Grammar-Based Genetic Programming with Dependence Learning and Bayesian Network Classifier

Pak-Kan Wong
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong
pkwong@cse.cuhk.edu.hk

Leung-Yau Lo
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong
lylo@cse.cuhk.edu.hk

Man-Leung Wong
Department of Computing and Decision Sciences
Lingnan University
Tuen Mun
Hong Kong
mlwong@ln.edu.hk

Kwong-Sak Leung
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong
ksleung@cse.cuhk.edu.hk

ABSTRACT

Grammar-Based Genetic Programming formalizes constraints on the solution structure based on domain knowledge to reduce the search space and generate grammatically correct individuals. Nevertheless, building blocks in a program can often be dependent, so the effective search space can be further reduced. Approaches have been proposed to learn the dependence using probabilistic models and shown to be useful in finding the optimal solutions with complex structure. It raises questions on how to use the individuals in the population to uncover the underlying dependence. Usually, only the good individuals are selected. To model the dependence better, we introduce Grammar-Based Genetic Programming with Bayesian Network Classifier (GBGPBC) which also uses poorer individuals. With the introduction of class labels, we further propose a refinement technique on probability distribution based on class label. Our results show that GBGPBC performs well on two benchmark problems. These techniques boost the performance of our system.

Categories and Subject Descriptors

I.2 [Artificial Intelligence]: Automatic Programming

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s). Copyright is held by the author/owner(s).

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s). Copyright is held by the author/owner(s).

GECCO ’14, July 12–16, 2014, Vancouver, BC, Canada.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2862-9/14/07...$15.00.
http://dx.doi.org/10.1145/2576768.2598256.
short, we utilize the discrimination power of probabilistic classifier and knowledge on the class in order to generate fitter individuals. This can be viewed as a way to increase the diversity.

In this paper, a new PMBGP framework, called Grammar Based Genetic Programming with Bayesian Network Classifier (GBGPBC), is proposed for the purpose of learning the dependence to boost the performance of PMBGP. GBGPBC learns the local dependence from both good and poor individuals through Bayesian network (BN) classifiers. Each BN classifier aims at discriminating these two classes of individuals by using the non-terminals as the attributes. GBGPBC also generates individuals using two different schemes. Our approach is scalable to large problem and efficient.

This paper is organized as follows. Section 2 summarizes existing works related to PMBGP approaches and BN classifiers. Then our grammar model is formalized in Section 3. The details on the work flow of the whole system are discussed in Section 4. We apply our system in two benchmark problems and the results are shown in Section 5. Finally, we discuss the performance and the potentials of the new GBGPBC.

2. RELATED WORKS

Since our framework is closely related to probabilistic model-building genetic programming and applies BN classifier, the related works in these areas are summarized.

2.1 PMBGAs and PMBGP

The objective of Probabilistic Model-Building Genetic Algorithm (PMBGA) or Estimation of Distribution Algorithm (EDA) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to describe a population of optimal solutions of an algorithm (PMBGA) or Estimation of Distribution Algorithm (ED) [4] is to descr...
conditional independence assumption using a tree structure on the predicator attributes. TAN can be generalized to use a k-tree in which each attribute has at most k attribute nodes as parents. It uses conditional mutual information between attributes given the class variable as the weights and constructs a maximum weighted spanning tree. It balances the use of information in the network and the network complexity giving a competitive classification power [37]. The structures of these BN classifiers are depicted in Figure 1. With these attractive features, our system has adopted multiple TANs in learning the conditional probabilities in the PCSG.

3. GRAMMAR MODEL

Our PCSG formulation differs from previous works in the introduction of the class labels and combining grammar rules with Bayesian network classifiers. Formally, a PCSG $G$ is defined by 7-tuple:

1. A set of non-terminal symbols $N$
2. A set of terminal symbols $T$
3. A start non-terminal symbol $S \in N$
4. A set of derivation rules $R \subset N \times (N \cup T)^*$
5. A set of context elements $C$
6. A set of class labels $X$
7. A set of derivation probabilities $D$ containing the probabilities $p(w, c, x | r)$, where $r \in R$ and $c$ contains non-terminal in its right-hand side, $L'_r = \{s \in R : LHS(s) = ith element in RHS(r)\}$, $L' = L'_1 \times L'_2 \times ... \times L'_{|RHS(r)|}$, $w \in L'$, $c \subseteq C$ and $x \in X$. In other words, $L'_r$ contains all the possible ways to derive the $ith$ non-terminal and $L'$ contains all valid combinations for the whole rule $r$. $p(w, c, x | r)$ gives the joint distribution of the context elements, the class label and the expansion choices of the non-terminals for a rule $r$.

A right-hand side function $RHS : R \rightarrow N^*$ is defined to obtain an ordered set of non-terminals on the right-hand side of a derivation rule. Similarly, a left-hand side function $LHS : R \rightarrow N$ gives the left-hand side of a derivation rule. Assuming there are only two classes labelled as $p$ (poor) and $g$ (good), the formalized version of an example of the PCSG for the Deceptive Max Problem [17] is depicted in Table 1. We use a pair of square brackets (e.g. [0.95]) to enclose the terminals while the non-terminals (e.g. $Exp, Start$) are not enclosed by them. Since each rule is associated with a BN classifier, there are in total five BN classifiers representing $p(w, c, x | r)$. When the set of context elements $C$ is empty, we may use $p(w, r)$ to represent $p(w, \emptyset, x | r)$ for notational simplicity.

Table 1: Formalized PCSG for the DMax problem.

<table>
<thead>
<tr>
<th>$N$</th>
<th>(Start, Exp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>[s, +, 0.95, X]</td>
</tr>
<tr>
<td>$S$</td>
<td>Start</td>
</tr>
<tr>
<td>$R$</td>
<td>{(r_0, r_1, r_2, r_3, r_4)}, where</td>
</tr>
<tr>
<td></td>
<td>$r_0 = Start \rightarrow Exp$</td>
</tr>
<tr>
<td></td>
<td>$r_1 = Exp \rightarrow x</td>
</tr>
<tr>
<td></td>
<td>$r_2 = Exp \rightarrow +</td>
</tr>
<tr>
<td></td>
<td>$r_3 = Exp \rightarrow 0.95$</td>
</tr>
<tr>
<td></td>
<td>$r_4 = Exp \rightarrow \lambda$</td>
</tr>
<tr>
<td>$C$</td>
<td>{}</td>
</tr>
<tr>
<td>$X$</td>
<td>{q, p}</td>
</tr>
<tr>
<td>$D$</td>
<td>Initial probability distributions for the rules $p(w, \emptyset, x</td>
</tr>
<tr>
<td></td>
<td>$p(w, \emptyset, x</td>
</tr>
</tbody>
</table>

Given the grammar, we can derive an individual from the start symbol $S$ by choosing the derivation rules for each non-terminal. Previous works on PMBGP applied different probabilistic models for the variables (e.g. PIPE: a position-dependent model; POLE: a global dependence model for all nodes in a parse tree). They did not use classifier nor did they learn the model using the poor individuals. In contrast, GBGPBC applies a BN classifier for each rule that encodes the choice dependency (i.e. $p(w, c, x | r)$) and modifies the distributions based on the class label.

4. PROPOSED ALGORITHM

In this section, we present the details in GBGPBC. The whole process of GBGPBC involves seven steps.

1. Initialize a BN classifier with independent uniform distribution for each derivation rule.
2. Produce a population of individuals from the grammar following the (refined) probabilities specified in the BN classifiers given the class label which is randomly chosen along the generation.
3. Select the fitter individuals and the poorer individuals based on the fitness values and label the respective parse tree as “good” ($g$) and “poor” ($p$) accordingly.
4. Collect the usage counts of class label and derivation choices for each rule from the selected individuals.
5. For each production rule in the grammar, check if the number of samples accumulated exceeds a predefined threshold, if so, its BN classifier is reconstructed and the samples are then cleared.
6. Refine the probability distributions in the BN classifier based on the class labels so as to reduce the probability of generating poor individuals.
7. Repeat from step 2 and the selected good individuals survive to the next generation.
Step 2 to step 7 are repeated until certain criteria, such as reaching the maximum number of generation, are satisfied.

### 4.1 Initialization of Bayesian Networks

Initially, all variables are assumed to be independent and the joint probability distribution is uniform. Therefore, no edge can be found between any nodes in the BN classifiers.

### 4.2 Individual Generation

An individual is essentially a tree satisfying the language defined in the grammar. We construct a tree starting from the only rule with the start symbol $S$. Then we choose a class label $x$ from $\{g, p\}$ with a probability $prob_g$ and $prob_p$ respectively, where $g$ and $p$ represent the class of “good” and “poor” individuals respectively; $prob_g + prob_p = 1$. The rules for deriving all non-terminals on the right-hand side of it will be chosen according to the probability distribution $p(w, x|r)$ of the starting rule. For any newly derived non-terminals, we recursively apply this procedure until no more non-terminal needs to be instantiated.

To illustrate the derivation procedure, an example is shown in Figure 2. We focus on the derivation of the $Exp_0$ node (circled in dotted line) under the $Start$ node. At first, we instantiate the rule $r = Exp \rightarrow [+] Exp_0 Exp_1 Exp_2$ according to a topological order in the BN classifier (a TAN classifier) $B_r$, (i.e. $X Exp$, $Exp_0$ $Exp_2$ in the Figure 1d), where the subset distinguishes the three $Exp$ non-terminals in the RHS. In a TAN classifier, $Exp_0$, $Exp_1$ and $Exp_2$ must form a tree structure. Then we choose a class label $x$ from $\{g, p\}$ with a fixed probability $prob_g$ and $prob_p$ respectively, say $g$. Next, we choose the state for $Exp_1$. It has four possible states representing the four rules (i.e. 2.0, 2.1, 2.2, 2.3) of $Exp$ respectively. To instantiate $Exp_1$, we choose a state according to the distribution $p(Exp_1|g)$, say state 3, so the derivation rule 2.3 of $Exp$ is picked. After that, we instantiate $Exp_0$ according to the distribution $p(Exp_0|Exp_1, g)$ and obtain 2. We follow the same procedure for $Exp_2$ to get 1. Therefore, the states are $(2, 3, 1)$ which means the rule is derived to $(2, 2, 2.3, 2.1)$. The chosen rules can then be recursively expanded to produce a parse tree. The chosen rule by the non-terminal will be as shown in the individual $i_0$ in Figure 2, where $n$ in the circles represent the instantiated choice of state for that non-terminal is $n$.

### 4.3 Learning Bayesian Network Classifier

After generating and evaluating all the individuals in a population, the individuals are sorted according to their fitness values. The number of individuals selected depends on the selection rate $r_{select}$ and $n_{select} = r_{select} \times pop$, where $pop$ is the population size. The first $n_{select}$ of individuals are labelled as $g$. Another parameter $pivot$ ranging from 0 to 1 is $r_{select}$ for locating the position of poor samples. The $n_{select}$ of individuals starting at $pivot \times pop$ are labelled as $p$. With the categorized selected individuals as training examples, BN classifiers are learnt.

Since each node in the parse tree records the rule choice, it is possible to acquire the set of applied rules in constructing an individual. In the example in Figure 2, we know that the $Start$ symbol applies the rule 1.0, and the children $Exp$ is in turn instantiated to state 1 and the non-terminals of the right-hand side of the rule 2.1, are being instantiated to states $(2, 3, 1)$. We collect the derivation events together with the class label of the respective parse tree, and construct a table for each derivation rule (See Figure 2). With this table of cases, BN classifiers can be used to model the relations of variables in the network. In this work, the TAN classifier learning algorithm is chosen to learn the structures of the BN classifiers for the variables to classify the class labels given the derivation events. Since the population will gradually bias to the optimal solutions, the classifiers have to be reconstructed from time to time based on a parameter $accumulation-size$. They are reconstructed only when there are at least $accumulation-size$ number of cases. It is encouraged to start with a small $accumulation-size$, say 15, when applying this system to a new problem.

### 4.4 Context Elements during Derivation

The proposed grammar model allows us to add extra contextual information to the conditional probabilities. In the BN classifier, a new predictor variable corresponding to each context element will be inserted (preceding the nodes for the non-terminals). Two context elements are introduced to improve the performance of GGPBDC.

**Depth**: The substructure of the solution may change across different tree depth. Therefore, it may be helpful to learn the dependence between the non-terminals and the depth of the current level. As an illustration, in Figure 3, the state of Depth context is 2 for the bottom layer of non-terminals so the choices of the three $Exp$ non-terminals will be conditioned on $Depth = 2$.

**Caller**: During the derivation of a parse tree, not only the derivation can be affected by other non-terminals within the same rule, but also affected by whom (the parent rule) calling the rule. Consider a non-terminal symbol $s$ and the subset of rules $Z \subseteq R$ containing $s$ in their RHS. We index these rules (starting from 0 to $|Z|-1$) by their order of occurrence in the grammar. Then, we subsequently annotate the non-terminals having symbol $s$ on the right-hand side of each rule in $Z$ using the index, namely the caller value. For example, considering the $Exp$ symbol in Figure 4, Rule 1.0, Rule 2.0 and Rule 2.1 are indexed by 0, 1 and 2 respectively with respect to the symbol. Next, the $Exp$ non-terminals on the right-hand side of Rule 1.0 (having index 0) are annotated by 0 as shown using superscript in $Exp^0$ and similarly for other rules. We repeat this for all the non-terminal symbols on the right-hand side of the grammar. Subsequently, each non-terminal is associated with a caller value.

**Figure 3**: Tree depth context.

**Figure 4**: Caller context.
4.5 Class Dependent Refinement

A BN classifier contains the original joint distribution of class $p$, but we do not follow it to generate individuals because it very likely leads to poor individuals. Therefore, we heuristically alter the probability distribution in the classifiers based on class type so as to raise the chance of generating useful structures of an individual to improve the fitness in the next generation. Now we propose a generation scheme to generate less poor ones as follows.

When we are about to derive a rule given the conditional distribution $p(w_i|p, r, pa(w_i))$, we set the most probable entry(ies) to a very small number (say 10$^{-6}$) and renormalize. Then, we sample from the renormalized conditional distribution. An example is shown in Table 2. This makes sense as the distribution is learnt from poor individuals and we can avoid it by suppressing the likelihood. This modification procedure can be applied to any types of problem in general if the semantics of the class label is known.

Table 2: Probability refinement assuming $\epsilon = 10^{-6}$

| Exp2 | Exp1 | $p(Exp2|Exp1, p)$ | $p'(Exp2|Exp1, p)$ |
|------|------|------------------|-------------------|
| ...  | ...  | ...              | ...               |
| 0    | 1    | 0.30             | $\epsilon$       |
| 1    | 1    | 0.30             | $\epsilon$       |
| 2    | 1    | 0.15             | 0.375             |
| 3    | 1    | 0.25             | 0.625             |
| ...  | ...  | ...              | ...               |

5. COMPARATIVE EXPERIMENTS

To demonstrate the effectiveness of our algorithm, we have implemented GBGPBC and tested it with two benchmark problems: the deceptive maximum (DMax) problems and the asymmetric royal tree (ART) problems. It was compared with three state-of-the-art GP methods: POLE[16, 17], PAGE-EM [27], and PAGE-VB[27]. These three approaches were shown to be superior to univariate model (PIPE), adjacent model (EDP model) and simple GP [27]. The parameters and their values of configuration on different problems are shown in Table 3. Due to time constraints, we assume an approach fails if the highest non-optimal fitness value of the individuals in the population keeps constant for ten generations or it cannot obtain the optimal solution in 200,000 fitness evaluations. We label GBGPBC with depth when depth context variable is used and similarly for other variables. Besides, ‘-’ is used if more than one variable is used. For example, caller-depth indicates that caller and depth variables are used. If none of the context variables is used, we call it a plain GPGPBC.

5.1 Deceptive Max

The DMax problem [17] is a modified version of the original Max problem [38]. Given a terminal set and a function set, the goal is to find a program which returns the maximum value within a depth limit. The DMax problem adds deceptiveness by introducing complex number in the terminal set but maximizing only the real part of the output of a program. Interaction between sub-trees must be considered in order to achieve this goal. Formally, the DMax problem has a function set $\{+m, \times n\}$ and a terminal set $\{0.95, \lambda\}$, where $\lambda' = 1$. We use the notation $DMax(m, r)$ to represent this class of problems. The grammar of $DMax(3, r)$ is shown in Figure 4. $DMax(5, 3)$ and $DMax(3, 2)$ are also called strong DMax and weak DMax respectively [27]. In this paper, we have tested four DMax problems with different combination of $m$ and $r$ (see Table 3). We follow the procedures in previous works [18][27] and report the average number of fitness evaluations needed to obtain the optimal solution with a probability of at least 90% in 50 independent runs under different configurations in Table 4.

It is observed that GBGPBC performs quite well in the DMax problems with depth and caller context elements. Its number of fitness evaluations is almost half of the second best approach PAGE-EM in DMax(3,2). For DMax(3,5) and DMax(3,4), GBGPBC performs slightly better than PAGE-EM and vice versa for DMax(5,3). The search performance of PAGE-VB is poorer than GBGPBC (being almost three times faster) at depth 5 because it needs to estimate the annotation size. Canonical GBGP cannot solve the DMax problems as the chance for obtaining an optimal solution drops by means of crossover and mutation. Using depth context element, GBGPBC is faster than POLE by at least two times. It also shows the importance of context elements in GBGPBC because the plain version fails to solve the DMax problem at a higher level when it does not consider where the derivation takes place. This is because the $Exp$ in Rule 2.0 has to be derived using both Rule 2.0 or Rule 2.1.

5.2 Asymmetric Royal Tree Problem

This is a variation of the traditional royal tree problem [39] where all the siblings of a node in the perfect tree have to be the same. Therefore, the optimal choice of a node depends only on the parent. In order to test the
Table 4: Results for Deceptive Max Problem. The table shows the statistics of the number of fitness evaluation needed under different configurations in 50 runs. For PAGE-EM, only the best results were shown. Numbers enclosed in brackets mean it fails to obtain the optimal solution with a probability of at least 90% in 50 independent runs. ‘X’ means none of the runs succeeds. '-' means results are unavailable.

<table>
<thead>
<tr>
<th>Problem</th>
<th>DMax(3,2)</th>
<th>DMax(3,5)</th>
<th>DMax(3,4)</th>
<th>DMax(5,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 4</td>
<td>Level 5</td>
<td>Level 4</td>
<td>Level 5</td>
<td>Level 4</td>
</tr>
<tr>
<td>µ</td>
<td>σ</td>
<td>µ</td>
<td>σ</td>
<td>µ</td>
</tr>
<tr>
<td>GBGPBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-plain</td>
<td>6,697</td>
<td>2,273</td>
<td>(141,979)</td>
<td>(25,347)</td>
</tr>
<tr>
<td>-depth</td>
<td>3,640</td>
<td>1,155</td>
<td>13,017</td>
<td>5,729</td>
</tr>
<tr>
<td>-caller</td>
<td>4,592</td>
<td>1,273</td>
<td>(155,842)</td>
<td>(20,977)</td>
</tr>
<tr>
<td>-depthcaller</td>
<td>4,487</td>
<td>1,214</td>
<td>8,977</td>
<td>2,296</td>
</tr>
<tr>
<td>GBGPBC</td>
<td>4,098</td>
<td>1,814</td>
<td>13,017</td>
<td>5,729</td>
</tr>
<tr>
<td>POLE</td>
<td>7,960</td>
<td>3,231</td>
<td>285,652</td>
<td>14,535</td>
</tr>
<tr>
<td>PAGE-EM</td>
<td>5,450</td>
<td>1,214</td>
<td>10,702</td>
<td>5,314</td>
</tr>
<tr>
<td>PAGE-VB</td>
<td>8,304</td>
<td>3,967</td>
<td>18,306</td>
<td>7,812</td>
</tr>
</tbody>
</table>

Figure 5: Examples of asymmetric royal tree. (a) A perfect tree at level A. Fitness = $2 \times (2 \times 2 \times 2 \times 2) = 8$. (b) A perfect tree at level B. Fitness = $2 \times (2 \times 2 \times 2 \times 2 \times 2 \times 2) = 32$. (c) Imperfect tree at level B. Fitness = $2 \times 2 \times 2 \times 2 = 16$. (d) Imperfect tree at level C. Fitness = $1 \times 1 \times 1 \times 1 \times 1 = 1$.

6. DISCUSSION AND CONCLUSION

From the experiments, GBGPBC has been shown to be competitive among the tested approaches in the problems requiring strong dependence. It has several advantages when compared with the previous approaches in the following ways.

6.1 Classification on Individuals

Our method combines probabilistic classification (the TAN classifier) with PMBG approach. The probability models are learnt from the selected good individuals for discovering the frequently occurred substructures in the good individuals, but these substructures may not be useful as some of them may commonly appear in both good individuals and poor individuals. In other words, discriminative substructures are more preferred. Rather than discarding the poor individuals, we select part of them to train the BN classifiers and store the discriminative information between the two classes of individuals. When we have the BN classifiers, it is easy to produce a good individual based on the joint distribution by conditioning on the good class. In this work, we only explore the use of good and poor classes, and it is possible to extend our approach by having other ways to classify the individuals.
6.2 Semantic Refinement on Distribution

Previous works such as EBCOA [11] generate individuals directly using the BN classifier learnt from both good and poor individuals. But, if we follow the distribution of poor individuals, it is likely to generate a poor one, so we modify the distribution to avoid the generation of poor individuals. In essence, the generation procedure of a given class should be refined according to the meaning of the class label.

6.3 Derivation Order Learning

Non-terminals in a derivation rule are usually inter-dependent. Tanev’s approach [32] accumulates context information using the derived terms but sampling enough data can be a problem. Besides, it implicitly assumed that the derivation is in a specific order (say in a depth-first search manner and from left-to-right). On the contrary, GBGPBC decides the order from the dependence of instantiating the non-terminals. Some non-terminals can be instantiated first if it does not depend on other non-terminals. Subsequently, other non-terminals can be instantiated.

6.4 Future Works

Our novel integration with BN classifiers and GBGP together with semantic refinement gives a new insight of PMBGA and PMBGP research to handle the dependence issues. To extend our approach, it may be interesting to apply other kinds of BN classifiers, especially those which can discover latent classes in the training samples. Furthermore, we would also like to develop problem specific semantic refinement in the future by identifying key features of a program in a certain class. Applying our GBGPBC system to real world problems will be promising. It is fast and scalable to large optimization problems in which the structure of solution can be defined using grammar.

7. ACKNOWLEDGMENTS

This research has been supported by General Research Fund LU310111 from the Research Grant Council of the Hong Kong Special Administrative Region.

8. REFERENCES