# Immediate Transfer of Global Improvements to All Individuals in a Population Compared to Automatically Defined Functions for the EVEN-5,6-PARITY Problems

Ricardo Aler

Universidad Carlos III de Madrid
Butarque 15
28911 Leganés (Madrid), España
aler@inf.uc3m.es
http://grial.uc3m.es/~aler

**Abstract.** Koza has shown how automatically defined functions (ADFs) can reduce computational effort in the GP paradigm. In Koza's ADF, as well as in standard GP, an improvement in a part of a program (an ADF or a main body) can only be transferred via crossover. In this article, we consider whether it is a good idea to transfer immediately improvements found by a single individual to the whole population. A system that implements this idea has been proposed and tested for the EVEN-5-PARITY and EVEN-6-PARITY problems. Results are very encouraging: computational effort is reduced (compared to Koza's ADFs) and the system seems to be less prone to early stagnation. Finally, our work suggests further research where less extreme approaches to our idea could be tested.

## 1 Introduction

In [4], Koza showed how "automatically defined functions enable genetic programming to solve a variety of problems in a way that can be interpreted as a decomposition of a problem into subproblems, a solving of the subproblems, and an assembly of the solutions to the subproblems into a solution to the overall problem". Also, he showed that "For a variety of problems, genetic programming requires less computational effort to solve a problem with automatically defined functions than without them, provided the difficulty of the problem is above a certain relatively low problem-specific breakeven point for computational effort".
With Koza's ADFs, each individual consists of both the main body of the program and of all its subroutines. An improvement in a subroutine (or a main body) of an individual can only be transferred to another individual via crossover between both individuals. Intuitively, it would

seem that if an individual can immediately use improvements obtained anywhere else in the population, then the rate of discovery would be higher. On the other hand, it might happen that an individual cannot use another's individual discovery because their structures are just too different. In that case, something could be an improvement for an individual but a hindrance to another and therefore, former good individuals would become instantly bad individuals. What would be the net effect of both tendencies is unclear. Thus, the aim of this article is to start exploring empirically this matter: what would happen if improvements[1] in a subroutine (or a program's main body) would be transferred immediately to all individuals of the population?. Explaining how this idea has been implemented is the purpose of the next section.

## 2 Implementation

Let us suppose that the architecture of our individuals consists of just two parts: a main body and one ADF (ADF0). As it is shown in Figure 1, our implementation divides the population of individuals into two separated populations: one for program's main bodies (called main population) and the other one for ADFs (named ADF population). Both populations will evolve independently. Each population will supply the best individual obtained so far to the other population so that individuals of the other population can be evaluated. More specifically, the main population will supply the best main body obtained so far to the ADF population. Likewise, the ADF population will supply the best ADF0 obtained so far to the main population.
In order to evaluate a member (a main body) of the main population, it will be coupled with the best ADF0 supplied from the ADF population, a whole individual will be built and the fitness obtained by that individual will be assigned to the main body being evaluated. Similarly, in order to evaluate an ADF in the ADF population, the ADF will be coupled with the best main body supplied by the main population and the resulting individual will be evaluated. Of course, it is impossible to evaluate an individual in a population until a best individual has been obtained in the other population. But this is also true for individuals in the other population. As the process must start somewhere, at the beginning of the run a randomly chosen individual from each population is designated as the best of that population.

[1] The reader should be aware that by *improvements* we mean *global* improvements, that is, improvements that lead to a change in the best of population

Therefore, all individuals in the main population will be coupled and evaluated with the same ADF (the best one obtained so far). And vice versa. Thus, an improvement found by the main population will be immediately transferred to all ADFs in the ADF population (and vice versa). An immediate advantage of this idea over Koza's ADF is that each population can run in a separate machine, being the interaction between populations (and therefore, network communication) very low. Although figure 1 displays only two populations, many more populations could be used in problems requiring more ADFs. Our implementation doesn't take advantage of parallelism, though, so populations are evaluated sequentially: if we let a $n$ population system run for 150 generations, $P_0$ will be run at generation 0, $P_1$ will be run at generation 1 and so on. $P_0$ will be

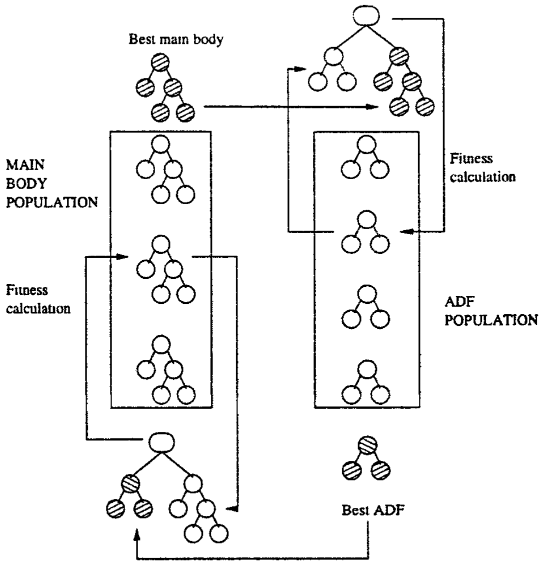

**Fig. 1.** Inter-relations between the main body population and the ADF population.

run again at generation $n$, $P_1$ at generation $n+1$ and so forth. Therefore, each $P_i$ will run for $150/n$ generations, interleaved with the rest of the populations. Table 1 shows the algorithm we have used for this article.

```
1. For each $i$, create population $P_i$ and choose randomly a best-of-run $B_i$.
2. Do until the number of generations is exhausted or success.
   (a) run GP on $P_0$ for 1 generation and update $B_0$. Stop if success.
   (b) run GP on $P_1$ for 1 generation and update $B_1$. Stop if success.
   (c) ...
   (d) run GP on $P_{n-1}$ for 1 generation and update $B_{n-1}$. Stop if success.
```

**Table 1.** Basic *Iadf* algorithm.

Next section shows some experimental results obtained by our system for the EVEN-5-PARITY and EVEN-6-PARITY problems.

## 3 Experimental Results

The system shown in figure 1 has been tested with the EVEN-5-PARITY and EVEN-6-PARITY problems, described in [4]. Table 3 shows the tableau for the EVEN-5-PARITY problem problem with ADFs (the EVEN-6-PARITY problem is similar).
This tableau is similar to Koza's but for M and G[2]. Koza's M is 16000 whereas we use a much smaller population size of 200 for EVEN-5-PARITY and 400 for EVEN-6-PARITY. Besides, as we wanted to explore the behaviour of the system for long runs, our G has been extended to 150 (being Koza's G = 51). As we use different parameters, we performed a series of experiments for Koza's ADFs as well, so that it can be compared to our system. From now on, Koza's ADF results will be referred to as *Kadf* and our system results as *Iadf* ("Independent ADFs").
As Koza states in [4], a good way to determine how well an adaptive system performs (for a given problem and chosen parameters) is to obtain the computational effort ($E$) for that problem. Computational effort and related data for both *Kadf* and *Iadf* are shown in table 3. Graphs displaying computational effort per generation are shown in figures 2 and 3. Also, figures 4 and 5 show the cumulative probabilities of solving EVEN-5-PARITY and EVEN-6-PARITY respectively.

[2] M is the size of the population and G is the number of generations

| | |
|---|---|
| Objective | Find a program that produces the value for the Boolean even 5-parity function as its output when given the values of the tree independent Boolean variables as its input. |
| Architecture | One result-producing branch and two two-argument functions-defining branches, with ADF1 hierarchically referring to ADF0. |
| Parameters | Branch typing. |
| Terminal set for the result-producing branch: | D0, D1, D2, D3, D4 |
| Function set for the result-producing branch: | ADF0, ADF1, AND, OR, NAND, and NOR |
| Terminal set for the function-defining branch ADF0: | ARG0 and ARG1 |
| Function set for the function-defining branch ADF0: | AND, OR, NAND, and NOR. |
| Terminal set for the function-defining branch ADF1: | ARG0 and ARG1 |
| Function set for the function-defining branch ADF1: | AND, OR, NAND, NOR, and ADF0 (hierarchical reference to ADF0 by ADF1). |
| Fitness cases | All $2^5 = 32$ combinations of the five Boolean arguments D0, D1, D2, D3, D4. |
| Raw fitness | The number of fitness cases for which the value returned by the program equals the correct value of the even-5-parity function. |
| Standardized fitness | The standardized fitness of a program is the sum, over the 32 fitness cases, of the Hamming distance (error) between the value returned by the program and the correct value of the Boolean even-5-parity function. |
| Hits | Same as raw fitness. |
| Wrapper | None. |
| Parameters | M = 200, G = 150 |
| Success predicate | A program scores the maximum number of hits |

**Table 2.** Tableau with ADFs for the even-5-parity problem.

| | EVEN-5-PARITY | | EVEN-6-PARITY | |
|---|---|---|---|---|
| | *Kadf* | *Iadf* | *Kadf* | *Iadf* |
| Number of experiments | 194 | 200 | 111 | 84 |
| Population size | 200 | | 400 | |
| Effort = $min_{i=0..149}(I(M, i, 0.99))$ | 408000 | 359600 | 1550000 | 627200 |
| Best generation $i*$ | 33 | 30 | 30 | 48 |
| $E_{Kadf}/E_{Iadf}$ | 1.134594 | | 2.471301 | |

**Table 3.** Computational effort results (and related data) for *Kadf* and *Iadf*.

## 4 Discussion

It turns out that *Iadf* performs slightly better than *Kadf* (see table 3) for the EVEN-5-PARITY problem (being the effort ratio $E = 1.134594$) and much better for the more complex EVEN-6-PARITY problem ($E = 2.471301$). However, that is the effort that the system would have spent had we chosen $G = i*$. But $i*$ is not a datum we can know a priori. Had we started our runs without this knowledge, we could have chosen any other G and spent a different computational effort $I$. In order to have a better picture of what happens for different values of G, graphs displaying computational effort are shown in figures 2 and 3. Also, figures 4 and 5 show the cumulative probabilities of solving EVEN-5-PARITY and EVEN-6-PARITY respectively. Results in these graphs can be easily summarized:

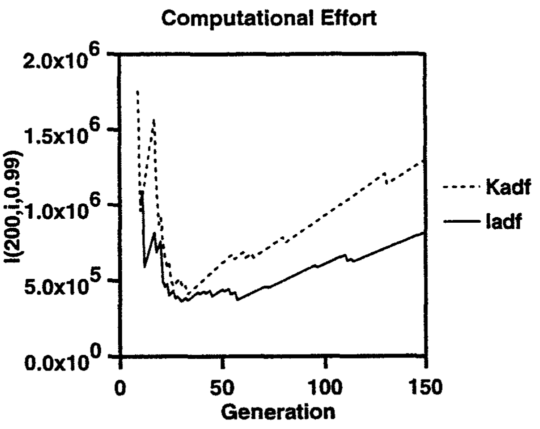

**Fig. 2.** Computational effort (I) for both *Kadf* and *Iadf*, given that EVEN-5-PARITY should be solved by generation i with probability z = 0.99

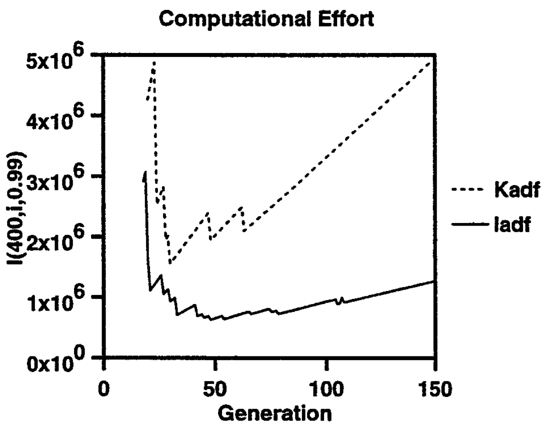

**Fig. 3.** Computational effort (I) for both *Kadf* and *Iadf*, given that EVEN-6-PARITY should be solved by generation i with probability z = 0.99

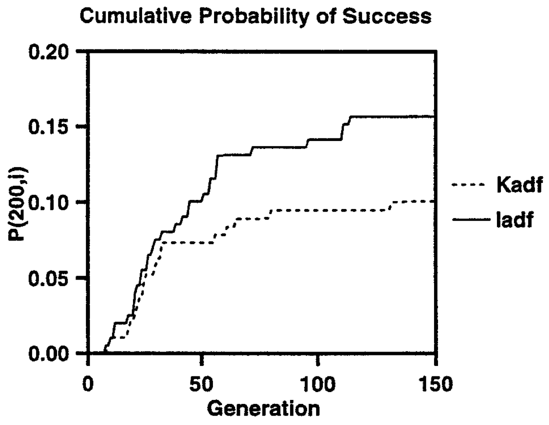

**Fig. 4.** Cumulative probability of solving EVEN-5-PARITY by generation i with M=200 for both *Kadf* and *Iadf*

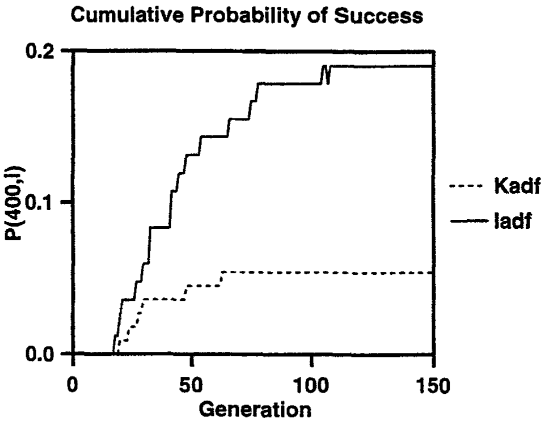

**Fig. 5.** Cumulative probability of solving EVEN-6-PARITY by generation i for both *Kadf* and *Iadf*

- *Iadf* has a smaller computational effort than *Kadf* for all generations and specially for late generations (see figure 2). This fact is even more noticeable for the more difficult problem (EVEN-6-PARITY) (see figure 3).
- *Iadf* manages to keep a steady rate of improvement (in terms of cumulative probability of success) for longer than *Kadf* (see figure 4). *Kadf*s rate diminishes by generation 30 whilst *Iadf* continues improving at a good pace for much longer. Again, this is even more noticeable in the EVEN-6-PARITY problem (see figure 5).

## 5 Future Work

In this article we have studied the effects of transferring improvements immediately to a whole population. Our expectations were that this might be useful in terms of computational effort, and it happened to be case for the EVEN-5,6-PARITY problems. However, in other problems it might happen that an improvement for an individual A could not always be accepted as such by another individual B and in the worst case, it will hinder that individual. Many other individuals could be hindered in such a way, being their evolutionary pathways lost as a result.
In that case, a softer way of transferring improvements might be useful. Instead of transferring the best individual of a whole population $P_1$ to

another whole population $P_2$, the former population could be divided into subpopulations $P_{11}, P_{12}, \ldots$, each one of them having a best individual of their own. The receiving population $P_2$ could be divided similarly. Then, the best individual of $P_{11}$ would be transferred to $P_{21}$ (and vice versa) and so on. It is a softer approach because improvements only affect a few individuals of a population and therefore, dammaging effects would be restricted to them. In an extreme case, each $P_{1i}$ would be exactly *one* individual, linked to its co-individual $P_{2i}$, which is exactly Koza's ADF approach. Thus, it seems that Koza's ADFs lies in an extreme of a gradation, and our approach lies in the opposite extreme. Testing approaches between both extremes will be our next step.

## 6 Related Work

*Iadf* originated in an idea that emerged from previous work done by the author. In [2], it was indirectly shown how fixing part of a program and letting the rest evolve, could be an interesting way for a programmer to introduce background knowledge into GP and to reduce the search space. This article is an offshoot of that idea, although it is an evolving population best individual (instead of the programmer) which fixes part of the program for the rest of the evolving populations.
The system we have studied in this paper can be considered as an extreme case of co-evolution [3], albeit a strange one, because interaction between populations happens only through the best individual of each population. Co-evolution of a main program and several independent ADF populations has already been dealt with in [1]. Both approaches differ in perspective, though: we are more interested in the simultaneous transfer of information from one population to all individuals in the other populations than in studying general ADF co-evolution. In their approach, in order to evaluate a main program, ADF individuals are selected from the ADF sub-populations. They tested several selection policies, being "the best individual" policy very similar to our own approach. However, in their work this policy doesn't fare well compared to GP+ADF, which is the opposite of the results obtained in our paper. Other differences are that we use a generational model for all evolving populations instead of a steady-state model and that we favor program-level fitness evaluation instead of evaluating directly the individuals in the ADF sub-populations. Finally, our results are in terms of computational effort to solve the problem rather than of average results per generation, as in their case.

## 7 Conclusions

This paper started by posing the question of whether it would be useful that improvements in a part of an individual (a subroutine, for instance) would be transferred to all members of the population as soon as they were found. We then proposed a system to test this idea and utilized it for the EVEN-5,6-PARITY problems. A comparison of our results with Koza's ADF applied to the same problem shows that performance (in terms of minimum computational effort) is better. Thus, there seems to be an advantage by immediately transferring improvements to all individuals, at least in this case.
Our approach is another way to parallelize GP, with the advantage that communication between populations happens at a very small rate: all the information populations need to exchange is the best individual obtained so far, which changes rather slowly. This kind of parallelism would be useful for problems requiring many different ADFs.
Our system shows a curious effect: the cumulative probability of success keeps increasing at a good rate for longer than *Kadf*. That is, it doesn't seem to stagnate as soon as GP (or GP+ADF) does.
The approach scales well, obtaining better results for the more complex problem than for the simpler one.
It has been shown that our approach and Koza's ADFs lie in opposite extremes of a gradation. Thus, our work suggests that testing approaches between both extremes might be an interesting line of research.
Finally, we are well aware that in order to draw general conclusions beyond the problems studied in this article, many more experiments must be carried out for different problems and different parameters.

## References

1. Manu Ahluwalia, Larry Bell, and Terence C. Fogarty. Co-evolving functions in genetic programming: A comparison in ADF selection strategies. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 3–8, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.
2. Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Evolving heuristics for planning. In *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, Lecture Notes in Artificial Intelligence, San Diego, CA, March 1998. Springer-Verlag.

3. W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, volume X of *Sante Fe Institute Studies in the Sciences of Complexity*, pages 313–324. Addison-Wesley, Santa Fe Institute, New Mexico, USA, February 1990 1992.
4. J.R. Koza. *Genetic Programming II.* The MIT Press, 1994.