu. *Class Association Rule Mining with Chi-Squared Test Using Genetic Network Programming.* In Proc. of the IEEE Conf. Syst., Man, Cybern., pages 5338–5344, 2006. (Cited on pages 25 and 26.)

[Shimada 2006b] K. Shimada, K. Hirasawa and J. Hu. *Genetic Network Programming with Acquisition Mechanisms of Association Rules.* J. Advanced Computational Intelligence and Intelligent Informatics, vol. 10, no. 1, pages 102–111, 2006. (Cited on pages 16 and 24.)

[Sutton 1998] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. MIT Press, 1998. (Cited on pages 53, 54, 55, 68, 80, 95, 98 and 100.)

[Teller 1995] A. Teller and M. Veloso. *PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System.* Tech. Report CMU-CS-95-101, Carnegie Mellon University, 1995. (Cited on pages 16 and 17.)

[Tsutsui 2001] S. Tsutsui, M. Pelikan and D. E. Goldberg. *Probabilistic model-building genetic algorithms using marginal histograms in continuous domain.* In Proc. of the KES, pages 112–121, 2001. (Cited on page 94.)

[Whigham 1995] P. A. Whigham. *Grammatically-based Genetic Programming.* In Proc. of the Workshop on Genetic Programming: From Theory to Real-world Applications, pages 33–41, 1995. (Cited on page 11.)

[Yanai 2003] 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. (Cited on pages 10, 35, 52, 53, 66, 72 and 79.)

[Yang 2008] S. Yang and X. Yao. *Population-Based Incremental Learning With Associative Memory for Dynamic Environments.* IEEE Trans. Evol. Comput., vol. 12, no. 5, pages 542–561, 2008. (Cited on pages 11 and 107.)

[Yin 2003] X. Yin and J. Han. *Classification based on Predictive Association Rules.* In Proc. of the SIAM Int'l Conf. on Data Mining, pages 331–335, 2003. (Cited on page 32.)

[Yu 2008] Y. Yu and Z. H. Zhou. *On the usefulness of infeasible solutions in evolutionary search: A theoretical study.* In Proc. IEEE Congress on Evol. Comput., pages 835–840, 2008. (Cited on page 52.)

[Zhang 2004] Q. Zhang and H. Mühlenbein. *On the Convergence of a Class of Estimation of Distribution Algorithms.* IEEE Trans. Evol. Comput., vol. 8, no. 2, pages 127–136, 2004. (Cited on pages 15 and 52.)

[Zhang 2005] 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, pages 192–200, 2005. (Cited on page 16.)

[Zhang 2008] 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, pages 41–63, 2008. (Cited on pages 11 and 107.)

[Zhou 2008] H. Zhou, W. Wei, K. Shimada, S. Mabu and K. Hirasawa. *Time Related Association Rules Mining with Attributes Accumulation Mechanism and its Application to Traffic Prediction.* J. Advanced Computational Intelligence and Intelligent Informatics, vol. 12, no. 5, pages 467–478, 2008. (Cited on pages 24, 25, 26, 28 and 29.)

Research Achievements

Journal papers

1. **X. Li**, S. Mabu, and K. Hirasawa, "A Novel Graph-Based Estimation of Distribution Algorithm and Its Extension Using Reinforcement Learning," IEEE Trans. on Evolutionary Computation. accepted

2. **X. Li**, S. Mabu, and K. Hirasawa, "An extended Probabilistic model building genetic network programming using both of good and bad individuals," IEEJ Trans. on Electrical and Electronic Engineering. accepted

3. B. Li, **X. Li**, S. Mabu, and K. Hirasawa, "Evolving graph-based chromosome by means of variable size genetic network programming with binomial distribution," IEEJ Trans. on Electrical and Electronic Engineering. accepted

4. **X. Li**, S. Mabu, B. Li and K. Hirasawa, "Probabilistic Model Building Genetic Network Programming using Reinforcement Learning," Transaction of the Japanese Society for Evolutionary Computation, Vol. 2, No.1, pp. 29-40, October, 2011.

5. **X. Li**, S. Mabu, H. Zhou, K. Shimada, and K. Hirasawa, "Analysis of various interestingness measures in class association rule mining," SICE Journal of Control, Measurement, and System Integration. Vol. 4, No. 4, pp. 295-304, July, 2011.

6. **X. Li**, S. Mabu, and K. Hirasawa, "Towards the maintenance of population diversity: A hybrid probabilistic model building genetic network programming," Transaction of the Japanese Society for Evolutionary Computation, Vol. 1, No. 1, pp. 89-101, December, 2010.

7. **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," J. Advanced Computational Intelligence and Intelligent Informatics. Vol. 14, No. 5, pp. 497-509, July, 2010.

Conference papers

1. **X. Li**, B. Li, S. Mabu, and K. Hirasawa, "A continuous estimation of distribution algorithm by evolving graph structures using reinforcement learning," In Proc. of the IEEE Congress on Evolutionary Computation (CEC 2012), pp. 2097-2104, Brisbane, Australia, June, 2012.

2. B. Li, **X. Li**, S. Mabu, and K. Hirasawa, "Towards automatic discovery and reuse of subroutines in variable size genetic network programming," In Proc. of the IEEE Congress on Evolutionary Computation (CEC 2012), pp. 485-492, Brisbane, Australia, June, 2012.

3. B. Li, **X. Li**, S. Mabu, and K. Hirasawa, "Analysis of crossover rate in variable size genetic network programming with binomial distribution," In Proc. of the SICE Annual Conference (SICE 2011), pp. 155-160, Tokyo, Japan, September, 2011.

4. **X. Li**, S. Mabu, and K. Hirasawa, "Use of infeasible individuals in probabilistic model building genetic network programming," In Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2011), pp. 601-608, Dublin, Ireland, July, 2011.

5. **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. of the IEEE Congress on Evolutionary Computation (CEC 2011), pp. 37-44, New Orleans, USA, June, 2011.

6. B. Li, **X. Li**, S. Mabu, and Kotaro Hirasawa, "Variable size genetic network programming with binomial distribution," In Proc. of the IEEE Congress on Evolutionary Computation (CEC 2011), pp. 972-979, New Orleans, USA, June, 2011.

7. **X. Li**, S. Mabu, M. K. Mainali, and K. Hirasawa, "Probabilistic model building genetic network programming using multiple probability vectors," In Proc. of the IEEE TENCON International Conference (TENCON 2010), pp. 1398-1403, Fukuoka, Japan, November, 2010.

8. M. K. Mainali, S. Mabu, **X. Li**, and K. Hirasawa, "Optimal route planning with restrictions for car navigation systems," In Proc. of the IEEE International Conference on Systems, Man and Cybernetics (SMC 2010), pp. 393-397, Istanbul, Turkey, October, 2010.

9. **X. Li**, S. Mabu, H. Zhou, K. Shimada, and K. Hirasawa, "Analysis of various interestingness measures in classification rule mining for traffic prediction," In Proc. of the SICE International Annual Conference (SICE 2010), pp. 1969-1974, Taipei, Taiwan, August, 2010.

10. Y. Zhang, **X. Li**, Y. Yang, S. Mabu, Y. Jin, and K. Hirasawa, "Functionally distributed systems using parallel genetic network programming," In Proc. of the SICE International Annual Conference (SICE 2010), pp. 2626-2630, Taipei, Taiwan, August, 2010.

11. **X. Li**, S. Mabu, H. Zhou, K. Shimada, and Kotaro Hirasawa, "Genetic network programming with estimation of distribution algorithms for class association rule mining in traffic prediction," In Proc. of the IEEE Congress on Evolutionary Computation (CEC 2010), pp. 2673-2680, Barcelona, Spain, July, 2010.

12. H. Zhou, S. Mabu, **X. Li**, K. Shimada, and K. Hirasawa, "Generalized rule extraction and traffic prediction in the optimal route search," In Proc. of the

IEEE Congress on Evolutionary Computation (CEC 2010), pp. 2625-2632, Barcelona, Spain, July, 2010.

13. **X. Li**, S. Mabu, H. Zhou, K. Shimada, and K. Hirasawa, "Genetic network programming with estimation of distribution algorithms and its application to association rule mining for traffic prediction," In Proc. of the ICROS-SICE International Joint Conference, pp. 3457-3462, Fukuoka, Japan, August, 2009.

14. H. Zhou, S. Mabu, M. K. Mainali, **X. Li**, K. Shimada, and K. Hirasawa, "Generalized association rule mining with multi-branches full-paths and its application to traffic volume prediction," In Proc. of the ICROS-SICE International Joint Conference, pp. 147-152, Fukuoka, Japan, August, 2009.