Swarm and Evolutionary Computation ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Regular Paper

Improved sampling using loopy belief propagation for probabilistic model building genetic programming

Hiroyuki Sato [a,*], Yoshihiko Hasegawa [b], Danushka Bollegala [c], Hitoshi Iba [b]

[a] The Graduate School of Engineering, The University of Tokyo, Japan
[b] The Graduate School of Information Science and Technology, The University of Tokyo, Japan
[c] Department of Computer Science, The University of Liverpool, United Kingdom

ARTICLE INFO

Article history:
Received 26 February 2013
Received in revised form
2 January 2015
Accepted 20 February 2015

Keywords:
Genetic programming
Estimation of distribution algorithms
Loopy belief propagation
Probabilistic model building GP

ABSTRACT

In recent years, probabilistic model building genetic programming (PMBGP) for program optimization has attracted considerable interest. PMBGPs generally use probabilistic logic sampling (PLS) to generate new individuals. However, the generation of the most probable solutions (MPSs), i.e., solutions with the highest probability, is not guaranteed. In the present paper, we introduce loopy belief propagation (LBP) for PMBGPs to generate MPSs during the sampling process. We selected program optimization with linkage estimation (POLE) as the foundation of our approach and we refer to our proposed method as POLE-BP. We apply POLE-BP and existing methods to three benchmark problems to investigate the effectiveness of LBP in the context of PMBGPs, and we describe detailed examinations of the behaviors of LBP. We find that POLE-BP shows better search performance with some problems because LBP boosts the generation of building blocks.

1. Introduction

In the present paper, we introduce loopy belief propagation (LBP) in probabilistic model building GPs (PMBGPs) in order to generate the most probable solutions in sampling process. We call our novel method as POLE-BP.

Estimation of distribution algorithms (EDAs) are promising evolutionary algorithms and attract much attention from a lot of practical fields. EDAs optimize solution candidates represented by one dimensional arrays as well as Genetic Algorithms (GAs). Although EDA and GA employ the same chromosome representation, EDAs are different from GAs in the sense that EDAs generate new individuals by estimation of probabilistic models and sampling, whereas GAs generate them using genetic operators. EDAs can solve deceptive problems more efficiently than GAs by estimating dependencies between loci [27], which is one of the notable features of EDAs. Because of their effectiveness, many EDAs have been devised by incorporating many distinct statistical and machine learning approaches. Recently, EDAs using loopy belief propagation (LBP) as sampling were proposed in order to improve the sampling process [24,21]. LBP approximately infers marginal and the highest joint probabilities with configurations, and has been applied to a wide range of real world problems [7,6]. In EDAs, the individual with the highest joint probability in learned probabilistic models describes the models most and is often called as most probable solution (MPS). MPS is the individual which most reflects the learned models, and generation of it is important to take advantage of the models efficiently. However, traditional sampling methods used in EDAs, e.g. probabilistic logic sampling (PLS) [15] and Gibbs sampling, do not always generate MPS, and EDAs using only those samplings cannot make the best use of the models. In order to solve this problem, [24,21] generate MPS by LBP in addition to traditional sampling and showed better search performance than existing methods using only traditional samplings (PLS or Gibbs sampling) in benchmark problems.

The estimation of distribution concept employed in EDAs has been applied to the optimization of tree structures, which is traditionally addressed using GP. GP optimizes tree structures using operators, such as crossover and mutation, as well as GA. Numerous improved genetic operators have been proposed because it is difficult to deal with tree structures using only these simple operators. EDAs for tree structures are often called as Genetic Programming-EDAs (GP-EDAs) [11] or Probabilistic Model Building GPs (PMBGPs) [32], and the present paper adopts the latter abbreviation throughout the paper. PMBGPs are broadly classified into two types. One type uses probabilistic context free grammar (PCFG) to represent distributions of promising solutions and learns production rule probabilities. The other type is a prototype tree based method, which converts trees to one dimensional arrays and applies EDAs to them. From the viewpoint of probabilistic models, the prototype tree-based method is essentially equivalent to EDAs and hence it can easily incorporate techniques devised in the field of EDA.

We propose POLE-BP [34], the novel prototype tree-based PMBGP with LBP. POLE-BP generates MPS at every generation in addition to normal samplings (i.e. PLS) and makes the optimal use of the learned

* Corresponding author. Tel.: +81 3 5841 6751.
E-mail addresses: sato@iba.t.u-tokyo.ac.jp, umkn.smag@gmail.com (H. Sato).

http://dx.doi.org/10.1016/j.swevo.2015.02.002
2210-6502/

probabilistic model. We compare our proposed method against existing methods on three benchmark problems: the problem with no dependencies between nodes (MAX problem), the deceptive problem (Deceptive MAX problem) and the problem with dependencies between nodes (Royal Tree Problem). From results of the experiments, we show that the proposed method competes with the existing method in the deceptive problem and beats the existing method in the problems with no deceptiveness from the point of the number of fitness evaluations to get an optimum solution. Moreover, we investigate behaviors of LBP in the context of PMBGP by observing fitness values and structures generated by LBP, and show reasons why the proposed method does not exhibit search performance improvement in deceptive problems whereas it does in other benchmark problems.

The present paper extends our prior work [34] by studying the effectiveness of LBP in detail. The remainder of the paper is organized as follows. Section 2 introduces related work. Section 3 explains details of the proposed method. Section 4 presents the experimental condition and results, which is followed by the discussion in Section 5. Finally Section 6 concludes the paper.

2. Related work

We introduce existing PMBGPs and methods using loopy belief propagation as sampling in this section.

2.1. PMBGP: Probabilistic Model Building GP

PMBGPs are extensions of EDAs for tree structures that generate the next population by estimating probabilistic distributions from better individuals and sampling individuals from them. For recent surveys on EDAs, interested reader is directed to [18,14]. PMBGPs are superior to GP in the sense that PMBGPs can search for solutions with a smaller number of fitness evaluations and they can solve problems that conventional GP cannot [10]. Two types of methods are known in the field of PMBGPs.

- Prototype tree-based method.
- PCFG-based method.

The prototype tree-based method translates trees into one dimensional arrays and applies conventional EDAs to them. By contrast, the PCFG-based method expresses individuals with derivation trees and learns their production rules as well as their parameters. Because derivation trees naturally derive functions and programs, PCFG-based methods can estimate position-independent substructures, and so many approaches have been proposed based on these approaches [2,29,30,35,36,13]. However, the prototype tree-based methods have advantages over PCFG-based methods because they can readily utilize existing EDAs. Furthermore, the prototype tree-based methods are computationally reasonable even when we consider the dependencies between nodes, whereas PCFG-based methods that consider them are very computationally intensive. In addition, the probabilistic distribution concept is also applied to genetic network programming (GNP) [16,23], which expresses programs using directed graphs as chromosomes [19,20].

2.1.1. Prototype tree-based method

Prototype tree-based methods regard all individuals as α-ary perfect trees, where α is the maximum number of arguments among function nodes, and translate them to one dimensional arrays, and then tree structures are optimized by applying EDAs to the translated arrays. Prototype tree-based methods attract considerable attention because they can easily exploit existing EDAs.

The first prototype tree-based method is probabilistic incremental program evolution (PIPE) [31], which is an extension of population-based incremental learning (PBIL) [1] for tree structures. PIPE is weak against problems with dependencies between nodes because PIPE assumes that each node is independent of the others. Estimation of distribution programming (EDP) [38] estimates dependencies between nodes, using Bayesian networks. However, EDP is weaker than other methods with structural learning because EDP estimates only fixed parent–child relationships in tree structures. Extended compact GP (ECGP) [33] bases on extended compact GA (ECGA) [9]. ECGP estimates the multivariate dependencies among nodes using the minimum description length (MDL) principle but ECGP cannot estimate building blocks with practical size because of the large number of symbols in GP. Bayesian optimization algorithm (BOA) programming (BOAP) [22] is an application of BOA to tree structures and it uses a zigzag tree as chromosome. Program optimization with linkage estimation (POLE) [12] also estimates the multivariate dependencies among nodes using Bayesian networks.

The conventional prototype tree-based methods listed above employ PLS as sampling. Therefore, those methods waste a part of learning because PLS does not guarantee that MPSs, which best reflect the learned probabilistic models, will be generated at each generation. In order to overcome this deficiency, we propose an efficient sampling method to generate MPSs at each generation.

2.2. EDAs and PMBGPs with loopy belief propagation

In a field of EDAs, several algorithms using LBP have been hitherto proposed. Those methods focus on sampling the solution with the highest joint probability. It is difficult to calculate joint probabilities directly for graphical models with complex graph structure, which appear frequently in EDA and PMBGP. However, we can calculate approximate joint probability from approximate *local* joint probability easily for those complex graphs. The approximate *local* joint probability is often called as message. The key idea of LBP is message passing, iteration of message updating for getting more accurate approximation of joint probabilities.

Ref. [24] uses EBNA [5] as a foundation and simply applies LBP to generate one individual (MPS) whereas the rest of individuals are generated by PLS. Ref. [24] applies normal EBNA and EBNA-LBP to the Ising problem and shows that LBP boosts the best fitness value in the latter part of the search. Ref. [21] proposes Loopy Substructural Local Search (Loopy SLS), which employs local fitness as the values of factors and all possible individuals are carried over to the next generation if message passing does not converge. Ref. [21] uses BOA [27] as a base method and applies normal BOA, BOA with standard LBP (the same as in [24]) and BOA with Loopy SLS to the trap function. Ref. [21] concludes that Loopy SLS is better than standard LBP when population size is large.

We have already proposed the application of LBP in the contexts of PMBGPs [34], however, the prior work has only shown the effectiveness of LBP in view of the number of fitness evaluations and has not studied how LBP works. In order to discuss roles of LBP in the search process of PMBGP, the present paper applies POLE-BP to three benchmark tests and analyzes not only the number of fitness evaluations but also fitness and tree structures generated by LBP. One of the main contributions in the present paper is the detailed analysis of fitness and tree structures that has not been examined in our prior work [34].

3. The proposed method: POLE-BP

We briefly describe POLE-BP [34] in this section. POLE-BP is the first approach combining PMBGP and LBP. POLE-BP introduces LBP to the sampling process of Program Optimization with Linkage

Please cite this article as: H. Sato, et al., Improved sampling using loopy belief propagation for probabilistic model building genetic programming, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.02.002

estimation (POLE) [12] and guarantees that population includes MPS, which is an individual having the highest joint probability, in each generation.

Prototype tree-based PMBGPs use more symbols than EDAs, which causes the following problems:

1. require more population size and the number of evaluations;
2. difficult to generate MPS by conventional sampling methods because the number of candidate of individual increases extremely.

Using expanded parse tree (EPT) [37], POLE solves the first problem directly and the second problem indirectly. By pushing terminal nodes on trunks to leaves, EPT reduces the number of candidates of the trunk nodes. The reduced number of candidate makes model building easier and requires lesser population size and the lesser number of evaluations. POLE solves the first problem directly in this manner.

In addition, POLE solves the second problem indirectly as follows. The lesser number of candidate of the trunk nodes leads to lesser number of candidate of individuals. This reduced number of candidate of individuals alleviates the difficulty of sampling MPS with conventional sampling algorithms such as PLS or Gibbs sampling.

However, the second problem is not solved completely because the conventional sampling algorithms do not necessarily generate MPS. We focus on solving the second problem completely and improve the search performance using LBP, which is an algorithm that guarantees PMBGPs to generate MPS.

First, we introduce the basic concepts behind our approach and then describe the proposed algorithm.

3.1. POLE: program optimization with linkage estimation

POLE [12] is a conventional PMBGP which employs EPT [37] as its chromosome to solve problems caused by the large number of symbols. Using a special function node $L(x, y, \ldots) = x$, EPT pushes terminal nodes on trunk to leaves. Let $\mathbb{F}$ and $\mathbb{T}$ be sets of function and terminal nodes, respectively. This operation reduces the number of symbols on trunk from $|\mathbb{F} \cup \mathbb{T}|$ to $|\mathbb{F} \cup \{L\}|$ and makes learning of Bayesian networks easier. We show the difference of normal tree and EPT in Figs. 1 and 2, respectively.

3.2. Loopy belief propagation (loopy max-sum)

Belief Propagation (BP) [26] is an inference algorithm for tree structured graphical models and has some variants depending on objectives. For example, max-sum is the instance of BP and calculates

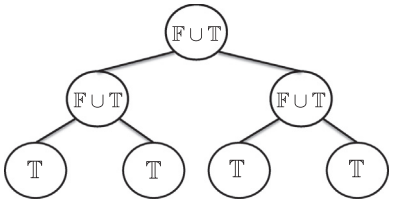

Fig. 1. Symbols on normal GP tree

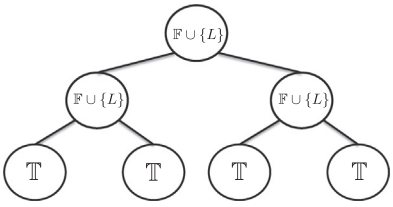

Fig. 2. Symbols on EPT

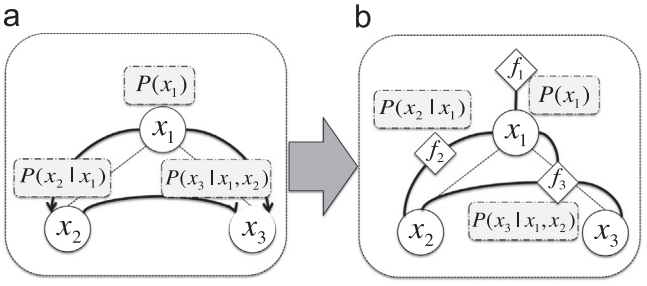

Fig. 3. (a) Bayesian network and (b) Factor graph. Dotted lines represent tree structures. Circles are variable nodes and lozenges are factor nodes. (b) is equivalent to (a).

the highest joint probability of those models effectively. LBP is the application of BP to graphical models with arbitrary loopy graph structure and approximately and speedily infers MPS or marginal probabilities. LBP works successfully in many real world applications [7,6], including EDAs. In the present paper, we use the term *loopy max-sum* to refer to the application of max-sum algorithm to loopy graphs. Loopy max-sum repeatedly updates messages, which are locally calculated joint probabilities, and finally approximately generates MPS from the messages.

In order to apply loopy max-sum to Bayesian networks, we must transform them to equivalent Factor graphs, which are a type of graphical models. It is possible to transform Bayesian networks to equivalent Factor graphs. The transformation we used in our proposed method is illustrated in Fig. 3. In Bayesian networks, variable nodes with no parents have prior probabilities, and directed edges represent conditional probabilities. On the other hand, in Factor graphs, factor nodes represent prior and conditional probabilities, and undirected edges represent only the connectivity.

We apply loopy max-sum to the transformed factor graph. Let x_i be ith symbol in prototype trees, and θ and G be a set of parameters and Bayesian networks, respectively. Then individuals and MPS are represented by

$$\mathbf{x} = (x_1, x_2, \ldots) \tag{1}$$

$$\mathbf{x}_{MPS} = \underset{\mathbf{x}}{\operatorname{argmax}}\, P(\mathbf{x}; \theta, G). \tag{2}$$

Let $\mu_{f \to x}$ and $\mu_{x \to f}$ be messages from factor nodes to variable nodes and those from variable nodes to factor nodes, respectively, and ne(X) and ne$(X) \setminus Y$ be the set of adjacent nodes to X and the set of adjacent nodes to X except Y, respectively. $f(x_0, \ldots, x_n)$ represents the value of factor f when its adjacent variable nodes are $x_0, \ldots, x_n$. The details of loopy max-sum are described next.

Step 1 : Initialization.
All messages are initialized to 0.

Step 2 : Message passing.Message passing is repeatedly executed, using Eqs. (3)–(6). Eqs. (3) and (4) represent messages from leaf nodes, and Eqs. (5) and (6) represent messages from nodes except leafs:

$$\mu_{f \to x}(x) = \ln f(x), \tag{3}$$

$$\mu_{x \to f}(x) = 0, \tag{4}$$

$$\mu_{f \to x}(x) = \max_{x_1, \ldots, x_N \in ne(f) \setminus x} \left[\ln f(x, x_1, \ldots, x_N) + \sum_{m \in ne(f_s) \setminus x} \mu_{x_m \to f}(x_m) \right], \tag{5}$$

Please cite this article as: H. Sato, et al., Improved sampling using loopy belief propagation for probabilistic model building genetic programming, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.02.002

$$\mu_{x\to f}(x) = \alpha_{xf} + \sum_{l \in ne(x)\setminus f} \mu_{f_l \to x}(x), \tag{6}$$

where α_{xf} is a normalization constant chosen such that $\sum_{x_n} \mu_{x\to f}(x) = 0$.

Step 3 : Check termination criteria.
If termination criterion is satisfied, message passing terminates. Otherwise, message passing continues.

Step 4 : Get the most probable solution (MPS).
Get the MPS, using Eqs. (7) and (8). x_i^{max} is the most probable instance of an ith and calculated from messages around an ith variable node:

$$x_i^{max} = \underset{x_i}{\operatorname{argmax}} \left[\sum_{s \in ne(x_i)} \mu_{f_s \to x_i}(x_i) \right] \tag{7}$$

$$\mathbf{x}_{MPS} \approx \mathbf{x}^{max} = (x_1^{max}, x_2^{max}, \cdots) \tag{8}$$

3.3. Algorithm

POLE-BP introduces the simplest application of LBP [24] in EDA to PMBGP. As well as [24], POLE-BP transforms estimated Bayesian networks to equivalent Factor graphs and creates MPS by running LBP on them. In addition, although [24] uses max-product, POLE-BP uses max-sum because PMBGP deals with problems accompanied by more symbols than EDA, which results in a underflow problem during the message passing. The details of POLE-BP is described in Algorithm 1.

Algorithm 1. POLE-BP.

```
g ← 0
P_g ← Initialize M individuals by GROW
Evaluate P_g
while terminate criterion is False
    g ← g + 1
    S_g ← Select N(N ≤ M) superior individuals by truncate selection
    B_g ← Construct Bayesian networks from S_g by K2 algorithm
    P_g ← Sampling M − 1 individuals from B_g by PLS
    F_g ← Translate B_g to equivalent Factor graph
    P_g ← Generate MPS by LBP on F_g
    Evaluate P_g
end while
```

POLE-BP estimates dependencies between varied nodes using Bayesian networks which are learned from scratch at each generation. The process of sampling and probability maximizing is almost same as POLE except generating MPS by LBP.

It is time consuming to calculate posterior probability for complex graph structures. Therefore, we calculate BIC score instead of posterior probability and search the graph structure which maximizes BIC score. Because searching graph structure with the highest BIC score is NP hard problem, we employ K2 algorithm [3] which is one of the greedy search algorithms. Let d, S and r be the number of data, the number of nodes and the number of symbols for a node, respectively. As described in [3], time complexity of K2 algorithm is $O(dS^4r)$ in the worst case. In POLE-BP, d is N and the upper bound of r is $|\mathbb{T}|$ because terminal nodes have more variation than functional nodes usually. Therefore, the complexity of Bayesian networks construction is $O(NS^4|\mathbb{T}|)$.

The main sampling method is PLS, which is standard sampling method for Bayesian networks. PLS determines the root symbol firstly, and sample symbols of descendants recursively. PLS requires $O(MS)$.

After PLS, POLE-BP samples MPS using LBP. As we illustrate in Fig. 3, the number of nodes in factor graph is smaller than $O(S^2)$. Message passing for a node considers only adjacent nodes (see Eqs. (5) and (6)) and needs smaller calculation than $O(S^2)$. If we update message C times for each node, time complexity of LBP is $O(CS^4)$ multiplying above quantities in the worst case.

Although the difference between POLE and POLE-BP is the way to generate only one individual among several hundreds or thousands individuals, we show that POLE-BP has much higher search performance than POLE on benchmark problems in Section 4.

Let t_{eval} be time complexity of fitness evaluation. Adding time complexity of each step, time complexity of whole POLE-BP per generation is $O(NS^4|\mathbb{T}| + t_{eval})$ because C is much smaller than $N|\mathbb{T}|$ in practice. t_{eval} is problem dependent, and we cannot describe generally. We believe that fitness evaluation becomes dominant considering real world applications which require complex fitness calculation.

4. Experiments

In order to evaluate the search performance of the proposed method, we apply POLE-BP, POLE and simple GP (SGP) to three benchmark problems, MAX problem, deceptive MAX (DMAX) problem and royal tree problem and compare their performances. Common parameters in POLE-BP and POLE are described in Table 1, and parameters of SGP are described in Table 2. In POLE-BP, message passing schedule of loopy max-sum is that messages are sent from all factor nodes and variable nodes by turns. A termination criterion is that all nodes send messages 100 times.

4.1. Experimental settings

This section describes two types of experiments: one describes the effect of LBP in view of the number of fitness evaluations (Section 4.1.1), and the other investigates how LBP works and analyzes observed results in detail (Section 4.1.2).

4.1.1. Experiments to show the effects of LBP

First, we compare the average number of fitness evaluations required to obtain an optimum solution by the proposed and existing methods. Because population size M significantly influences the search performance, the optimal M is different for various problems and algorithms. Therefore, the population size of each method is determined in the following manner. We start from $M = 100$, and increase the population size by $\sqrt[5]{10}$ times. For each population size, we execute 20 runs. If the algorithm obtains the optimum solution 20 times from the 20 runs, we stop increasing the population size and calculate the average number

Table 1
Common parameters in POLE-BP and POLE.

Parameters	Meaning	Value
M	Population size	–
P_s	Selection rate	0.1(MAX, DMAX) 0.2(Royal Tree)
P_e	Elite rate	0.005
P_F	Function selection rate while initialization	0.8(MAX,DMAX) 0.9(Royal Tree)

Please cite this article as: H. Sato, et al., Improved sampling using loopy belief propagation for probabilistic model building genetic programming, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.02.002

Table 2
Parameters of SGP.

Parameters	Meaning	Value
M	Population size	–
P_e	Elite rate	0.005
P_c	Crossover rate	0.995
P_m	Mutation rate	0

of fitness evaluations F_{avr}. This method of determining the population size has previously been used in [5,27,12]. Because F_{avr} has a large variance, we operated the above procedure 20 times and calculated average of F_{avr} and performed t-test (Welch, two-tailed) for each experiment. We calculate the P-value for the obtained data (e.g., the average of POLE-BP and the average of POLE) to verify any statistically significant differences between the result obtained with the different approaches. All algorithms are terminated when they converge to an optimum solution. Furthermore, we also adopt another termination criterion. Because POLE-BP and POLE tend to converge faster than SGP, these algorithms terminate when the best fitness value at each generation is not improved for 10 consecutive generations. In contrast, SGP terminates when the fitness values are not improved from generation g to generation $2g$ ($g > 10$). This method of calculating P-value and terminate criterion has previously been used in [12].

4.1.2. Experiments to examine function and operation of LBP

Along with the comparison of the number of fitness evaluations, we also study functionalities of LBP by observing behaviors of POLE-BP, which can make the effectiveness of LBP in the context of PMBGP clear. Because mathematical analysis of LBP in arbitrary loopy graphs used in PMBGP is difficult, we empirically analyze individuals generated by LBP. Using M defined by the above procedure, we perform additional 100 runs of POLE and POLE-BP to measure the following quantities:

- the average fitness of the best individual at each generation;
- the frequencies that LBP generates the best individual and that individuals generated by LBP are used for construction of Bayesian networks for the next generation (individuals reaching better $M{*}P_s$);
- building-block-satisfying-rate at each generation. We define it as ((the number of building blocks of the best individual at each generation)/(the number of building blocks which an optimum solution contains)).

We hypothesize that LBP boosts the search performance for building blocks and fitness during searching because LBP generates the individual reflecting Bayesian networks that estimate building blocks in the middle of learning. Therefore, we consider that these measurements can clarify behaviors of LBP on POLE-BP.

4.2. MAX problem

MAX problem [8,17] is designed to investigate the mechanism of crossover in GP, and is widely used as a benchmark test for PMBGPs [36,38,12]. The purpose of the MAX problem is to search a function that returns the largest real value within the limits of a maximum tree depth. In this problem, three symbols described in Eq. (9) are used:

$$\mathbb{F} = \{+, *\}, \quad \mathbb{T} = \{0.5\} \tag{9}$$

An optimum solution can be obtained by the following procedure. First, create the value "2" using four "0.5" symbols and three "+" symbols. Then, multiply the created "2"s using "*" symbols. $2^{2^{D_p}-3}$ represents the optimum value for a given maximum depth D_p.

4.2.1. Result and analysis

Tables 3 and 4 respectively show the average and standard deviation, and the reduction rate of the average number of evaluations ((the average number of fitness evaluations of POLE-−that of POLE-BP)/that of POLE * 100), over 20 trails. Fig. 4 visualizes Tables 3 and 4, where tree size is the number of nodes contained in the optimum structure ($2^{D_p}-1$ in the MAX problem). Furthermore, we carried t-test to see the statistical significance of the results (Table 5). According to Table 5, the P-value for POLE-BP and POLE is smaller than 1% in $D_p = 6, 7, 8$, and the difference between POLE and POLE-BP for $D_p = 6, 7, 8$ is statistically significant at 1% significant level. In $D_p = 6$, 7, 8, LBP reduces the number of fitness evaluations by 27.4% on average.

In $D_p = 8$, POLE-BP and POLE obtain the optimum solution 20 times from the 20 runs with $M = 630$. On this condition, we run POLE-BP and POLE additional 100 runs and analyse how LBP works. Figs. 5 and 6 visualize the average fitness of the best individual and the number of better individuals generated by LBP, building-block-satisfying-rate at each generation and the number of better individuals generated by LBP, respectively. According to Fig. 5, LBP generates individuals with high fitness values more frequently in POLE-BP. Overall, the average fitness of POLE-BP is better than that of POLE. Moreover, the more the LBP generates better individuals, the bigger the difference of fitness is. As can be seen from Fig. 6, POLE-BP generates individuals containing more building blocks at earlier generations than POLE. First, LBP boosts the creation of shallow building blocks with depth 3 or 4. Bayesian networks suitable for such shallow building blocks are learned until the 5th generation. After 6th generation, the number of better individuals found by LBP decreases temporarily. During this interval, it is considered that POLE-BP learned Bayesian networks for deeper building blocks with depth 5 or 6 to increase fitness values. The number of better individuals found using LBP dropped again at 14th generation, indicating that Bayesian networks capable of generating building blocks with depth 5 or 6 are refined to generate those with depth 7 or optimum solutions. Putting Figs. 4–6 together, the superior performance of POLE-BP is achieved by LBP first boosting the generation of building blocks. Once the building blocks are increased, these building blocks improve average fitness values, which eventually leads to the reduction of the number of fitness evaluations to obtain an optimum.

4.3. Deceptive MAX (DMAX) problem

The deceptive MAX problem (DMAX problem) [12] is a deceptive extension of the MAX problem using complex values as terminal nodes. The DMAX problem has the same objective as the MAX problem: to find functions which return the largest real value under the limitation of a maximum tree depth D_p. Symbols used for this

Table 3
The number of fitness evaluations for the MAX problem.

D_p: Maximum tree depth of the problem	$D_p = 5$	$D_p = 6$	$D_p = 7$	$D_p = 8$
POLE-BP				
Average	380	1272	3475	10,238
St. dev.	(87)	(310)	(918)	(1567)
POLE				
Average	435	1648	4900	14,679
St. dev.	(85)	(263)	(307)	(1353)
SGP				
Average	1416	3208	8513	59,950
St. dev.	(249)	(442)	(1136)	(4801)

Please cite this article as: H. Sato, et al., Improved sampling using loopy belief propagation for probabilistic model building genetic programming, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.02.002

Table 4
The reduction rate of the number of fitness evaluations by LBP in MAX problem.

D_p: Maximum tree depth of the problem	$D_p=5$	$D_p=6$	$D_p=7$	$D_p=8$
Reduction rate (%)	12.64	22.82	29.08	30.26

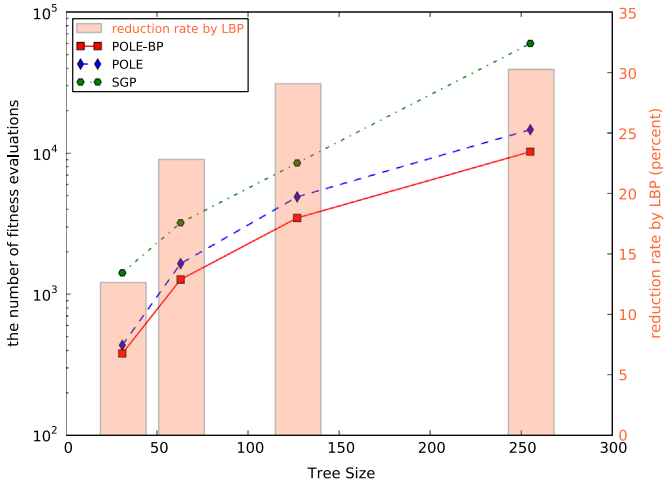

Fig. 4. The number of evaluations required for the MAX problem.

Table 5
t-test. The values represent *P*-values for the MAX problem.

D_p: Maximum tree depth of the problem	$D_p=5$	$D_p=6$	$D_p=7$	$D_p=8$
POLE-BP vs POLE	5.02E−2	2.0E−4	1.02E−06	Underflow
POLE-BP vs SGP	3.33E−15	Underflow	Underflow	Underflow

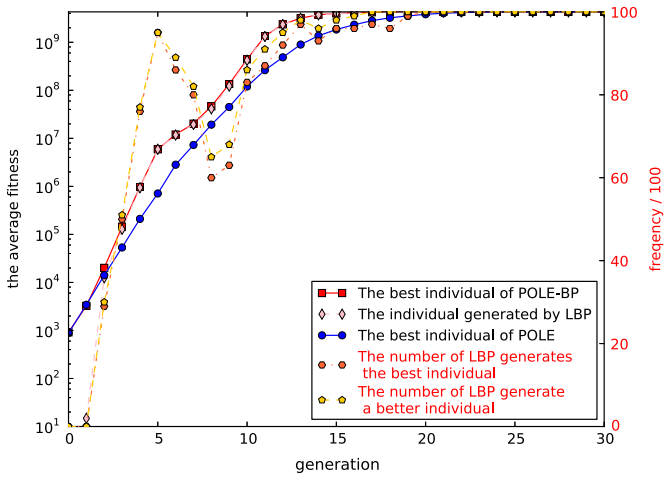

Fig. 5. The average fitness of the best individual at each generation in MAX problem ($M = 630$, $D_p = 8$).

experiments are described in Eqs. (10)–(13):

$$\mathbb{F} = \{\text{add}_5, \text{multiply}_5\} \quad \mathbb{T} = \{\lambda_3, 0.95\} \tag{10}$$

$$\text{add}_5(a_0, \ldots, a_4) = \sum_{i=0}^{4} a_i \tag{11}$$

$$\text{multiply}_5(a_0, \ldots, a_4) = \prod_{i=0}^{4} a_i \tag{12}$$

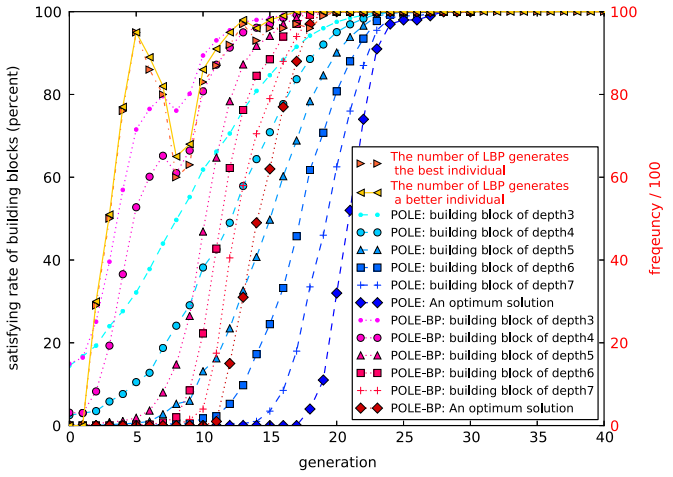

Fig. 6. Building block satisfying rate of the best individual at each generation in MAX problem ($M = 630$, $D_p = 8$).

Table 6
The number of fitness evaluations for the DMAX problem.

D_p: Maximum tree depth of the problem	$D_p=3$	$D_p=4$
POLE-BP		
Average	1539	120,708
St. dev.	(276)	(10,569)
POLE		
Average	1517	122,031
St. dev.	(283)	(5819)
SGP		
Average	34,875	–
St. dev.	(4533)	(–)

$$\lambda_3 = \left(-\frac{1}{2} + \frac{i\sqrt{3}}{2}\right) \tag{13}$$

Let us consider the optimum value for the DMAX problem with $D_p = 3$. In order to get the maximum absolute value, first, create $5\lambda_3$, using five λ_3 and add_5. Then, create $(5\lambda_3)^5 = 5^5\lambda_3^2$, using $5\lambda_3$ and multiply_5. However, $\text{Re}(5^5\lambda_3^2)$ is negative, and $5^5\lambda_3^2$ is not a optimum solution. Therefore, substituting two $5*0.95$ for two $5\lambda_3$ makes the optimum value, $(5\lambda_3)^3(0.95*5)^2 = 2820.3125$. We can find that the optimum value with $D_p = 4$ is $(5\lambda_3)^{24}(0.95*5) = 2.83*10^{17}$ in a similar way.

4.3.1. Results and analysis

Table 6 presents the average number of fitness evaluations and standard deviations in 20 trials. Note that SGP could not obtain an optimum solution at $D_p = 4$. Table 7 shows the reduction rate of the average number of evaluations by LBP, which is visualized in Fig. 7. We additionally carried *t*-test to confirm the statistical significance of the results (Table 8). Because each *P*-value between POLE-BP and POLE is larger than 1 %, the performance of POLE-BP may not surpass POLE.

In D_p=4, POLE-BP and POLE obtain the optimum solution 20 times from the 20 runs with M=6300. Under this condition, we run POLE-BP and POLE additional 100 runs and analyze the behavior of LBP. Figs. 8 and 9 visualize the average fitness of the best individual and the number of better individuals generated by LBP, and building-block-satisfying-rate and the number of better individuals generated by LBP at each generation, respectively. The strong deceptiveness of DMAX problem is caused by rotation on a complex plane using function nodes, and consequently, estimation of function nodes is important. Therefore, Fig. 9 visualizes not only frequency of building blocks but also correct part of function

Please cite this article as: H. Sato, et al., Improved sampling using loopy belief propagation for probabilistic model building genetic programming, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.02.002

Table 7
The reduction rate of the number of fitness evaluations by LBP.

D_p: Maximum tree depth of the problem	$D_p=3$	$D_p=4$
Reduction rate (%)	−1.45	1.08

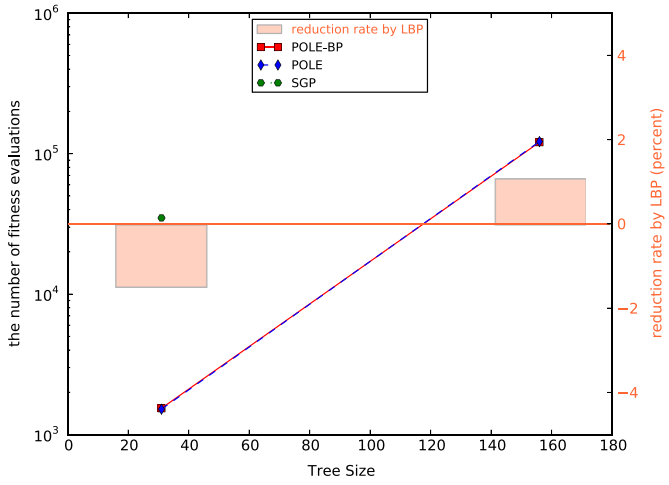

Fig. 7. The number of evaluations required for the DMAX problem.

Table 8
t-test. The values represent *P*-values for the DMAX problem.

D_p: Maximum tree depth of the problem	$D_p=3$	$D_p=4$
POLE-BP vs POLE	5.78E−1	2.75E−1
POLE-BP vs SGP	Underflow	–

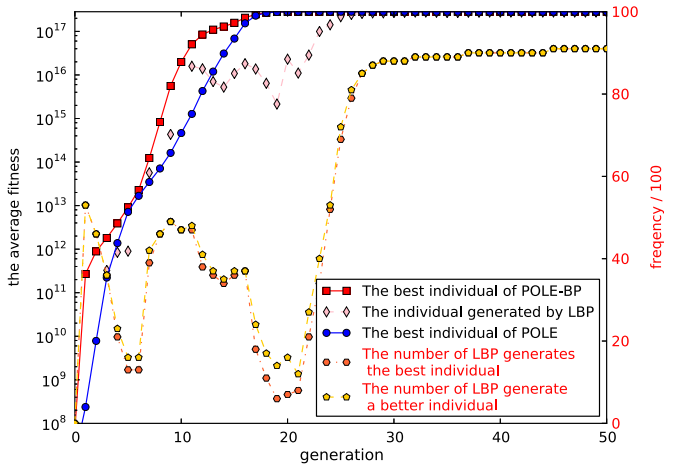

Fig. 8. The average fitness of the best individual at each generation in DMAX problem ($M = 6300$, $D_p = 4$).

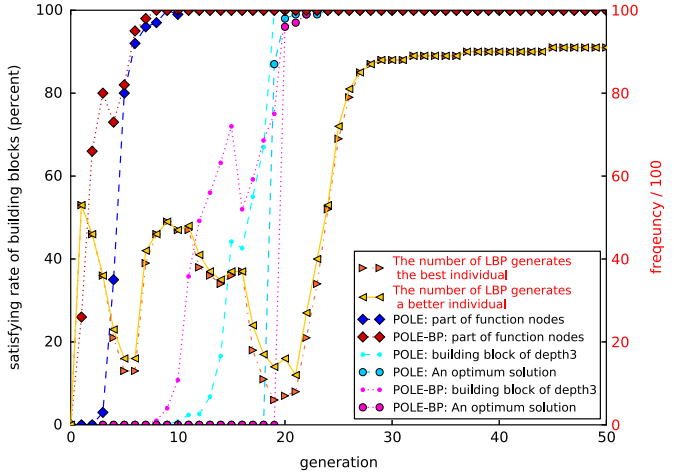

Fig. 9. Building block satisfying rate of the best individual at each generation in DMAX problem ($M = 6300$, $D_p = 4$).

nodes. From Fig. 8, fitness of the individual generated by LBP is not high. Although the average fitness of POLE-BP is temporarily higher than that of POLE during LBP generating relatively better individuals, there is no difference between generations when two methods finally obtain an optimum solution. As can be seen in Fig. 9, although POLE-BP generates the individual with more building blocks and/or correct function nodes at earlier generation than POLE, POLE catches up with POLE-BP in mid-course. For example, LBP boosted the creation of correct parts of function nodes until 4th generation, however, POLE got to the same level at 5th generation. Regarding building blocks with depth 3, although POLE-BP generated them more frequently from 10th to 15th generations , POLE became to create them as frequently as POLE-BP at 16th generation. Finally, POLE-BP obtained optimum solutions about 90 times at 18th generation whereas POLE did not at all. However, POLE succeeded to obtain optimum solutions at the next generation. Putting Figs. 7–9 together, although LBP works as local search and increases fitness temporarily by finding local optima, LBP does not reduce the number of fitness evaluations to obtain an optimum solution. In deceptive problems because the learned probabilistic model reflects the nature of problems that may lead algorithms to local optima, LBP of such a model is likely to generate locally optimal solutions. POLE-BP, the hybrid approach of POLE and LBP, have shown restrictive power in the deceptive problem, however, we want to note that PMBGPs are originally a powerful method in the deceptive problem.

4.4. Royal tree problem

The royal tree problem [28] is an extension of the royal road function [25] to GP. In the royal tree problem, the optimum structure is gotten by combining the building blocks. Symbols used in royal tree problem are described as

$$\mathbb{F} = \{A, B, C, D, E, F\}, \quad \mathbb{T} = \{x\}, \ x = 1 \tag{14}$$

The goal of the royal tree problem is to search a tree structure with the largest score within the limits of a maximum tree depth. An optimum solution is call as *Perfect Tree*, where each trunk node has the previous alphabet of children (e.g. children nodes of a function *D* are *C*). We describe the score function in Algorithm 2. We define that all function nodes take two arguments, so *n* at line 5 in Algorithm 2 is 2. childOf(*node*, *i*) returns the *i*th child of *node*, and correct at line 6 means *child* is *x* if *node* is *A*, or *child* is the previous alphabet if *node* is a function except *A*.

Algorithm 2. The score function of Royal tree Problem: score(*node*)

```
1:if node is x then
2:   return 1
3:else
4:   s←0
5:   for i=1 to n do
6:      child←childOf(node, i)
7:      if child is correct then
8:         if child is Perfect Tree then
9:            bonus←Full Bonus
```

```
10:       else
11:         bonus ← Partial Bonus
12:       end if
13:     else
14:       bonus ← Penalty
15:     end if
16:     s ← s + score(child)*bonus
17:   end for
18:   if node is Perfect Tree then
19:     s ← s*Complete Bonus
20:   end if
21:   return s
22:end if
```

We use Full Bonus = 2, Partial Bonus = 1, Penalty = 1/3, Complete Bonus = 2. Let maximum depth be D_p, and the fitness of the optimum solution is $2^{3(D_p-1)}$.

4.4.1. Results and analysis

Table 9 presents the average number of fitness evaluations and standard deviations in 20 trials. Table 10 shows the reduction rate of the average number of evaluations by LBP. Fig. 10 shows some of the data in Tables 9 and 10. Table 11 represents the results of the *t*-test, which indicates that POLE-BP is superior to POLE and SGP. According to Table 11, the *P*-value for POLE-BP and POLE is smaller than 1% in $D_p = 6, 7$, and the difference between POLE and POLE-BP for $D_p = 6, 7$ is statistically significant at 1% significance level. In $D_p = 6, 7$, LBP reduces the number of fitness evaluations by 35.5% on average.

In $D_p = 7$, POLE-BP and POLE obtain the optimum solution 20 times from the 20 runs with $M = 2500$. On this condition, we run POLE-BP and POLE additional 100 runs and analyze the behavior of LBP. Figs. 11 and 12 visualize the average fitness of the best individual and the number of better individuals generated by LBP, building-block-satisfying-rate and the number of better individuals at each generation, respectively. From Fig. 11, we see that LBP generates the best or better individuals, and the average fitness of POLE-BP is higher than that of POLE. A distinguishing point of this result is that LBP generates more better individuals but not the best individual compared to the cases in MAX and DMAX problems. According to Fig. 12, POLE-BP generates the individual with more building blocks at earlier generation than POLE. Until the 10th generation, LBP boosts the generation of shallow building blocks with depths 3, 4, and 5, and Bayesian networks suitable for generation of those shallow building blocks have been learned. After the 10th generation, the number of better individuals found by LBP decreased temporarily. The reason of the decrease is considered that Bayesian networks, capable of generating both deeper building blocks with depth 6 and an optimum solution, are learned during the decreasing interval. From Figs. 10–12, the superiority of POLE-BP against POLE in the royal tree problem can be accounted for by the same mechanism as in the MAX problem:

Table 9
The number of fitness evaluations for the Royal tree problem.

D_p: Maximum tree depth of the problem	$D_p = 4$	$D_p = 5$	$D_p = 6$	$D_p = 7$
POLE-BP				
Average	260	1352	14720	67250
St. dev.	(66)	(413)	(2844)	(3250)
POLE				
Average	290	1560	25375	94600
St. dev.	(54)	(469)	(1431)	(4432)
SGP				
Average	420	3000	29520	119800
St. dev.	(144)	(316)	(2115)	(6750)

Table 10
The reduction rate of the number of fitness evaluations by LBP in Royal Tree problem.

D_p: Maximum tree depth of the problem	$D_p = 4$	$D_p = 5$	$D_p = 6$	$D_p = 7$
Reduction rare (%)	10.3	13.3	42.0	28.9

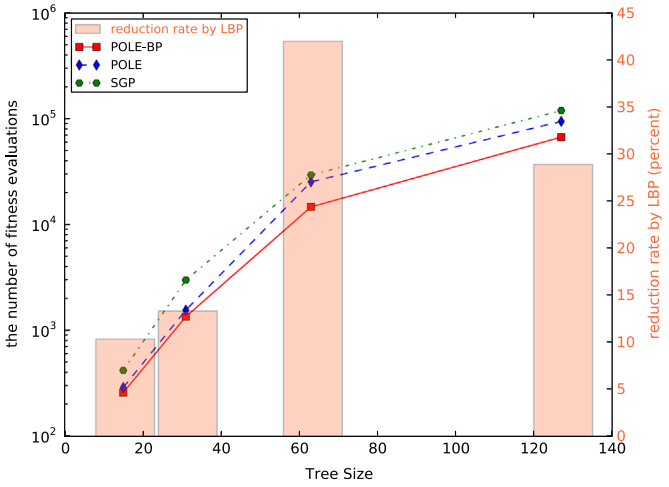

Fig. 10. The number of evaluations required for the Royal tree problem.

Table 11
t-test. The values represent *P*-values for the Royal tree problem

D_p: Maximum tree depth of the problem	$D_p = 4$	$D_p = 5$	$D_p = 6$	$D_p = 7$
POLE-BP vs POLE	1.24E−1	1.45E−1	6.88E−15	Underflow
POLE-BP vs SGP	1.11E−5	2.22E−16	Underflow	Underflow

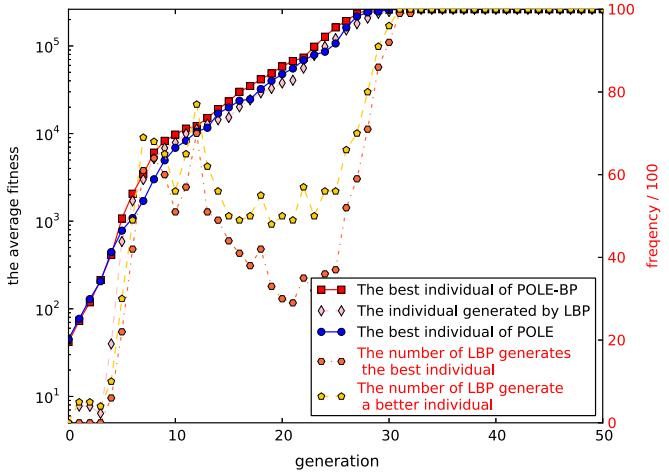

Fig. 11. The average fitness of the best individual at each generation in the Royal tree problem ($M = 2500$, $D_p = 7$).

boosting of building block generation due to LBP first improves the average fitness values, which result in the reduction of the number of evaluations to obtain an optimum solution.

5. Discussion

In the previous section, for respective problems, we displayed the reduction of the average number of fitness evaluations by LBP and the behaviors of LBP in a PMBGP. Up to this point, we summarize the

Please cite this article as: H. Sato, et al., Improved sampling using loopy belief propagation for probabilistic model building genetic programming, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.02.002

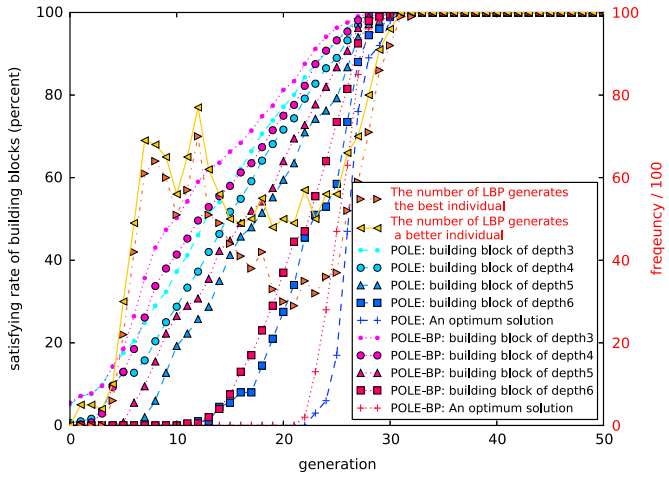

Fig. 12. Building block satisfying rate of the best individual at each generation in Royal tree problem ($M = 2500$, $D_p = 7$).

function of LBP in a PMBGP as the following: although LBP stimulates generation of building blocks, degree of the effectiveness differs dependent on the nature of problems. In the MAX problem, the best individual at each generation is apt to be generated by LBP. The MAX problem has no dependencies between nodes, and it is easy to learn probabilistic models for PMBGPs. Because LBP generates the individual which most reflects a *good* probabilistic model, it often becomes the best individual. From this result, we see that LBP works well in the problem with no dependencies between nodes. However, this is not the case in DMAX problem where strong deceptiveness exists. In DMAX problem, local optima and a global optimum are very distinct in their structures, whereas their fitness values differ only slightly. Although the global optimum of DMAX problem with depth 4 is gotten by combining optimum with depth 3 and the structure whose fitness is a negative value, LBP tends to generate local optima because the probabilistic model that reflects the nature of deceptiveness is learned. Those local optima have somewhat large fitness and contribute to the increase of the average fitness and are used for the construction of probabilistic model for the next generation, which finally gives rise to the acceleration of convergence to local optima. Nevertheless, because DMAX problem requires large population size and LBP generates only one individual, the negative effect is limited and POLE-BP holds the same performance as POLE. The royal tree problem has dependencies between nodes, i.e. deep building blocks are created by combining shallow building blocks, and an optimum solution is eventually obtained by combining these deep building blocks. In the royal tree problem, for a start, probabilistic model for shallow building blocks is learned, and probabilistic model is gradually refined to generate deeper building blocks in the process of learning. Because deep building blocks are gotten by combining shallow building blocks, by generating the individual which reflects probabilistic model in the middle of learning, LBP contributes to the construction of probabilistic models for deep building blocks, which become to create an optimum solution.

In problems without deceptiveness, *good* probabilistic models toward the global optimum are always learned. Therefore, LBP generates the individual with high fitness, which contributes to the increase of the average fitness and obtaining the global optimum at earlier generation. On the other hand, since probabilistic models that tend to generate local optima are often learned in deceptive problems, LBP generates locally optimal individuals. This is a reason for delay in obtaining the global optimum, however, the delay remains in limited extent because LBP generates only one individual. In addition, we note that LBP works better in problems with more symbols. In the condition of statistically significant D_p ($D_p = 6, 7, 8$ for the MAX problem and $D_p = 6, 7$ for the royal tree problem), LBP reduces more the number of fitness evaluations in royal tree problem (average 35.5%) than MAX problem (average 27.4%). This is considered that royal tree problem has more symbols than MAX problem, and it is more difficult for royal tree problem than MAX problem to generate better solutions by PLS, therefore LBP works better in royal tree problem.

Finally, we discuss the number of optima. Ref. [4] closely examines the behavior of EDA on problems with different numbers of optimal solutions. Ref. [4] shows that although the probability of generation of optima increases monotonically and finally reaches 1 on unimodal problems, multimodal problems make the probability of generation of optima distributed among optima. In other words, multimodality reduces the probability of generation of MPS. Therefore, generation of MPS by LBP should be more effective on multimodal problems than unimodal problems because of the low probability of generation of MPS. However, our experiments show contradictory results. In our experiments, MAX and Royal tree problems are unimodal, and DMAX problem is multimodal. Although LBP improves all measures: the number of evaluations, fitness and building block satisfaction rate on unimodal MAX and Royal tree problems, LBP only improves fitness and building block satisfaction rate on multimodal DMAX problem. We conclude that this contradiction is caused by the strong deceptiveness of DMAX. Because the deceptiveness, which leads MPS to a local optima, is more dominant than multimodality of the DMAX problem, the performance of multimodal DMAX problem does not improve more than the unimodal MAX and Royal tree problem.

6. Conclusion

We proposed POLE-BP, a variant of PMBGP that uses LBP for sampling to generate MPS at each generation. We compared the proposed POLE-BP against existing methods using the number of fitness function evaluations required as the evaluation criterion. Our experimental results showed that POLE-BPB's performance is statistically comparable with that of the existing methods for deceptive problems. Moreover, POLE-BP significantly outperformed other methods in non-deceptive problems. We analyzed the fitness and tree structure of individuals produced by our proposed method against that by existing methods and found that LBP often generates local optima in deceptive problems and does not contribute to obtaining an optimum solution. However, LBP generates individuals with many building blocks and contributes to obtaining an optimum solution in non-deceptive problems. So far, almost all of the prototype tree-based PMBGPs have focused only on the of learning probabilistic models. However, we showed that by improving the sampling method we can improve the search performance of PMBGPs, without any modifications to the learning algorithm. We conclude that both the sampling method and the learning algorithm are equally important for improving the search performance.

References

[1] S. Baluja, Population-based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning, Technical Report, 1994.
[2] P.A.N. Bosman, E.D. de Jong, Grammar Transformations in an EDA for genetic programming, in: GECCO 2004 Workshop Proceedings, 2004.
[3] G.F. Cooper, E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, Mach. Learn. 9 (1992) 309–347.
[4] C. Echegoyen, A. Mendiburu, R. Santana, J.A. Lozano, Toward understanding EDAs based on Bayesian networks through a quantitative analysis, IEEE Trans. Evol. Comput. (2012) 173–189.
[5] R. Etxeberria, P. Larrañaga, Global optimization using Bayesian networks, in: Proceedings of 2nd Symposium on Artificial Intelligence (CIMAF-99), 1999, pp. 332–339.

Please cite this article as: H. Sato, et al., Improved sampling using loopy belief propagation for probabilistic model building genetic programming, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.02.002

[6] A. Farinelli, A. Rogers, A. Petcu, N.R. Jennings, Decentralised coordination of low-power embedded devices using the max-sum algorithm, in: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, vol. 2. AAMAS '08. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2008, pp. 639–646.
[7] P.F. Felzenszwalb, D.P. Huttenlocher, Efficient belief propagation for early vision, Int. J. Comput. Vis. 70 (October) (2006) 41–54.
[8] C. Gathercole, P. Ross, S. Bridge, An adverse interaction between the crossover operator and a restriction on tree depth, in: Proceedings of 1st Annual Conference on Genetic Programming, 1996, pp. 291–296.
[9] G. Harik, Linkage Learning via Probabilistic Modeling in the ECGA, Technical Report, 1999.
[10] Y. Hasegawa, H. Iba, Estimation of Bayesian network for program generation, in: Proceedings of The Third Asian-Pacific Workshop on Genetic Programming, 2006, pp. 35–46.
[11] Y. Hasegawa, H. Iba, Estimation of distribution algorithm based on probabilistic grammar with latent annotations, in: Proceedings of IEEE Congress of Evolutionary Computation (CEC 2007), 2007, 1043–1050.
[12] Y. Hasegawa, H. Iba, A Bayesian network approach to program generation, IEEE Trans. Evol. Comput. 12 (6) (2008) 750–764.
[13] Y. Hasegawa, H. Iba, Latent variable model for estimation of distribution algorithm based on a probabilistic context-free grammar, Trans. Evol. Comput. 13 (August) (2009) 858–878. http://dx.doi.org/10.1109/TEVC.2009.2015574.
[14] M. Hauschild, M. Pelikan, An introduction and survey of estimation of distribution algorithms, Swarm Evol. Comput. 1 (3) (2011) 111–128.
[15] M. Henrion, Propagating uncertainty in Bayesian networks by probabilistic logic sampling, in: UAI, 1986, pp. 149–164.
[16] K. Hirasawa, M. Okubo, J. Hu, J. Murata, Comparison between genetic network programming (GNP) and genetic programming (GP), in: Proceedings of the 2001 Congress on Evolutionary Computation CEC2001, IEEE Press, Seoul, Korea, 27–30 May 2001, pp. 1276–1282.
[17] W.B. Langdon, R. Poli, An analysis of the max problem in genetic programming, in: Advances in Genetic Programming, vol. 3, MIT Press, 1997, pp. 301–323 (Chapter 13).
[18] P. Larra naga, H. Karshenas, C. Bielza, R. Santana, A review on probabilistic graphical models in evolutionary computation, J. Heuristics 18 (October (5)) (2012) 795–819.
[19] X. Li, B. Li, S. Mabu, K. Hirasawa, A continuous estimation of distribution algorithm by evolving graph structures using reinforcement learning, in: IEEE Congress on Evolutionary Computation, 2012, pp. 1–8.
[20] X. Li, S. Mabu, K. Hirasawa, A novel graph-based estimation of distribution algorithm and its extension using reinforcement learning, IEEE Trans. Evol. Comput. (1) (2013) 99.
[21] C.F. Lima, M. Pelikan, F.G. Lobo, D.E. Goldberg, Loopy substructural local search for the Bayesian optimization algorithm, in: Proceedings of the Second International Workshop on Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. SLS '09. Springer-Verlag, Berlin, Heidelberg, 2009, pp. 61–75.
[22] M. Looks, B. Goertzel, C. Pennachin, Learning computer programs with the Bayesian optimization algorithm, in: Proceedings of the 2005 Genetic and Evolutionary Computation Conference, 2005, pp. 747–748.
[23] S. Mabu, K. Hirasawa, J. Hu, A graph-based evolutionary algorithm: genetic network programming (gnp) and its extension using reinforcement learning, Evol. Comput. 15 (Sep. (3)) (2007) 369–398. http://dx.doi.org/10.1162/evco.2007.15.3.369.
[24] A. Mendiburu, R. Santana, J.A. Lozano, Introducing Belief Propagation in Estimation of Distribution Algorithms: A Parallel Framework, Technical Report, 2007.
[25] M. Mitchell, S. Forrest, J.H. Holland, The royal road for genetic algorithms: fitness landscapes and ga performance, in: Proceedings of the First European Conference on Artificial Life, 1992, pp. 245–254.
[26] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, San Francisco, CA, USA, 1988, ISBN: 1558604790.
[27] M. Pelikan, D.E. Goldberg, E. Cantu-Paz, Boa: The Bayesian optimization algorithm. IlliGAL Report No. 98013: Illinois Genetic Algorithm Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.
[28] W.F. Punch, How effective are multiple populations in genetic programming, in: John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, Rick Riolo (Eds.), Genetic Programming 1998: Proceedings of the Third Annual Conference: 308–313308–313. 1998.
[29] A. Ratle, M. Sebag, Avoiding the bloat with probabilistic grammar-guided genetic programming, in: P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, M. Schoenauer (Eds.), Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001, Lecture Notes in Computer Science, 29–31 October 2001, vol. 2310, Springer Verlag, Creusot, France, 2001, pp. 255–266.
[30] E.N. Regolin, A.T.R. Pozo, Bayesian automatic programming, in: Genetic Programming, Lecture Notes in Computer Science, vol. 3447, 2005, pp. 38–49.
[31] R.P. Sałustowicz, J. Schmidhuber, Probabilistic incremental program evolution, Evol. Comput. 5 (1997) 123–141.
[32] K. Sastry, D.E. Goldberg, Probabilistic model building and competent genetic programming, in: Genetic Programming Theory and Practise, 2003, pp. 205–220 (Chapter 13).
[33] K. Sastry, D.E. Goldberg, Probabilistic model building and competent genetic programming, in: Genetic Programming Theory and Practise, Kluwer, Springer, US. ISBN: 978-1-4613-4747-7, 2003, pp. 205–220, http://dx.doi.org/10.1007/978-1-4419-8983-3_13 (Chapter 13).
[34] H. Sato, Y. Hasegawa, D. Bollegala, H. Iba, Probabilistic model building GP with belief propagation, in: X. Li (Ed.), Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, 10–15 June 2012, pp. 2089–2096.
[35] Y. Shan, R.I. Mckay, H.A. Abbass, D. Essam, Program evolution with explicit learning: a new framework for program automatic synthesis, in: Proceedings of the 2003 Congress on Evolutionary Computation CEC2003, 2003, pp. 1639–1646.
[36] Y. Shan, R.I. Mckay, R. Baxter, Grammar model-based program evolution, in: Proceedings of the 2004 IEEE Congress on Evolutionary Computation, 2004, pp. 478–485.
[37] M. Wineberg, F. Oppacher, A representation scheme to perform program induction in a canonical genetic algorithm, in: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, 1994, pp. 292–301.
[38] K. Yanai, H. Iba, Estimation of distribution programming based on Bayesian network, in: Proceedings of the Congress on Evolutionary Computation, 2003, pp. 1618–1625.

Please cite this article as: H. Sato, et al., Improved sampling using loopy belief propagation for probabilistic model building genetic programming, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.02.002