Probabilistic Model Building and Competent Genetic Programming

Kumara Sastry
David E. Goldberg

IlliGAL Report No. 2003013
April, 2003

Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue, Urbana, IL 61801

Probabilistic Model Building and Competent Genetic Programming

Kumara Sastry, David E. Goldberg
Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois, Urbana-Champaign, IL
104 S. Mathews Ave, Urbana, IL 61801

Abstract

This paper describes a probabilistic model building genetic programming (PMBGP) developed based on the extended compact genetic algorithm (eCGA). Unlike traditional genetic programming, which use fixed recombination operators, the proposed PMBGA adapts linkages. The proposed algorithms, called the extended compact genetic programming (eCGP) adaptively identifies and exchanges non-overlapping building blocks by constructing and sampling probabilistic models of promising solutions. The results show that eCGP scales-up polynomially with the problem size (the number of functionals and terminals) on both GP-easy problem and boundedly difficult GP-hard problem.

1 Introduction

In the last few decades, there has been a significant progress in the theory, design and development of genetic algorithms (GAs). A design methodology for understand the mechanics of genetic algorithms and developing better genetic and evolutionary algorithms in a principled manner has been proposed (Goldberg, Deb, & Clark, 1992). Based on the design decomposition, different facets of selectorecombinative GAs have been modeled using simple dimensional and facetwise models. These models have not only enabled a better understanding of the complex mechanisms of GAs, but also have provided GA practitioners with guidelines for tuning GA parameters. Finally, the design decomposition has led to the creation of *competent genetic algorithms*—genetic algorithms that solve hard problems, *quickly*, *reliably*, and *accurately* (Goldberg, 1999; Goldberg, 2002). The competent GAs have been very effective in solving a wide variety of problems—including hierarchical problems—of *bounded* difficulty.

On the other hand, even though the growth in application of genetic programming (GP) to problems of practical and scientific importance has been remarkable (Koza, 1992; Spector, Langdon, O'Reilly, & Angeline, 1999), there has been limited attention given to the development of competent operators that adapt linkage. Most of the studies on GP employ fixed crossover operators such as sub-tree crossover. Analysis of fixed recombination operators in selectorecombinative GAs suggest that fixed operators had highly inadequate and suggest operators that adapt linkage are essential for solving GA-hard problems in tractable time (Thierens & Goldberg, 1993; Thierens, 1999; Sastry & Goldberg, 2003).

Therefore, the purpose of this paper is to develop a competent GP—called the extended compact genetic programming (eCGP)—that adaptively identifies and propagates important subsolutions of a search problem. The proposed algorithm is similar to the probabilistic incremental program evolution (PIPE) (Sałustowicz & Schmidhuber, 1997). Unlike PIPE, our algorithm is not only

capable of handling univariate variable interactions, but also can capture fairly complex multivariate interactions among the variables of the search problem. The multivariate interactions are captured in a similar fashion as in the extended compact genetic algorithm (eCGA) (Harik, 1999a). Initial scalability results show that eCGP solves problems of bounded difficulty in polynomial time, as opposed to exponential time required by simple GP.

This paper is structured as follows. The next section presents a brief review of related work followed by a synopsis of extended compact genetic algorithm in section 3. The key features of the proposed algorithm (eCGP) are described in section 4 followed by an outline of test problems used to compare the performance of eCGP and simple GP in section 5. Section 6 describes the results followed by a note on the future work and key conclusions.

2 Literature Review

The last few decades have witnessed great strides in the development of so-called *competent* genetic algorithms—GAs that solve hard problems, quickly, reliably, and accurately (Goldberg, 1999). Since the development of the *messy genetic algorithm* (Goldberg, Korb, & Deb, 1989), a number of competent GAs have been constructed using different mechanisms and can be broadly classified into three classes:

Perturbation techniques include the fast messy GA (fmGA) (Goldberg, Deb, Kargupta, & Harik, 1993), gene expression messy GA (GEMGA) (Kargupta, 1996), and linkage identification by nonlinearity check GA (LINC GA), and linkage identification by monotonicity detection GA (LIMD GA) (Munetomo & Goldberg, 1999).

Linkage adaptation techniques such as linkage learning GA (LLGA) (Harik & Goldberg, 1997; Harik, 1999b; Chen & Goldberg, 2002).

Probabilistic model building techniques such as population-based incremental learning (PBIL) (Baluja, 1994), the bivariate marginal distribution algorithm (BMDA) (Pelikan & Mühlenbein, 1999), the extended compact GA (eCGA) (Harik, 1999a), iterated distribution estimation algorithm (IDEA) (Bosman & Thierens, 1999), Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 2000), and hierarchical Bayesian optimization algorithm (hBOA) (Pelikan & Goldberg, 2001). For further details on probabilistic model building GAs, the interested reader should refer elsewhere (Pelikan, 2002b; Pelikan, 2002a; Larrañaga & Lozano, 2002).

While most of the above competent GAs operate on binary and real valued variables, there exist very few competent genetic programming designs that operate on program trees. An exception is the probabilistic incremental program evolution (PIPE) (Sałustowicz & Schmidhuber, 1997). PIPE is a univariate probabilistic model building technique and is based on Baluja's PBIL (Baluja, 1994). In PIPE computer programs or mathematical functions are evolved as in genetic programming. Programs are represented by trees where each internal node represents a function/instruction and leaves represent terminal. In PIPE, probabilistic representation of the program trees is used and the probabilities of each instruction in each node in a maximal possible tree are used to model promising and generate new programs.

Similar to other univariate PMBGAs, PIPE can only handle problems where there is no interactions among variables. However, there are a large class of problems where variables do interact and it is essential to capture those interactions and the design operators that respect these higher-order interactions. This study uses ECGA to adaptively model higher-order interactions among variables (nodes of the program tree).

3 Extended Compact Genetic Algorithm (eCGA)

The extended compact GA proposed by Harik (1999a) is based on a key idea that the choice of a good probability distribution is equivalent to linkage learning. The measure of a good distribution is quantified based on minimum description length (MDL) models. The key concept behind MDL models is that all things being equal, simpler distributions are better than more complex ones. The MDL restriction penalizes both inaccurate and complex models, thereby leading to an optimal probability distribution. Thus, MDL restriction reformulates the problem of finding a good distribution as an optimization problem that minimizes both the probability model as well as population representation. The probability distribution used in eCGA is a class of probability models known as marginal product models (MPMs). MPMs are formed as a product of marginal distributions on a partition of the genes and are similar to those of the compact GA (CGA) (Harik, Lobo, & Goldberg, 1998) and PBIL (Baluja, 1994). Unlike the models used in CGA and PBIL, MPMs can represent probability distributions for more than one gene at a time. MPMs also facilitate a direct linkage map with each partition separating tightly linked genes. For example, the following MPM, `[1,3][2][4]`, for a four-bit problem represents that the 1^{st} and 3^{rd} genes are linked and 2^{nd} and 4^{th} genes are independent. Furthermore, the MPM consists of the following marginal probabilities $\{p(x_1 = 0, x_3 = 0)$, $p(x_1 = 0, x_3 = 1)$, $p(x_1 = 1, x_3 = 0)$, $p(x_1 = 1, x_3 = 1)$, $p(x_2 = 0)$, $p(x_2 = 1)$, $p(x_4 = 0)$, $p(x_4 = 1)\}$, where x_i is the value of the i^{th} gene.

The eCGA can be algorithmically outlined as follows:

1. Initialization: The population is usually initialized with random individuals. However, other initialization procedure can also be used.
2. Evaluate the fitness value of the individuals
3. Selection: The eCGA uses s-wise tournament selection (Goldberg, Korb, & Deb, 1989). However, other selection procedures can be used instead of tournament selection.
4. Build the probabilistic model: In eCGA, both the structure and the parameters of the model are searched. A greedy search heuristic is used to find an optimal model of the selected individuals in the population.
5. Create new individuals: In eCGA, new individuals are created by sampling the probabilistic model.
6. Replace the parental population with the offspring population.
7. Repeat steps 2–6 until some convergence criteria are met.

Two things need further explanation, one is the identification of MPM using MDL and the other is the creation of a new population based on MPM. The identification of MPM in every generation is formulated as a constrained optimization problem,

$$\begin{aligned} &\text{Minimize} && C_m + C_p && (1)\\ &\text{Subject to} && && \\ & && \chi^{k_i} \leq n \quad \forall i \in [1, m]. && (2) \end{aligned}$$

where C_m is the model complexity which represents the cost of a complex model.In essence, the model complexity, C_m, quantifies the model representation size in terms of number of bits required to store all the marginal probabilities. Let, a given problem of size ℓ with alphabet cardinality χ,

have m partitions with k_i genes in the i^{th} partition, such that $\sum_{i=1}^{m} k_i = \ell$. Then each partition i requires $\chi^k - 1$ independent frequencies to completely define its marginal distribution. Furthermore, each frequency is of size $\log_2(n)$, where n is the population size. Therefore, the model complexity (or the model representation size), C_m, is given by

$$C_m = \log_2(n) \sum_{i=1}^{m} \left(\chi^{k_i} - 1 \right). \tag{3}$$

The compressed population complexity, C_p, represents the cost of using a simple model as against a complex one. In essence, the compressed population complexity, C_p, quantifies the data compression in terms of the entropy of the marginal distribution over all partitions. Therefore, C_p is evaluated as

$$C_p = n \sum_{i=1}^{m} \sum_{j=1}^{\chi^{k_i}} -p_{ij} \log_2 \left(p_{ij} \right) \tag{4}$$

where p_{ij} is the frequency of the j^{th} gene sequence of the genes belonging to the i^{th} partition. In other words, $p_{ij} = N_{ij}/n$, where N_{ij} is the number of chromosomes in the population (after selection) possessing bit-sequence $j \in [1, \chi^{k_i}]$ [1] for i^{th} partition. The constraint (Equation 2) arises due to finite population size.

The following greedy search heuristic is used to find an optimal or near-optimal probabilistic model:

1. Assume each variable is independent of each other. The model is a vector of probabilities.
2. Compute the model complexity and population complexity values of the current model.
3. Consider all possible $\frac{1}{2}\ell(\ell - 1)$ merges of two variables.
4. Evaluate the model and compressed population complexity values for each model structure.
5. Select the merged model with lowest combined complexity.
6. If the combined complexity of the best merged model is better than the combined complexity of the model evaluated in step 2., replace it with the best merged model and go to step 2.
7. If the combined complexity of the best merged model is less than or equal to the combined complexity, the model cannot be improved and the model of step 2. is the probabilistic model of the current generation.

A new population is generated based on the optimal MPM as follows: A population of size $n(1 - P_c)$ where P_c is the crossover probability, is filled by the best individuals in the current population. The remaining nP_c individuals are generated by randomly choosing subsets from the current individuals according to the probabilities of the subsets as calculated in the probabilistic model.

[1] Note that a BB of length k has χ^k possible sequences where the first sequence denotes be $00\cdots0$ and the last sequence $(\chi - 1)(\chi - 1)\cdots(\chi - 1)$

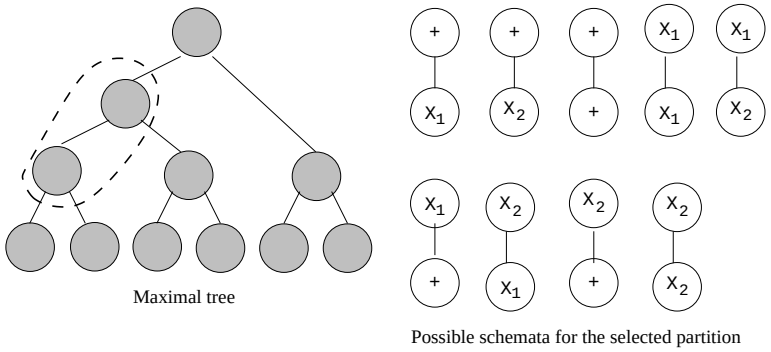

Figure 1: Illustration of all possible subtrees in a given partition.

4 Extended Compact Genetic Programming (eCGP)

The previous section outlined the key features of extended compact genetic algorithm. This section combines the features of eCGA (Harik, 1999a) and PIPE (Sałustowicz & Schmidhuber, 1997) to create a multivariate probabilistic model building genetic programming. The proposed algorithm, called extended compact genetic programming (eCGP), adaptively identifies both the building blocks and their structure. eCGP also exchanges the building blocks from different partitions effectively, and therefore drastically improves BB mixing when compared to fixed recombination operators. The eCGP can be viewed as an extension of both ECGA and PIPE. While eCGA operates on only fixed-length bitstrings, eCGP operates on variable-size program trees. While PIPE uses only fixed-structure univariate probabilistic models to create new offspring, eCGP utilizes adaptive-structure multivariate probabilistic models.

The eCGP algorithm is similar to that of eCGA described in the previous section (Section 3). Similar to PIPE, the probabilistic model is built on a maximal tree. It should be noted that we use a maximal tree only to build the probabilistic model. The trees that are generated from the probabilistic model need not, and indeed are not, maximal trees. Unlike PIPE which considers each node in the tree to be independent of each other, eCGP accommodates multivariate interactions between nodes. In other words, eCGP decomposes or partitions the maximal tree into subtrees and simultaneously builds the probabilistic models for each subtree. Therefore, eCGP not only searches for the structure of the probabilistic model, but also learns the parameters of the model. The structure of the model, similar to eCGA, is MPM, and the parameters are the respective frequencies of all possible subtrees in a given partition. For example, figure 1 shows all possible subtrees, for a 3-variable problem (1 functional, 2 terminals) (Sastry, O'Reilly, Goldberg, & Hill, 2003).

A greedy search heuristic is used to find an optimal probabilistic model and the MDL metric is used as a measure of the model quality. Finally, the "best" model is used to sample new individuals (program trees). Similar to PIPE, a maximal tree is created using the probabilistic model for each offspring, and unused portions of the tree are pruned before the evaluation of the candidate solution. We emphasize once more that even though we sample new offspring from maximal-tree probabilistic model, the final pruned tree need not be a maximal tree. This is because, during the sampling process, both functionals and terminals are candidates and if a terminal is chosen at a particular internal node all the other nodes that are below it and connected to it are discarded during the pruning process.

5 Test Problems

Our approach in designing pilot experiments for investigating the scalability of eCGP is to *design* bounding *adversarial problems* that exploit one or more dimensions of problem difficulty. Particularly, our pilot test problems should possess the following properties:

- Building-block identification should be critical for successful innovation. That is, if the BBs of the problem are not identified and exchanged, it should be impossible to attain the global solution.
- Building-block structure and interactions of the problem should be known to the researchers, but not to the problem solver (search method). Ensuring that the BB identification methods work on such problems, provides assurance that they would also identify BBs of real-world problems, where the BBs are not known a priori.
- The properties such as building-block size, and problem difficulty should be tunable without significantly changing the functional.

This adversarial and systematic design method contrasts sharply with the common practices of using historical, randomly generated, or ad hoc test functions (Goldberg, 2002).

In this study we employ two classes of problems: 1. OneMax-like GP-easy problem, and 2. Deceptive trap problem. They are respectively described in the following sections.

5.1 GP-Easy Problem: `ORDER`

`ORDER` is a GP version of the OneMax problem in GAs (Goldberg & O'Reilly, 1998; O'Reilly & Goldberg, 1998), and therefore is a GP-easy problem. The primitive set of `ORDER` consists of the primitive `JOIN` of arity two and complimentary primitive pairs $(X_i, \bar{X}_i)$, $i = 0, 1, \cdots, \ell$ of arity one. A candidate solution of the `ORDER` problem is a binary tree with `JOIN` primitive at the internal nodes and either X_i's or $\bar{X}_i$'s at its leaves. The candidate solution's output is determined by parsing the program tree inorder (from left to right). The program expresses the value X_i if, during the inorder parse, an X_i leaf is encountered before its complement $\bar{X}_i$ and neither X_i nor its complement are encountered earlier. That is, the program only expresses unique primitives during the inorder parse.

For each unique X_i (or $\bar{X}_i$) that a program expresses, an equal unit of fitness value is accredited. That is,

$$f_1(x_i) = \begin{cases} 1 & if x_i \in \{X_1, X_2, \cdots, X_\ell\} \\ 0 & otherwise \end{cases} . \tag{5}$$

The fitness function for `ORDER` is then defined as

$$F(\mathbf{x}) = \sum_{i=1}^{\ell} f_1(x_i) \tag{6}$$

where $\mathbf{x}$ is the set of primitives output by the program. The expression for optimal solution of a ℓ-primitive `ORDER` problem is $\{X_1, X_2, \cdots, X_\ell\}$, and its fitness value is ℓ.

For example, consider a candidate solution for a 4-primitive `ORDER` problem as shown in figure 2. The sequence of leaves for the tree is $\{X_1, \bar{X}_1, \bar{X}_1, X_4, X_1, \bar{X}_2\}$, the expression during inorder parse is $\{X_1, \bar{X}_2, X_4\}$, and its fitness is 2.

For more details, motivations, and analysis of the `ORDER` problem, the interested reader should refer elsewhere (Goldberg & O'Reilly, 1998; O'Reilly & Goldberg, 1998)

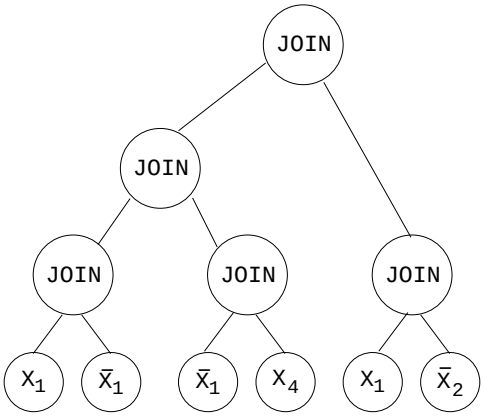

Figure 2: A candidate solution for a 4-primitive ORDER problem. The output of the program is $\{X_1, \bar{X}_2, X_3\}$ and its fitness is 2.

5.2 GP-Hard Problem: Deceptive Trap

Another test problem used in this study is the deceptive *trap* function (Deb & Goldberg, 1993; Deb & Goldberg, 1994) which consists of additively separable *deceptive* functions (Goldberg, 1987). Deceptive functions are designed to thwart the very mechanism of selectorecombinative search by punishing any localized hillclimbing and requiring mixing of whole building blocks at or above the order of deception. Using such *adversarially* designed functions is a stiff test—in some sense the stiffest test—of algorithm performance. The idea is that if an algorithm can beat an adversarially designed test function, it can solve other problems that are equally hard or easier than the adversary. Furthermore, if the building blocks of such deceptive functions are not identified and respected by a selectorecombinative GAs, then they almost always converge to the local optimum.

The expression mechanism of the program for deceptive trap function is identical to that of ORDER, and the difference is in the fitness evaluation procedure. Unlike ORDER, a deceptive trap function divides the expressed primitives into subgroups of k-primitives, and the fitness value for a k-primitive subgroups is defined as follows:

$$f_k\left(u(x_1, x_2, \cdots, x_k)\right) = \begin{cases} 1.0 & u = k \\ (1.0 - \delta)\left(1 - \frac{u}{k-1}\right) & u < k \end{cases}, \tag{7}$$

where u is the unitation, or the number of primitives, X_i, in a portion of the tree:

$$u\left(x_1, x_2, \cdots, x_k\right) = \sum_{i=1}^{k} f_1(x_i), \tag{8}$$

where, x_i is the ith unique primitive, δ is the difference in the functional value between the good BB (all ones) and its deceptive attractor (all zeros). The difficulty of a trap function can be adjusted by modifying the values k, and δ. The problem becomes more difficult as the value of k is increased and that of δ is decreased. A 4-primitive deceptive trap function is illustrated in figure 3.

The fitness function of the trap function is then defined as

$$\begin{aligned} F\left(\mathbf{x}\right) \quad = \quad & f_k\left(u(x_1, x_2, \cdots, x_k)\right) + f_k\left(u(x_{k+1}, \cdots, x_{2k})\right) + \\ & \cdots + f_k\left(u(x_{(m-1)k+1}, \cdots, x_{mk})\right), \end{aligned} \tag{9}$$

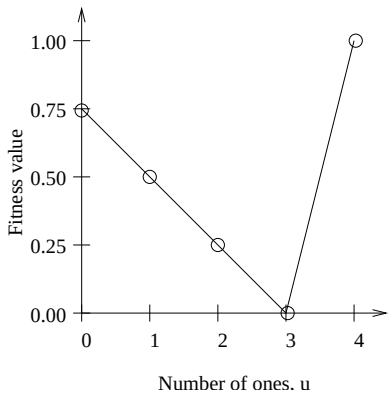

Figure 3: A Fully deceptive trap function with $k = 4$, and $\delta = 0.25$.

where, F is the fitness function, $\mathbf{x}$ is the expressed primitives (expressed leaf nodes of the tree), m is the number of BBs, and $\ell = mk$.

The important feature of additively separable trap functions is that if the good BB (all ones) in any particular partition is not identified, then the GA tends to converge to the deceptive attractor (all zeros) in that partition. Therefore, BB identification and mixing is critical to innovation success. Furthermore, notice that the problems are of bounded difficulty $k < \ell$. If the trap was of length ℓ, nothing could work better than enumeration or random search without replacement. However, given that the difficulty is bounded, a GA that identifies BBs and mixes them well has the opportunity to solve the problem in polynomial time (Goldberg, 2002).

6 Results and Discussion

This section compares the performance of eCGP and a simple GP for both `ORDER` and and 3-primitive deceptive trap function (k = 3). Specifically, we investigate how eCGP and simple GP scale-up with the problem size (number of terminals and functionals) for both `ORDER`, which is a GP-easy problem, and deceptive trap function which is a boundedly GP-hard problem.

A simple GP consists of a tree-swap crossover and s-wise tournament selection. Crossover probability of 1.0 and tournament size of 2 was used for both eCGP and simple GA. Mutation is not considered in this study. The initial population for both eCGP and simple GP were generated using the ramped half-and-half method and maximum tree depth of $2 \log_2 \ell$ was used. A GP run was stopped when the fitness variance of the population was less than ℓ^{-2} for `ORDER` and less than m^{-2} for the deceptive trap function. The number of function evaluations required in order to obtain a population with at least $m - 1$ correct building blocks are empirically computed. All the results presented in this section are averaged over 50 independent runs.

Figure 4 compares the scalability of eCGP and simple GP for the `ORDER`. The figure plots the number of function evaluations against problem size. The results indicate that even though eCGP requires a slightly larger number of function evaluations, the slope of the curve is slightly less than that for simple GP. This result is consistent with the performance of competent GAs on GA-easy problems (Pelikan, Goldberg, & Cantú-Paz, 2000; Pelikan, 2002a) and is caused by the spurious dependencies that are introduced by the selection mechanism and fixed population size (Pelikan, Goldberg, & Sastry, 2001; Pelikan, Sastry, & Goldberg, 2003). However, as the problem size increases, simple GP will suffer from mixing effects (Sastry & Goldberg, 2003) and the slope of the number of function evaluations will increase.

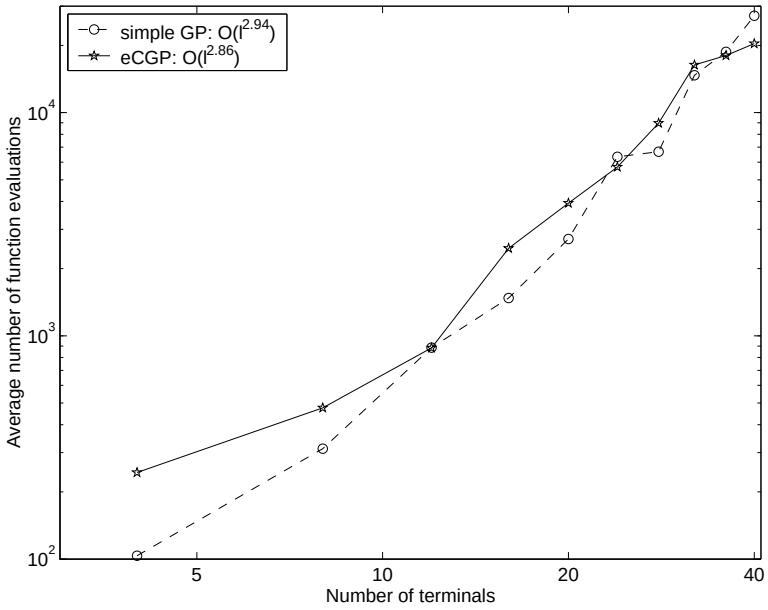

Figure 4: Comparison of eCGP and simple GP for ORDER. The figure plots the average number of function evaluations as a function of problem size. The results are averaged over 50 independent runs.

Figure 5 compares the scalability of the eCGP and the simple GP for the deceptive trap problem. The building block size, k is set to 3 and the signal difference, d is set to 0.25. The figure plots the number of function evaluations as a function of problem size. The results clearly indicate the effectiveness of linkage-adaptive recombination operator of eCGP over the fixed recombination operator of simple GP. The results indicate that eCGP scales-up polynomially (cubic) with problem size. Furthermore, the scalability of eCGP is about the same for both ORDER and deceptive trap function. Finally, the number of function evaluations required for simple GP appears to be very high for problems with more than 24 terminals, and therefore the problems become intractable quickly. For example, for deceptive trap problem with 30 terminals, the simple GP was not able to converge to the optimal solution even after 5×10^8 evaluations. However, for the problems sizes that simple GP was able to converge to the optimal solutions, the scale-up with problem size is $O(\ell^9)$.

We recognize that both ORDER and the deceptive trap problems have non-overlapping building blocks. That is, an expressed primitive is a part of only one building block. As mentioned earlier, eCGP partitions the search problem into linkage groups and therefore is very effective in identifying non-overlapping building blocks. Even though the success of eCGP in identifying, propagating, and exchanging of non-overlapping building blocks, enables it to solve a broad class of additively decomposable problems, we acknowledge that even broader class of problems can be tacked if complex interactions such as overlapping BBs and hierarchically interacting BBs can also be identified. The successful adaptation of eCGA into genetic programming domain, makes the adaptation of other powerful competent GAs such as the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 2000) and the hierarchical Bayesian optimization algorithm (hBOA)(Pelikan & Goldberg, 2001) into GP domain straightforward and promising.

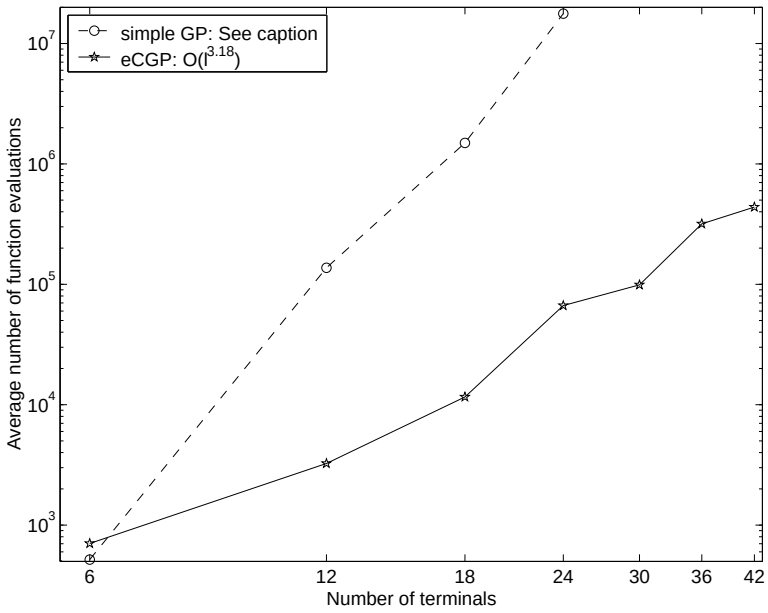

Figure 5: Comparison of eCGP and simple GP for the deceptive trap function (k = 3). The figure plots the average number of function evaluations as a function of problem size. The results are averaged over 50 independent runs. The simple GP was not able to solve problems beyond 24 primitives. For example, for a deceptive trap function with 30 primitive, simple GP did not converge to the optimal solution even after 5×10^8 function evaluations. For the problem sizes that we were able to obtain the optimal solution via a simple GP, the scale-up is $O(\ell^{8.91})$. But its failure to converge to the optimal solution for larger problems suggests that the scale-up is indeed much larger than $O(\ell^{8.91})$.

7 Summary, Future Work, and Conclusions

This paper developed a competent GP, called extended compact GP (eCGP) using the ideas of extended compact genetic algorithm (ECGA) and probabilistic incremental program evolution (PIPE). The proposed algorithm adaptively identifies, propagates, and exchanges important subsolutions of a search problem. The subsolutions are identified by building a multivariate probabilistic model of promising solutions and the subsolutions are exchanged by creating new offspring by sampling the probabilistic model. The results of eCGP was compared to those of a simple GP for two test problems: 1. `ORDER`: A GP-easy problem, and 2. Deceptive trap: A GP-hard problem. The results show that eCGP scales up polynomially with the problem size (number of terminals) for both GP-easy and GP-hard problems. On the other hand, as expected, a simple GP scales-up polynomially for a GP-easy problem, and exponentially for GP-hard problems.

This paper presents initial results of eCGP, and there are many avenues of future work, some of which are listed as follows:

- The eCGP can only identify non-overlapping BBs, and we need to develop competent operators that not only identify complex BB structures, but also tackle BBs with hierarchical interactions.

- Two adversarially designed test problems were considered in this study, and more tests have to be conducted on different class of problems and the scalability of eCGP has to be analyzed.

- We need to investigate the convergence time and population-sizing requirement of eCGP, both analytically through facetwise and dimensionless models, and also empirically for different classes of problems.
- This study considered problems with only one functional and many more terminals. However, many GP problems consider many functionals and only a few terminals (assuming the ephemeral random constant is one type of terminal). The performance of eCGP on such a class of problems needs to be investigated.

Acknowledgments

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-0163 and F49620-03-1-0129, and the National Science Foundation under grant DMI-9908252, and CSE fellowship, UIUC. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

References

Baluja, S. (1994). *Population-based incremental learning: A method of integrating genetic search based function optimization and competitive learning* (Technical Report CMU-CS-94-163). Carnegie Mellon University.

Bosman, P., & Thierens, D. (1999). Linkage information processing in distribution estimation algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference*, 60–67.

Chen, Y.-P., & Goldberg, D. E. (2002). Introducing start expression genes to the linkage learning genetic algorithm. *Parallel Problem Solving from Nature*, *7*, 351–360. (Also IlliGAL Report No. 2002007).

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. *Foundations of Genetic Algorithms*, *2*, 93–108. (Also IlliGAL Report No. 91009).

Deb, K., & Goldberg, D. E. (1994). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, *10*, 385–408. (Also IlliGAL Report No. 92001).

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In Davis, L. (Ed.), *Genetic algorithms and simulated annealing* (Chapter 6, pp. 74–88). Los Altos, CA: Morgan Kaufmann.

Goldberg, D. E. (1999). The race, the hurdle, and the sweet spot: Lessons from genetic algorithms for the automation of design innovation and creativity. In Bentley, P. (Ed.), *Evolutionary Design by Computers* (Chapter 4, pp. 105–118). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Boston, MA: Kluwer Academic Publishers.

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*(4), 333–362. (Also IlliGAL Report No. 91010).

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms*, 56–64. (Also IlliGAL Report No. 93004).

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3*(5), 493–530. (Also IlliGAL Report No. 89003).

Goldberg, D. E., & O'Reilly, U.-M. (1998). Where does the good stuff go, and why? How contextual semantics influences program structure in simple genetic programming. (pp. 16–36).

Harik, G. (1999a, January). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL Report No. 99010). Urbana, IL: University of Illinois at Urbana-Champaign.

Harik, G., & Goldberg, D. E. (1997). Learning linkage. *Foundations of Genetic Algorithms*, *4*, 247–262. (Also IlliGAL Report No. 96006).

Harik, G., Lobo, F., & Goldberg, D. E. (1998). The compact genetic algorithm. pp. 523–528. (Also IlliGAL Report No. 97006).

Harik, G. R. (1999b). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Doctoral dissertation, University of Michigan, Ann Arbor, MI. (Also IlliGAL Report No. 97005).

Kargupta, H. (1996). The gene expression messy genetic algorithm. *Proceedings of the International Conference on Evolutionary Computation*, 814–819.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Larrañaga, P., & Lozano, J. A. (Eds.) (2002). *Estimation of distribution algorithms*. Boston, MA: Kluwer Academic Publishers.

Munetomo, M., & Goldberg, D. E. (1999). Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, *7*(4), 377–398.

O'Reilly, U.-M., & Goldberg, D. E. (1998). How fitness structure affects subsolution acquisition in genetic programming. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 269–277.

Pelikan, M. (2002a). *Bayesian optimization algorithm: From single level to hierarchy*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL.

Pelikan, M. (2002b). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*, 5–20. (Also IlliGAL Report No. 99018).

Pelikan, M., & Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference*, 511–518.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000). Linkage learning, estimation distribution, and Bayesian networks. *Evolutionary Computation*, *8*(3), 314–341.

Pelikan, M., Goldberg, D. E., & Sastry, K. (2001). Bayesian optimization algorithm, decision graphs, and Occam's razor. *Proceedings of the Genetic and Evolutionary Computation Conference*, 519–526. (Also IlliGAL Report No. 2000020).

Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In Roy, R., Furuhashi, T., & Chawdhry, P. K. (Eds.), *Advances in Soft Computing - Engineering Design and Manufacturing* (pp. 521–535). London: Springer-Verlag.

Pelikan, M., Sastry, K., & Goldberg, D. E. (2003). Scalability of the Bayesian optimization algorithm. *International Journal of Approximate Reasoning*, *31*(3), 221–258. (Also IlliGAL Report No. 2002024).

Sałustowicz, R. P., & Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, *5*(2), 123–141.

Sastry, K., & Goldberg, D. E. (2003). Scalability of simple selectorecombinative genetic algorithms on problems with tight linkage. *Proceedings of the 2003 Genetic and Evolutionary Computation Conference*, –. (Also IlliGAL Report No. 2002013).

Sastry, K., O'Reilly, U.-M., Goldberg, D. E., & Hill, D. (2003). Building-block supply in genetic programming. In *Proceedings of the Genetic Programming Workshop on Theory & Practice* (Chapter 9, pp. –). (Also IlliGAL Report No. 2003011).

Spector, L., Langdon, W. B., O'Reilly, U.-M., & Angeline, P. J. (Eds.) (1999). *Advances in genetic programming 3*. Cambridge, MA: MIT Press.

Thierens, D. (1999). Scalability problems of simple genetic algorithms. *Evolutionary Computation*, *7*(4), 331–352.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proceedings of the International Conference On Genetic Algorithms*, 38–45.