Estimation of Distribution Algorithm based on Probabilistic Grammar with Latent Annotations

Yoshihiko Hasegawa† and Hitoshi Iba†

Abstract—Genetic Programming (GP) which mimics the natural evolution to optimize functions and programs, has been applied to many problems. In recent years, evolutionary algorithms are seen from the viewpoint of the estimation of distribution. Many algorithms called EDAs (Estimation of Distribution Algorithms) based on probabilistic techniques have been proposed. Although probabilistic context free grammar (PCFG) is often used for the function and program evolution, it assumes the independence among the production rules. With this simple PCFG, it is not able to induce the building-blocks from promising solutions. We have proposed a new function evolution algorithm based on PCFG using latent annotations which weaken the independence assumption. Computational experiments on two subjects (the royal tree problem and the DMAX problem) demonstrate that our new approach is highly effective compared to prior approaches.

I. INTRODUCTION

In this paper, we propose a function optimization algorithm based on probabilistic context free grammar (PCFG) with latent annotations (PCFG-LA) [12]. Our approach named PAGE (Programming with Annotated Grammar Estimation) uses latent annotations which enable an induction of promising sub-functions (sub-routines).

GP (Genetic Programming) [7], [8] is an extended algorithm of GA (Genetic Algorithm) and is capable of handling programs and functions. Because GP has structural chromosomes which are very flexible, GP has been applied to many problems (robot engineering, financial engineering, bioinformatics, etc) and is considered to be a very powerful way for solving these problems.

GA and GP evolve solution candidates with crossover and mutation operators. These methods are considered to be highly effective because they take advantage of the mechanism of natural evolution. Recently, Evolutionary Algorithms are seen from the viewpoint of the distribution estimation. A new way for evolving solutions based on the estimation of distribution has been proposed. More and more attention has been paid to this method, which is called EDA (Estimation of Distribution Algorithm) [10], [14]. In EDAs, there are two types of algorithms : those based on GA style linear chromosome (GA-EDA) and those based on GP style tree chromosome (GP-EDA).

In GP-EDA, algorithms can be broadly classified into two groups: proto-type tree based method and PCFG based method. The former method translates variable length tree-structures into fixed length structures (mostly linear arrays).

The later method uses context free grammar (CFG) for expressing programs and functions. Algorithms based on PCFG are considered to be well suited for expressing functions in GP. Thus many algorithms based on the later approach have been proposed (Section II).

The basic PCFG adopts the context freedom assumption that the probabilities of production rules do not depend on the ascendant nodes or sibling nodes. The basic GP-EDA based on PCFG just estimates parameters (production rule probability) and generate the programs using the parameters. Although the independence assumption makes the statistical estimation easier, this PCFG basically can not take into account the interactions among nodes. Thus it is not suitable for grasping the sub-functions or sub-routines in GP programs, because GP functions often have strong interactions among nodes. In order to weaken the independence assumption in PCFG, annotations have been proposed in the natural language. For example, vertical markovization annotates the symbols with their ancestor symbols. Prior GP-EDAs such as PEEL or Grammar Transformation in EDA (GT-EDA) used annotated PCFG for base line grammars. In these algorithms, the depth of the production rules is taken into account. Although these annotations may work in specific problems, this assumption may be too strong or wrong in other problems.

Matsuzaki et. al. proposed the PCFG with latent annotations (PCFG-LA) [12] which assumes that the annotations are hidden and the production rules with annotations can be estimated with EM algorithm. We expect that PCFG-LA is suitable for more precisely grasping the interactions in GP compared to previous methods and it is more flexible compared to the method based on heuristics based annotations as depth.

This paper is structured as follows. In the next section, we briefly introduce the program evolution methods based on probabilistic techniques. In Section III we explain the PCFG-LA which is used in our proposal describing parameters update formula. In section V, we compare the performance of our method to other methods, selecting two benchmark tests for experiments. We discuss the results obtained in the experiment in section VI. Finally, we conclude this paper in section VII.

II. RELATED WORKS

Probabilistic evolutionary algorithms based on probabilistic models have been first carried in GA style linear chromosome. Because GP uses tree structures which are more complex than linear arrays used in GA, GP-EDA is a newer...
field than GA-EDA. Many GP-EDAs have been proposed up
to now and can be broadly classified into two groups: (i) prototype tree based method and (ii) grammar model based
methods.

The method (i) takes advantage of the techniques devised
in GA-EDA. This type of algorithms translates tree structures
into fixed length chromosomes used in GA and applies the
probabilistic model. PIPE (Probabilistic Incremental Program
Evolution) [19] adopted the univariate model which can be
considered as the combination of PBIL (Population Based
Incremental Learning) [1] and GP. EDP (Estimation of
Distribution Programming) [25], [26] considered the parent-
children relationship in the tree structure. EDP considers
conditional probability distribution of children given parent
nodes. ECGP (Extended Compact GP) [20] combines the
ECGA (Extended Compact GA) [4] with GP to take into
account the relationship among nodes. This algorithm esti-
mates the group of joint distribution with MDL principle.
BOAP (Bayesian Optimization Algorithm) [15] to be able to handle programs and functions. BOAP (BOA
Programming) [11] uses zig-zag tree which is based on
lambda function. POLE (Program Optimization with Linkage
Estimation) [5], [6] estimated the interactions among nodes
with estimating the Bayesian network. In this algorithm,
special chromosome called expanded parse tree is used to
convert the GP programs into linear arrays.

The method (ii) which is based on PCFG has a lot to do
with GGGP (Grammar Guided Genetic Programming) [24].
GGGP applies the CFG to GP. SG-GP (Stochastic Grammar
based GP) [17] uses simple PCFG. SG-GP integrates PBIL
with GGGP. In this algorithm, the parameters are updated
using an incremental learning strategy. PEEL (Program Evo-
lution with Explicit Learning) [21] is an extended algorithm
of SG-GP which takes into account the depth of symbols.
GMPE (Grammar Model based Program Evolution) [22]
merge the production rules to find the general production
rules. GMPE starts from specialized production rules and
merge non-terminals to be more general production rules
with MDL principle. Grammar Transformation in EDA (GT-
EDA) [2] takes advantage of MDL principle to estimate
expanded production rules (tree structure) to extract good
subroutines. BAP (Bayesian Automatic Programming) [18]
uses Bayesian network to consider the relation among pro-
duction rules.

Unlike PEEL and GT-EDA which use fixed annotations,
our proposal uses annotations which are induced from
promising solutions using EM algorithm. Our method is more
flexible than the fixed annotation model and we expect that
our method is more effective than the methods based on fixed
annotations. In GA-EDA, an algorithm taking advantage of
latent variables have been proposed [23]. In this work, EM
algorithm is also used to estimated the parameters.

III. PCFG WITH LATENT ANNOTATIONS

In CFG, many approaches have been proposed to weaken
the context freedom assumption by annotating non-terminal
symbols with many features. Many of the works use fixed
annotations as labels of ancestor symbols, and sibling nodes.
Matsuzaki et. al. [12] proposed the probabilistic context free
grammar with latent annotations (PCFG-LA) which automat-
ically induces the production rules with latent annotations.
They also derived the parameter update formula with EM
algorithm.

In PCFG-LA, every non-terminal is labeled with annota-
tions. In the complete form, non-terminals are represented
by
\[ \Theta \]

\[ \beta(r) \]

\[ \pi \]

\[ \lambda \]

Because annotations are not observed, the parameters \( \Theta \) have to be estimated by EM algorithm. The difference
of log-likelihood between parameters \( \Theta' \) and \( \Theta \) can be
calculated by Equation 3.

\[ \log P(T; \Theta') - \log P(T; \Theta) \]

\[ = \log \frac{P(T; \Theta')}{P(T; \Theta)} \]

\[ = \sum_X P(X|T; \Theta) \log \frac{P(T, X; \Theta')}{P(T, X; \Theta)} \frac{P(X|T; \Theta)}{P(X|T; \Theta')} \]

\[ \geq \sum_X P(X|T; \Theta) \log \frac{P(T, X; \Theta')}{P(T, X; \Theta)} \]

(3)
Using Equation 3, the update formula can be obtained by optimizing $Q(\Theta'|\Theta)$ (Equation 4).

$$Q(\Theta'|\Theta) = \sum_{T_i \in T} \sum_{X_i} P(X_i|T_i; \Theta) \log P(T_i, X_i; \Theta')$$

In this paper, production rules are not CNF (Chomsky Normal Form) which is assumed in the original PCFG-LA paper, because of understandability of GP programs. Any functions which can be handled with traditional GP can be represented by

$$S \rightarrow g S ... S,$$

which is a subset of GNF (Greibach Normal Form). Here $S \in N$ and $g \in T$ ($N$ and $T$ are sets of non-terminal and terminal symbols in CFG, respectively). $g$, terminal symbol in CFG, is a function node ($+$, $-$, $\sin$, $\cos$) or a terminal $(x, y)$ in GP. In the program evolution, only $S$ is used for non-terminal. In the right-side of Equation 5, the number of $S$ is identical to the arity of $g$. If $g = +$, then $S \rightarrow + SS$ and if $g = x$, then $S \rightarrow x$. Annotated production rules can be expressed by Equation 6,

$$S[x] \rightarrow g S[z_1] ... S[z_n]$$

where $x, z_m \in H$, $a$ is the arity of $g$ in GP. However, we made an assumption in our algorithm because of the following reason. Let us $h$ be the size of annotation $H$. If $g$ has an arity, the number of parameters for the production rule $S \rightarrow g S ... S$ with annotations becomes $h^{a+1}$. Estimating the large number of parameters may cause an over fitting problem and requires much more computational power. In order to reduce the number of parameters, we assume that all right-side non-terminal symbols have the same annotation,

$$S[x] \rightarrow g S[y] S[y] ... S[y]$$

With this assumption, the size of parameters can be reduced to $h^2$, which is tractable. Because there are many functions which are commutative (e.g. $+$, $\times$, etc) in GP, we think that this assumption may not be so strong. From now on, we assume Equation 7 in the following explanation. In the following sections, let $R[H]$ be the set of annotated rules expressed by Equation 7.

$$R[H] = \{S[x] \rightarrow g S[y] S[y] ... S[y]|x, y, \in H, g \in T\}$$

A. Forward-backward probability

For calculating the Equation 4, forward-backward probability is used. Backward probability $b_f^T(x)$ stands for the probability that the tree beneath $n$-th non-terminal $S[x]$ is generated. Forward probability $f_f^T(y)$ stands for the probability that the tree above $n$-th non-terminal $S[y]$ is generated. These probabilities can be recursively calculated (Equation 8, 9 and 10).

$$b_f^T(x) = \sum_{y \in H} \beta(S[x] \rightarrow g_i S[y] ... S[y]) \prod_{j \in ch(i, T)} b_f^T(y)$$

$$f_f^T(y) = \sum_{x \in H} \beta(S[x] \rightarrow g S[y] ... S[y]) \prod_{j \in ch(i, T)} f_f^T(x)$$

$$f_f^T(y) = \sum_{x \in H} \beta(S[x] \rightarrow g S[y] ... S[y]) \prod_{j \in ch(i, T)} f_f^T(x)$$

$$f_f^T(y) = \sum_{x \in H} \beta(S[x] \rightarrow g S[y] ... S[y]) \prod_{j \in ch(i, T)} f_f^T(x)$$

$$f_f^T(y) = \sum_{x \in H} \beta(S[x] \rightarrow g S[y] ... S[y]) \prod_{j \in ch(i, T)} f_f^T(x)$$

$$f_f^T(y) = \sum_{x \in H} \beta(S[x] \rightarrow g S[y] ... S[y]) \prod_{j \in ch(i, T)} f_f^T(x)$$

$$f_f^T(y) = \sum_{x \in H} \beta(S[x] \rightarrow g S[y] ... S[y]) \prod_{j \in ch(i, T)} f_f^T(x)$$

In this equations, $ch(i, T)$ is a function which returns the set of non-terminal children indices of $i$-th non-terminal in $T$. $pa(i, T)$ returns a parent index of $i$-th non-terminal in $T$. $g_f^T$ is a terminal symbol in CFG and is connected to $i$-th non-terminal symbol in $T$. For example, $ch(3, T) = \{5, 6\}$, $pa(5, T) = 3$ and $g_f^T = \sin$ in Figure 2.

Using the forward-backward probability, $P(T; \Theta)$ can be represented by following Equations.

$$P(T; \Theta) = \sum_{x \in H} \pi(S[x]) b_f^T(x)$$

$$P(T; \Theta) = \sum_{x, y \in H} \beta(S[x] \rightarrow g S[y] ... S[y]) f_f^T(x)$$

In this equation, $cover(g, T)$ represents a function which returns a set of non-terminal indices at which the production rule generating $g$ without annotations is rooted to $T$. For example, if $g = +$ and $T$ is a tree represented in Figure 2, then $cover(+, T) = \{1, 3\}$.

B. Parameter update formula

Using the forward-backward probability and optimizing $Q(\Theta'|\Theta)$, the update formula can be derived. Because our approach is not based on CNF and we limit the right-side annotations in production rules, the update formula is different from that in the original paper (see Appendix).

$$\pi'(S[x]) \propto \pi(S[x]) \sum_{T_i \in T} \frac{b_f^T(x)}{P(T_i)}$$

$$\beta(S[x] \rightarrow g S[y] ... S[y]) \propto \beta(S[x] \rightarrow g S[y] ... S[y]) \sum_{T_i \in T} \frac{1}{P(T_i)} \sum_{j \in cover(g, T_i)} f_f^T(x) \prod_{k \in ch(j, T_i)} b_f^T(y)$$
EM algorithm maximizes the log-likelihood \( L(\theta; T) = \log P(T; \theta) = \sum_{x \in T} \log P(T|x; \theta) \) monotonically from the initial parameters. For initial parameters, we used the values represented with Equation 15 and 16.

\[
\beta(S[x] \rightarrow g S[y]...S[z]) \propto e^{-\kappa} \gamma(S \rightarrow g S...S) \tag{15}
\]

\[
\beta(S[x] \rightarrow g) \propto e^{-\kappa} \gamma(S[x] \rightarrow g) \tag{16}
\]

In Equation 15, \( \kappa \) is a random value which is uniformly distributed over \([−\log 3, \log 3]\) and \( \gamma(S \rightarrow g S...S) \) is a probability of observed production rule (without annotations). We also set \( \beta(S[x] \rightarrow g S[y]...S[z]) = 0 \).

Because the obtained parameters are greatly affected by initial values, we carried EM algorithm several times (in this paper, 3 times at each generation) with different initial parameters and then adopted the parameters which gave the best likelihood.

IV. FLOWCHART

Combining the PCFG-LA with GP, we propose a new algorithm PAGE (Programming with Annotated Grammar Estimation). In this section, a flowchart of PAGE is shown.

1) Initialization of individuals

Individuals are generated randomly according to the initial parameters. The number of initialized individuals is represented with \( M \).

2) Evaluation of individuals

All individuals are assigned fitness values according to the evaluation function.

3) Selection of individuals

Promising individuals are selected which are used for the parameter estimation. In our implementation, a truncate selection is used. The number of individuals to select is represented by \( M \times P_e \).

4) Estimation of parameters

Parameters are estimated with EM algorithm explained in the proceeding section. Parameters are estimated at every generation. We stop updating the parameters if the log-likelihood improvement is smaller than 0.5%.

5) Generation of new individuals

New individuals are generated with estimated parameters. If we want to use an elitist strategy, the best \( M \times P_e \) individuals are copied from the previous generation.

Steps from (2) to (5) are repeated until termination criteria are met.

V. COMPARATIVE EXPERIMENTS

We carried comparative experiments to show the effectiveness of our approach (PAGE). We compared the search performance among the algorithms listed below.

- PAGE

This is our proposed method. This method extends PCFG-GP by introducing the latent annotations. The number of latent annotations \( h(h > 1) \) is given beforehand. The annotations in PAGE are represented by integer \( (H = \{0, 1, ..., h − 1\}) \).

- PCFG-GP

We denote PCFG-GP for expressing a GP based on a simple PCFG. This algorithm does not take annotations into account. This is \( h = 1 \) case of PAGE, and it is basically identical to the GP-EDA which uses simple PCFG for probabilistic model (SG-GP [17]).

- Simple GP (SGP)

This is a simple implementation of GP. In the experiments, \( P_c = 0.01 \) (elite rate), \( P_e = 0.9 \) (crossover rate), \( P_m = 0 \) (mutation rate) and \( P_r = 0.09 \) (reproduction rate) are used. Crossover points are selected in the following way: When applying crossover to two individuals, we select the first crossover point from function nodes at the probability of 0.9 and from terminals at 0.1. The second crossover point is selected under the condition that the depth of both individuals does not exceed the depth limitation.

A. Royal Tree Problem

We applied PAGE to the royal tree problem to compare the performance between PAGE and PCFG-GP. This experiment shows the effectiveness of latent annotations.

1) Problem Description: In order to show the effectiveness of latent annotations, we compared the search performance in the royal tree problem [16]. The royal tree problem is an extension of the royal road function [13] which has been the famous benchmark test in GA. The royal tree problem uses a set of functions which are defined with alphabetical symbols \( \mathcal{S} = \{a, b, c, d, ...\} \) which have increasing arity (\( a \) has 1 arity, \( b \) has 2 arity and so on) and terminal nodes \( \mathcal{T} = \{x\} \). \( \mathcal{S} \) and \( \mathcal{T} \) represents set of GP functions and GP terminals, respectively. The royal tree problem defines the state \( \text{perfect tree} \) in each level. Perfect tree of some level is composed of the perfect tree whose level is smaller by 1 level than the level. Perfect tree of level \( c \) is composed of the perfect tree of level \( b \). In perfect trees, alphabets of functions descend by one from a root to leaves in a tree. A function \( a \) has a terminal \( x \). For more details, see [16].

In our setups, we used level \( d \) royal tree problem. The production rules for the royal tree problem are shown in the list below, where symbols with small case denote terminal symbols in CFG. \( a \sim d \) are function nodes and \( x \) is a terminal node in GP.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M )</td>
<td>Population size</td>
<td>–</td>
</tr>
<tr>
<td>( G )</td>
<td>Generations to run</td>
<td>–</td>
</tr>
<tr>
<td>( h )</td>
<td>The number of annotations</td>
<td>–</td>
</tr>
<tr>
<td>( P_e )</td>
<td>Selection rate</td>
<td>0.1</td>
</tr>
<tr>
<td>( P_c )</td>
<td>Elite rate</td>
<td>0.1</td>
</tr>
</tbody>
</table>
In this experiment, $M = 1000$ is used in both PAGE and PCFG-GP. We observe the behavior of our algorithm in terms of the number of latent annotations ($h$) with $h = 2, 4, 8, 16$ in PAGE.

2) Results: Figure 3 represents the probability of success at each generation. As can be seen in Figure 3, PCFG-GP did not obtain the optimum at all. This reason can be easily estimated. Table II (lower table) describes the obtained production rule probabilities of PCFG-GP. As can be seen with this table, probability of production rule $S \rightarrow x$ is the highest. Thus using simple PCFG, the size of generated individuals tends to be small, because probabilities of large trees tends to be very small. On the other hand, our approach uses the production rules with latent annotations. In Table II (upper table), the production rules with annotations are described ($h = 8$). In this table, because of space limitation, we omitted the production rules which have very small probabilities (smaller than 0.01). In the probabilities with annotations, the annotations are used for generating different symbols. For example, $S[0]$ and $S[3]$ generate $x$ and $S[1]$ generates $d$ exclusively. Using these production rules, we can grasp the sub-structures with higher probability.

In PAGE, the number of annotations $h$ strongly affect the search performance. In Figure 3, we can see that the larger $h$ gives the better performance. In the case of smaller annotation size, there are many overlapping symbols. For example in Table III, we can see $S[2]$ mainly generates $a S[0]$ and $a S[3]$. However, $S[2]$ also generates $b S[3] S[3]$ and $c S[3] S[3] S[3]$. This overlap makes the probability $P(T; \Theta)$ of the optimum smaller and it is more difficult to find the optimum. On the other hand, in the case of large $h$, there are many redundant symbols. Thus overlappingly used symbol as $S[2]$ is less in this case.

PAGE estimates the parameters with EM algorithm which monotonically increases the likelihood. Figure 4 describes the increase of log-likelihood ($\sum_{T_i \in T} \log P(T_i; \Theta)$) in the royal tree problem using PAGE. We show the transition of generation 0 and generation 11. As can be seen with this figure, parameters are converged around by 10 iterations. The log-likelihood improvement at generation 11 is larger than that at generation 0, because the tree structures are converged at the end of the search.
B. DMAX problem

1) Problem Description: We applied our approach to DMAX problem [5] to show the superiority of our proposal over Simple GP. DMAX problem is an extended problem of the MAX problem. Because the original MAX problem does not have deceptiveness, it is very easy to solve by GP-EDAs without considering the interactions. We extended the MAX problem by adding the deceptiveness and we call this extended problem deceptive MAX problem (DMAX problem). We employed the function \( \mathcal{F} \) and terminal \( \mathcal{T} \) given by

\[
\mathcal{F} = \{+_{m}, \times_{m}\} \quad \text{(17)}
\]

\[
\mathcal{T} = \{\lambda, 0.95\} \quad \text{(18)}
\]

with

\[
\lambda^r = 1, \lambda \in \mathbb{C}, r \in \mathbb{N}
\]

where \( +_{m} \) and \( \times_{m} \) are \( m \) arity version of \( + \) and \( \times \), respectively, and \( \lambda \) is generally a complex value.

The main objective of the DMAX problem is identical to the original one: to find the functions which return the largest real value under the limitation on maximum tree depth \( D \). However, the symbols used in the DMAX problem are different from those used in the MAX problem. The DMAX problem uses the symbols represented with Equation 17. A fitness value of individual is the real part of its function. If the value of a function is \( a + bi \) (where \( a, b \in \mathbb{R}, i = \sqrt{-1} \)), then its fitness value is \( a \). The DMAX problem can be represented with 3 parameters, \( m \) (arity), \( r \) (power) and \( D \) (maximum tree depth).

In this problem, the optimum value is more complicated than that of the MAX problem [3], [9] which does not have the deceptiveness. We explain the optimum with an intuitive example. We use \( m = 5, r = 3 \), for example, because DMAX problem has very strong deceptiveness with this setup. In the comparative experiment, we used \( D = 4 \), but for simplification, we explain the case with \( D = 3 \) (Figure 5). In this parameter setup, \( \lambda = \cos \frac{2\pi}{5} + i \sin \frac{2\pi}{5} \). First, we add \( \lambda \) with \( +_{5} \) to make \( 5\lambda \). Then we multiply this value with \( \times_{3} \) which leads to \( (5\lambda)^{2} = 5^{3}\lambda^{2} = 5^{3}\lambda^{2} \). However, \( \text{Re}(5^{3}\lambda^{2}) \) is negative value and is not a good solution. So 2 values out of 5 values multiplied have to be real values. \( 5 \times 0.95 \) is used as a substitution for \( 5\lambda \) and the maximum value is \( (5\lambda)^{2}(0.95 \times 5)^{2} = 2820.3125 \). In \( D = 4 \), the maximum value is \( (5\lambda)^{2}(0.95 \times 5) = 2.83 \times 10^{17} \). Figure 6 shows a visualization of the values in the complex plane whose horizontal (real) axis expresses a fitness value. As can been seen, the replacement of one \( 0.95 \times 5 \) with \( 5\lambda \) increases the argument by 120 degrees in the complex plane. After three times of replacement, the value returns to the same phase but has a larger absolute value.

The production rules employed in PAGE and PCFG-GP are represented by following equations.

\[
\begin{align*}
S & \rightarrow +_{3}SSSSS \\
S & \rightarrow \times_{3}SSSSS \\
S & \rightarrow 0.95 \\
S & \rightarrow \lambda
\end{align*}
\]

For the DMAX problem of \( D = 4, m = 5 \) and \( r = 3 \), we used \( M = 3000 \) in PAGE, PCFG-GP and SGP. The number of latent annotations \( h \) is set to 8. The tournament size of SGP is \( t_s = 2 \), \( t_s = 4 \). We run each algorithm 20 times and observed the probability of success at each generation. We stop every algorithm when they obtain the optimum. Furthermore, we stop PAGE and PCFG-GP when the best fitness is not improved for 10 consecutive generations and stop SGP when the best fitness is not improved for generation \( g \) to \( 2g \) \((g > 10)\).

2) Result: We show the probability of success in DMAX problem in Figure 6. Because PCFG-GP could not find the optimum even at once, we omit the result of PCFG-GP. In DMAX problem, two building-blocks have to be used to obtain the optimum: the one composed of five 0.95 and...
TABLE III
ESTIMATED PROBABILITIES IN PAG E (h = 8). BECAUSE OF SPACE LIMITATION, RULES WITH PROBABILITIES SMALLER THAN 0.01 ARE OMITTED.

<table>
<thead>
<tr>
<th>Production Rule (PAGE)</th>
<th>Probability (× 10^-2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S[2]</td>
<td>0.691</td>
</tr>
<tr>
<td>S[4]</td>
<td>0.132</td>
</tr>
<tr>
<td>S[5]</td>
<td>0.174</td>
</tr>
<tr>
<td>$S \rightarrow \times S[h</td>
<td>S]</td>
</tr>
<tr>
<td>$S[1] \rightarrow \lambda</td>
<td>0.0218</td>
</tr>
<tr>
<td>$S[1] \rightarrow 0.95</td>
<td>0.978</td>
</tr>
<tr>
<td>$S[2] \rightarrow \times S[0]</td>
<td>S[0]</td>
</tr>
<tr>
<td>$S[3] \rightarrow \lambda</td>
<td>0.972</td>
</tr>
<tr>
<td>$S[3] \rightarrow 0.95</td>
<td>0.977</td>
</tr>
<tr>
<td>$S[4] \rightarrow \times S[0]</td>
<td>S[0]</td>
</tr>
<tr>
<td>$S[5] \rightarrow \times S[0]</td>
<td>S[0]</td>
</tr>
<tr>
<td>$S[9] \rightarrow \lambda</td>
<td>0.986</td>
</tr>
<tr>
<td>$S[9] \rightarrow 0.95</td>
<td>0.0136</td>
</tr>
</tbody>
</table>

The optimized parameters are greatly affected by the initial parameters. In this paper, we tried EM algorithm 3 times at each generation and adopted the set of parameters which offered the best log-likelihood. More sophisticated way to optimize parameters may be used.

VI. DISCUSSION

We have employed the PCFG with latent annotations which weakens the independence assumption. Because the statistical estimation in PAGE uses EM algorithm and is a much more complex way compared to simple PCFG, our approach is more computationally expensive. However, the performance of PAGE is much more superior to the algorithm not including the annotations. We can say that PAGE is a stronger algorithm than simple PCFG-GP as a whole.

PAGE has extra parameters and the number of annotations has to be given in advance. The experiment of the royal tree problem showed that the larger $h$ gave the better performance. However, an estimation of larger $h$ requires more computational time. We have to optimize these two contradicting factors which will be examined in the future work. Because EM algorithm is basically hill-climbing strategy,
Equation 19.

\[ \log \pi'(S|x_i') + \log \prod_{r \in D_r(x_i)} \beta'(r) \]  

(19)

In this equation, \( x_i' \) is an annotation of a \( j \)th node in an \( i \)th tree. Equation 19 consists of two terms \( \pi'(S|x_i') \) and \( \beta'(r) \), and both parts can be minimized independently. Let the left and right part be \( Q_{\beta} \) and \( Q_{\beta'} \), respectively.

We only show the derivation of \( \beta' \), because \( \beta' \) can be derived in the similar and easier way. By using Equation 12, \( Q_{\beta} \) can be represented by Equation 20.

\[ Q_{\beta} = \sum_{\mathbf{T} \in \mathcal{T}} \frac{1}{P(T_i; \Theta)} \sum_{j \in \text{COV}(y,T_i)} \sum_{x_i} P(X_i, T_i; \Theta) \sum_{r \in D_r(x_i)} \log \beta'(r) \]  

(20)

Let us focus on the production rule which generates \( g(S|x) \rightarrow g S[y_j] \ldots S[y_j] \). Equation 20 can be decomposed into Equation 21.

\[ Q_{\beta}(g) = \sum_{\mathbf{T} \in \mathcal{T}} \frac{1}{P(T_i; \Theta)} \sum_{j \in \text{COV}(y,T_i)} \sum_{x_i} P(X_i, T_i; \Theta) \]  

\[ \times \log \beta'(S|x_i') \rightarrow g S[y_j] \ldots S[y_j] \]  

(21)

Let \( x_i' \) be the annotation of \( j \)th non-terminal at \( T_i \) and \( y_j' \) be the annotation of right-side symbol of \( j \)th non-terminal at \( T_i \).

\[ Q_{\beta}(g) = \sum_{\mathbf{T} \in \mathcal{T}} \frac{1}{P(T_i; \Theta)} \sum_{j \in \text{COV}(y,T_i)} \sum_{x_i} P(X_i, T_i; \Theta) \]  

\[ \times \log \beta'(S|x_i') \rightarrow g S[y_j'] \ldots S[y_j'] \]  

(21)

To optimize \( Q(\Theta'|\Theta) \) with constraint \( \sum_{r \in \mathcal{R}[H]} \beta'(S|x) \rightarrow \zeta = 1 \) \((\zeta \in \eta[S|x] \rightarrow \eta \in \mathcal{R}[H])\), Lagrange multiplier \( \mu \) is used.

\[ \mathcal{L} = Q_{\beta} + \mu \left( 1 - \sum_{\zeta \in \mathcal{R}[H]} \beta'(S|x) \rightarrow \zeta \right) \]  

(22)

Parameter update formula can be obtained by solving Equation 22. Update formula of \( \pi'(S|x) \) in \( Q_{\beta} \) can be derived in the same ways.