Scalability of Genetic Programming and Probabilistic Incremental Program Evolution

Radovan Ondas
University of Missouri - St. Louis
Dept. of Math and Computer Science, CCB 331
8001 Natural Bridge Rd.
St. Louis, MO 63121, USA
ondasr@umsl.edu

Martin Pelikan
University of Missouri - St. Louis
Dept. of Math and Computer Science CCB 320
8001 Natural Bridge Rd.
St. Louis, MO 63121, USA
pelikan@cs.umsl.edu

Kumara Sastry
University of Illinois at Urbana-Champaign
Dept. of General Engineering
117 Transportation Bldg.
104 S. Mathews Ave.
Urbana, IL 61801, USA
ksastry@uiuc.edu

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search;
I.2.6 [**Artificial Intelligence**]: Learning;
G.1.6 [**Numerical Analysis**]: Optimization

General Terms

Algorithms, Performance

Keywords

Genetic programming, PIPE, scalability, order problem, trap problem

1. INTRODUCTION

To solve large and complex problems, scalability is among the primary concerns of an optimization practitioner. However, only few studies [6] exist that study scalability in genetic programming (GP) [1]. The same holds for simple approaches to using probabilistic recombination in GP within the estimation of distribution algorithm (EDA) [3, 4, 2] framework, such as the probabilistic incremental program evolution (PIPE) [5].

The purpose of this poster is to show the scalability of standard GP and PIPE on two decomposable GP problems: `ORDER` and `TRAP` [6]. The two algorithms perform as expected and they solve `ORDER` scalably while failing to scale up on `TRAP`. Additionally, the poster studies the effects of introducing unnecessary and irrelevant primitives.

2. METHODS AND TEST PROBLEMS

Both GP and PIPE work with programs encoded as labeled-tree structures and both can be applied to the same class of problems. While GP [1] generates new candidate programs using standard variation operators, such as crossover and mutation, PIPE [5] builds and samples a probabilistic model in the form of a tree of mutually independent nodes. Therefore, the difference between GP and PIPE is in their variation operator.

In order to test scalability, we need a class of problems where size can be modified while the inherent problem difficulty does not grow prohibitively fast. Two types of decomposable problems for fixed-length string GAs are common: Onemax and concatenated traps. Similar problems were also created for GP where candidate solutions are represented by program trees [6].

We consider two classes of problems from [6]: `ORDER` (a onemax-like, GP-easy problem) and `TRAP` (a deceptive-trap-like, GP-difficult problem). In addition to standard `ORDER` and `TRAP`, we tested GP and PIPE on `ORDER` with `JUNK` terminals (unexpressed terminals). Junk-code or `JUNK` terminals represent unnecessary primitives that are irrelevant for the particular problem. In biological terms, `JUNK` terminals correspond to the junk code in DNA.

3. RESULTS

The scalability of GP and PIPE was tested on three classes of problems: Basic `ORDER` (no `JUNK`), basic `TRAP` (no `JUNK`), and `ORDER` with `JUNK` terminals, where the number of unique `JUNK` terminals was set to $l/5$. The scalability experiments were performed by testing both algorithms on problem instances with an increasing number of primitives.

Binary tournament selection was used in both GP and PIPE. The probability of crossover in GP was set to 1.0. To focus on the effects of recombination, no mutation was used. The initial population in both methods was generated using the standard half-and-half method. Maximum tree depth was set to be one more than the depth of the minimum tree to store the global optimum. The population size was determined using a bisection method so that it was within 10% of the minimum population size required to successfully solve 30 independent runs. The runs were terminated when the algorithms found the global optimum or when the number of generations was too large for the particular problem.

Figure 1 shows the scalability of GP and PIPE on `ORDER` without `JUNK` terminals. The results indicate that PIPE is slightly more efficient than GP but both GP and PIPE

GECCO'05, June 25–29, 2005, Washington, DC, USA.
ACM 1-59593-010-8/05/0006.

scale up with a low-order polynomial. These results are in agreement with the behavior observed in binary-string GAs on the simple onemax problem.

Figure 2 compares the scalability of GP and the PIPE on TRAP without JUNK terminals. On TRAP, GP performs slightly better than PIPE. This can be explained by its weaker recombination operator because here recombination causes disruption of important partial solutions [7]. Nonetheless, both GP and PIPE scale up poorly and they indicate an exponential growth of the number of function evaluations with problem size.

Figure 3 compares the scalability of GP and PIPE on ORDER with $l/5$ unique JUNK terminals. Both GP and PIPE seem to be capable of dealing with these irrelevant terminals and achieve performance comparable to that on basic ORDER.

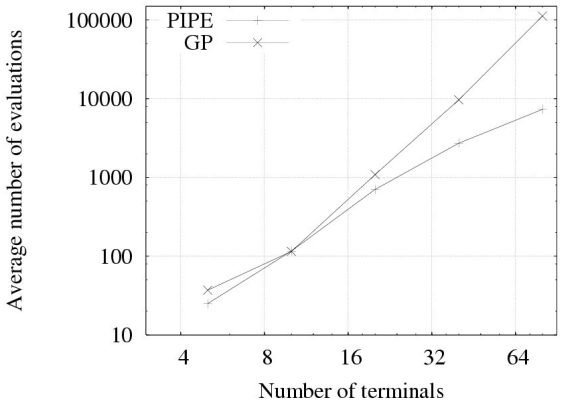

Figure 1: Scalability of GP and PIPE on ORDER.

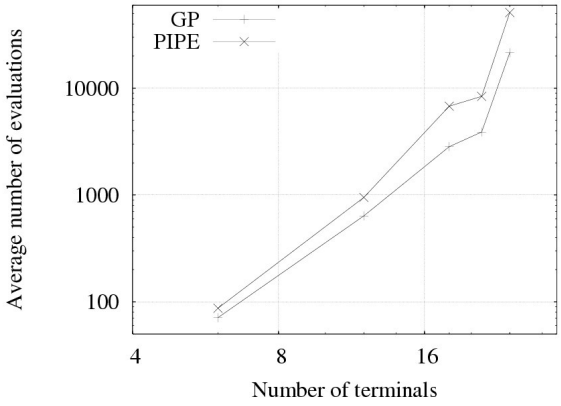

Figure 2: Scalability of GP and PIPE on TRAP.

4. CONCLUSIONS

The results presented in this poster indicate that the behavior of different variants of GP can be expected to be similar to that of standard binary-string GAs. There are two important consequences of this fact. First, as it was indicated in [6], to solve some classes of problems scalably, linkage learning may have to be incorporated into GP in order to identify and exploit interactions between different

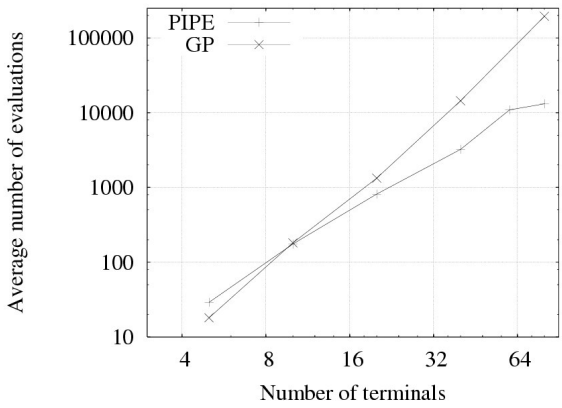

Figure 3: Scalability of GP and PIPE on ORDER with $l/5$ copies of JUNK terminals.

program components. Second, some lessons learned in the design and application of binary-string GAs should carry over to GP.

The results also indicate that if the recombination operator captures interactions in the problem properly, increasing the mixing effects of recombination leads to better performance. That is why PIPE outperformed standard GP on problems where program components could be treated independently. This fact together with the need for linkage learning should encourage the application of probabilistic recombination operators of estimation of distribution algorithms (EDAs) [3, 4, 2] to the domain of GP. Some representatives of EDAs applied to the GP domain are [5, 6]. Finally, the results show that both GP and PIPE can deal with irrelevant terminals relatively well and their performance gets only slightly worse when adding such primitives.

Acknowledgments

This work was partially supported by the Research Award and the Research Board at the University of Missouri.

5. REFERENCES

[1] J. R. Koza. *Genetic programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, 1992.

[2] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, Boston, MA, 2002.

[3] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, pages 178–187, 1996.

[4] M. Pelikan, D. E. Goldberg, and F. G. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.

[5] R. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.

[6] K. Sastry and D. E. Goldberg. Probabilistic model building and competent genetic programming. In R. Riolo and B. Worzel, editors, *Genetic Programming Theory and Practice*, pages 205–220. Kluwer, Boston, MA, 2003.

[7] D. Thierens. *Analysis and design of genetic algorithms*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 1995.