A CO-EVOLUTIONARY APPROACH TO PARALLEL DISTRIBUTED GENETIC PROGRAMMING

Shotaro Kamio and Hitoshi Iba

Graduate School of Frontier Science, The University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan.
{kamio,iba}@miv.t.u-tokyo.ac.jp

Abstract

We propose a co-evolutionary approach to parallel distributed GP with the ADF method. This paper presents results of experiments to prove the effectiveness of this method. We also introduce a way to analyze how this method works in run time using the entropy. Furthermore, we show this will help to obtain the best performance of the proposed method.

1. Introduction

Genetic Algorithms (GAs) and Genetic Programming (GP) have been proven to be efficient tools for a wide range of applications. However, for complex problems, they require massive computational power. In order to speed up the computation, parallel distributed GA and parallel distributed GP are studied by many researchers (Tanese, 1989; Gordon and Whitley, 1993; Martin, *et al.*, 1997; Andre and Koza, 1998; Iba and Niwa, 1996; Punch, 1998). However, there are few previous works which focus on the ADF method in GP.

In this paper, we propose a co-evolutionary approach to parallel distributed GP. Thereafter, we show results of experiments to confirm the effectiveness of our approach.

2. A Co-evolutionary Method

When GP is applied to a complex task, Automatically Defined Functions (ADF) method (Koza, 1994) is often used. In ADF method, an individual program generated by GP consists of one main program and ADF subroutines. A main program is the one including subroutine calls to ADF subroutines. An ADF subroutine is also a program which is called from a main program with arguments. Our method comes from the analogy of *co-evolution* in nature. The co-evolution among main programs and ADF subroutines means that they cooperate each other to acquire a good fitness value. In other words, main programs and ADF subroutines evolved separately are expected to become efficient programs. We have adopted this idea into the parallel distributed GP.

The island model is a popular method in the parallel distributed GP. In this model, the whole population are split into many sub-populations (which are called demes). Demes communicate with each other to exchange the individuals. This communication is called *migration*.

Our method is based on the island model. In our method, demes have a little different responsibility from the traditional ones (Andre and Koza, 1998; Iba and Niwa, 1996). Each deme evolves one program among a main program and ADF subroutines. In addition, each individual on the demes has one program among a main program and ADF subroutines. A deme for main programs consists only of a population of main programs, whereas a deme for ADF subroutines consists only of a population of ADF subroutines.

Communication is used for the evaluation. We cannot evaluate ADF subroutines only by themselves. This is because both a main program and its ADF subroutines are required for the evaluation. The communication between demes is used for transferring a few main programs or ADF subroutines. We call them as "sub-programs for the evaluation". This means a few main programs in demes for ADF subroutines and a few ADF subroutines in demes for main programs. However, too much communication makes the computation slow down. In our method, this communication is executed only once every several generations to reduce the amount of communication.

The migration is also used in our method as in the island model. Elite individuals are transferred by migrating between demes. In our method, the migration occurs only between the same kind of demes, i.e., we can migrate from a deme of main programs to another deme of main programs. In other words, we cannot migrate from a deme of main program to a deme of the ADF subroutines, and vice versa.

Our method has three phases described below:

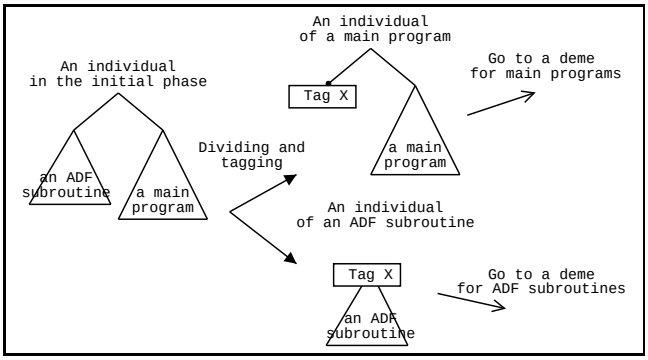


Fig. 1 A diagram of the division phase.

(1) The initial phase
Individuals on a deme are evolved with the usual ADF method. Each individual has one main program and ADF subroutines. In this phase, each deme is independent from each other. Individuals in this phase become an initial population for the next phases.

(2) The division phase
The whole GP individual in the previous phase is divided into an individual in a deme of one main program and individuals in several demes of one ADF subroutine (Fig. 1). Each deme becomes specialized to one of them. Individuals are transferred to a specified deme for the evolution. This phase does not include any calculation.

(3) The island phase
Individuals in a deme are evolved with the usual GP. The communication for the evaluation and migration between demes is executed every several generations.

2.1 Tags for ADF subroutines

We have defined a number tag for coupling a main program and ADF subroutines to evaluate. At the division phase, each individual in the initial phase is split into an individual in a deme of a main program and individuals in demes of one ADF subroutine. At that time, all ADF subroutines are tagged by a unique number. The same number tags are attached to the main program's points in which they call the ADF subroutines (Fig. 1).

During the evaluation, an individual of main program calls an ADF subroutine matching the number tag. In the same way, an individual of the ADF subroutine can find the caller program, or a main program, by the tag. The number of a tag is not constant during the evolution. When an individual fails to find a program by a tag, the tag is changed to a certain value which can be found by another. It means that the individual changes the program to call or to be called by (Fig. 2).

We have defined the actions of the tags when applying the genetic operators to the individuals as follows:

- By the mutation, the values of tags are not changed.

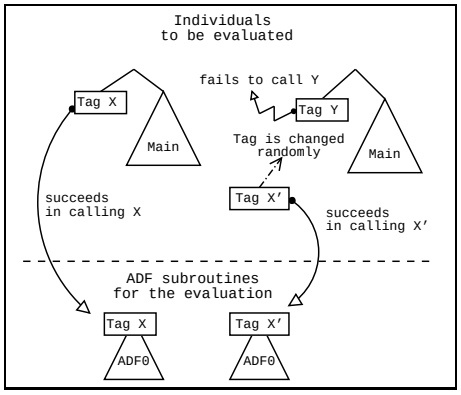


Fig. 2 The evaluation process using the tags (on a deme for main programs).

- By the crossover, the values of the tags of parents are inherited by their children, i.e., two parents whose tags are x and y generate two children whose tags are x and y.

These actions do not change the total amount of types of tags. However, some selection methods (e.g., the tournament selection) may cause to decrease the amount with generations.

The elite strategy retains the elite individuals. When using this strategy, the individuals with worse fitness values will be reduced. Therefore, using the elite strategy also decreases the amount of types of tags.

2.2 The fitness function for ADF subroutines

It is important for ADF subroutines to be called by a main program. To implement this, we design the fitness function of an ADF subroutine as follows:

$$f_{\mathrm{ADF}} = \frac{f_{\mathrm{main}}}{count_{\mathrm{ADF}}} \ , \tag{1}$$

where f_{main} means the fitness value of a main program calling the ADF subroutine. And $count_{\mathrm{ADF}}$ means the number of times the ADF subroutine is called during the evaluation. In this case, smaller fitness values are better.

This equation implies that the more frequently ADF subroutines are called, the better. As a result, ADF subroutines can evolve to be called as many times as they can and to contribute to the fitness values of a main program.

3. Experiments

We have applied our method to solve three benchmark problems; i.e., the Mackey-Glass problem, the 8-input multiplexer problem and the Ant problem (The Santa Fe trail). Used parameters for our method are described in Table 1. These problems require one main program and one ADF subroutine. We have experimented in two cases; i.e., four demes (two for main programs and two for ADF subroutines) and six demes

Table 1 Parameters used in our method.

Population size on each deme	1000
#. demes	4 or 6
Division phase	At the 10th generation
Comm. for the evaluation	Every five generations
Migration	Every five generations

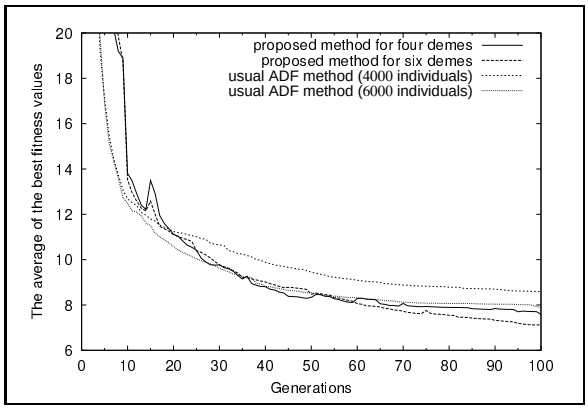


Fig. 3 The result of the Mackey-Glass problem.

(four for main programs and two for ADF subroutines). All experiments were repeated 30 times.

To compare with our method, panmictic GP with usual ADF method was also executed. Their population size was set to 4000 ($= 4 \times 1000$) and 6000 ($= 6 \times 1000$). The other parameters were the same for the fair comparison.

4. Results

The results are plotted in Figs. 3, 4 and 5. These graphs show the averages of the best fitness values with generations.

In our method for the Mackey-Glass problem and the Ant problem, a large slump appeared at the 10th generation and a large peak appeared at the 15th generation. Note that the fitness value in our method was decreasing almost monotonously, otherwise. Our method for six demes gave lower values than for four demes at the end of the run.

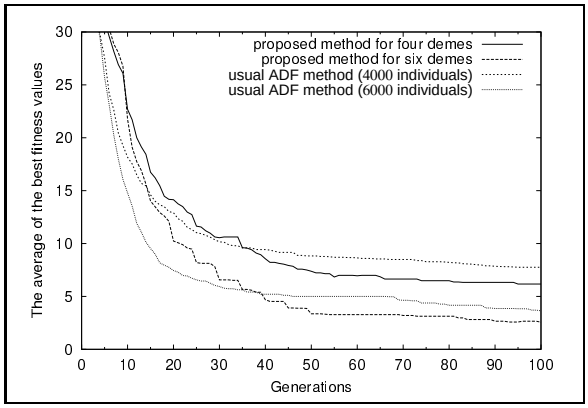


Fig. 4 The result of the 8-input multiplexer problem.

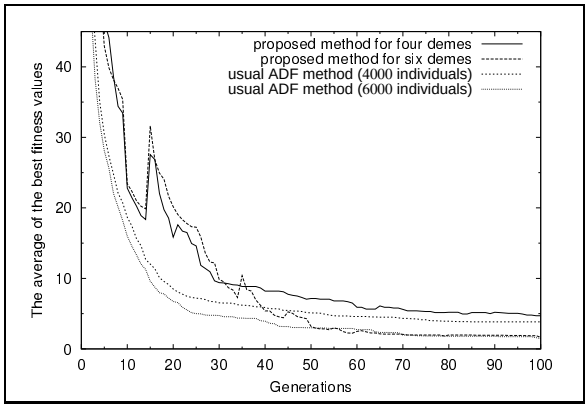


Fig. 5 The result of the Ant problem.

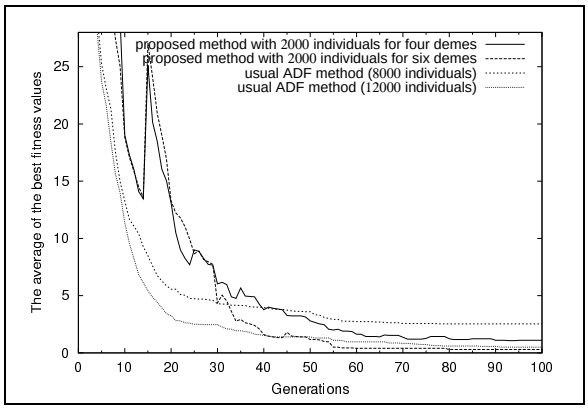


Fig. 6 The result of the Ant problem with 2000 individuals on each deme.

For the multiplexer problem and the Mackey-Glass problem, our method for four demes performed better than the usual ADF method with 4000 individuals. Six demes performed better than both 4000 individuals and 6000 individuals. For the Ant problem, our method for four demes performed worse than the usual ADF method. Six demes performed almost as well as 6000 individuals at later generations.

The Ant problem is a problem that the population size is the major factor of the performance. Hence, the larger population we use on the usual method, the better the performance of the result is. Moreover, if we use larger population than a certain size, GP can find the complete answer soon. In this experiment, the population size on each deme was small. As a result, our method performed worse than the usual ADF method with a larger population. We should use larger population for the Ant problem. When experimented with 2000 individuals on each deme, our method for four demes performed better than the usual ADF method with 8000 ($= 4 \times 2000$) individuals and six demes performed better than 12000($= 6 \times 2000$) individuals (Fig. 6).

5. Discussion

5.1 Summary of results

The results obtained using this method were better than those obtained using the usual ADF method. In addition, the results were improved further by increasing the number of demes. It can be said that these results demonstrate the effectiveness of this method.

Regarding the Ant problem and the Mackey-Glass problem attendant to this method, a large slump (improvement in results) occurred at the 10th generation, and a large peak (worsening of results) occurred at the 15th generation. These phenomena are due to communication. As a result of mutual communication between good individuals obtained from each deme in the 10th generation of the division phase, individuals that showed better results than even the best individuals in their own demes were transferred, hence the fitness values abruptly decreased. Subsequently, usual GP continued from 10th generation to 15th generation on each deme, resulting in a somewhat gentle reduction.

The 15th generation is the one in which transfer of sub-programs for the evaluation occurs. If the sub-programs for the evaluation change significantly from the previous ones due to this communication, there is a possibility that the results of the individuals that were called will change significantly. In addition, there is a possibility that individuals that could have been called up to that point in time can no longer be called. In such a case, the values of the tags change over. The new values are randomly chosen from the tags of sub-programs for the evaluation in the deme, so the results sometimes improve or sometimes become worse. The above phenomenon occurs, and is manifest as this peak in the results.

It can be clearly seen that, even after the 15th generation, small slumps and peaks appear at intervals of 5 generations in which communication is performed. This phenomenon substantiates the fact that slumps and peaks appear as a result of communication.

Also, the slump in the case of six demes at 15th generation is greater than that for the case of four demes.

The foregoing can be explained neatly as follows: If the diversity of individuals can be adequately maintained, the system can cope with changes in the sub-programs for the evaluation, enabling the most applicable individuals to be selected, which in turn improves the results.

In the multiplexer problem, neither the abovementioned large slump nor peak appeared. If there is a difference, it lies in the fact that the multiplexer problem is a problem in which the effect of the ADF is large because the answers of the multiplexer problem are liable to become a repetitive structure.

The absence of a slump in the 10th generation indicates that there is a possibility that favorable individuals that slumped abruptly remained undiscovered even among all of the demes. Also, the reason why a peak was not seen at the 15th generation is thought to be due to the fact that useful sub-programs for the evaluation were accumulated. The above can also be substantiated by looking at the execution log, which confirms that the sub-programs for the evaluation have changed over.

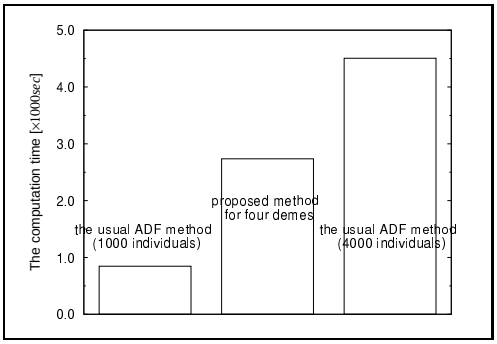


Fig. 7 The computation times.

5.2 Comparison of Computation Time

The target question in the parallel distributed method is how short the computation time becomes. In this method, the communication mechanism is added, so compared to the usual ADF method an overhead might be generated. We performed a comparison test using actual computation times in order to determine the order of magnitude of this overhead.

A comparison concerning the Mackey-Glass problem is indicated in Fig. 7. The vertical axis indicates the average computation time for one trial in 1000-second units.

As can be seen from the figure, our method requires about three times longer computation time than that for 1000 individuals using the ADF method, but does not require as much time as that required for 4000 individuals. Also, this method enables the communication traffic to be reduced by adjusting the parameters, so we believe that it permits additional reduction of computation time.

The computation time for 4000 individuals using the ADF method is more than four times of that for 1000 individuals. This is because the bloating [1] is more likely to occur with 4000 individuals.

Communication overhead of our method depends on the division and migration mainly. We can address why the communication for the evaluation does not have impact so much on the overhead. The amount of the communication for the evaluation is proportion to the number of types of the tags. Because these numbers are decreasing with generations, the amount of the communication is decreasing on each communication. There are only few types of tags by the 25th generation (Fig. 8). Therefore, the amount of the communication for the evaluation is little in the total amount of communication.

[1] This is a phenomenon in which the redundant part accumulates in a program within an individual. If bloating occurs, useless computation time is expended in proportion to the length of the redundant part (Banzhaf, *et al.*, 1998).

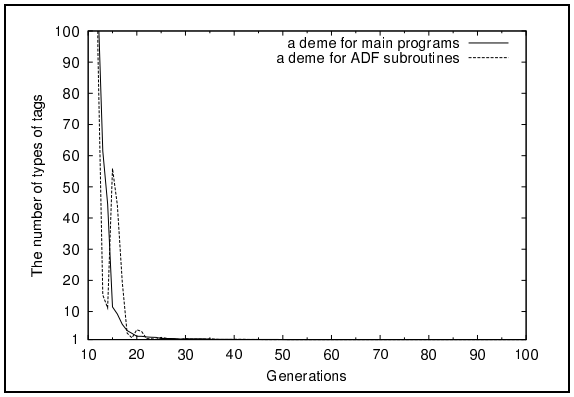


Fig. 8 A typical time series of the number of types of tags (for four demes on the Mackey-Glass problem).

In the light of these results, it can be said that our method enables the computation time to be reduced while providing better results than the usual ADF method.

5.3 Analysis based on Entropy

We carried out an analysis based on the entropy of tags. The feature of our method was the utilization of tags. This is because the main program and the ADF subroutines call each other via these tags so as to determine the results. In our method, it is considered that the diversity of the tags is more important than number of types. For this reason, we decided to use entropy, which is used in information theory (Cover and Thomas, 1991), to measure the diversity.

The definition of entropy $H(X)$ on a deme is as follows:

$$H(X) = -\sum_i X_i \log X_i \ . \tag{2}$$

X_i is the appearance ratio of a certain value of tags on the deme. This entropy is the maximum value if X_i of all types of tags is the same, and is 0 if all tags are the same type. We observed the transition of this entropy in order to figure out how our method works.

For each of the problems concerning which we performed the experiments above, we measured the entropy using this method for four demes and six demes, respectively. The results are plotted in Fig. 9. These graphs show the average values for all trials. The horizontal axis is the generation. The data were plotted from a point after an evaluation was made after the division phase (i.e., after the end of the 10th generation).

For all of the problems, peaks were observed in the generations at which the communication was performed. In generations other than peaks, we found an overall tendency for the entropy of tags to fall off. Judging from the peaks, it appears that the fluctuation due to communication is greater for the demes for the ADF subroutines than ones for main programs.

From the figures, one can understand the way in which our method works. First, when an evaluation is performed right after the division phase (at the end of the 10th generation), the number of types of tags falls to the number of sub-programs for the evaluation. Next, the number of types of tags continues to fall as evolution is progressively repeated by GP.

When the sub-programs for the evaluation are renewed by communication, the tags of the individuals which could not find the caller program or the ADF subroutine in the sub-programs for the evaluation are changed over at the time of evaluation, and the diversity of the tags increases temporarily. This is indicated by the temporary increase of the entropy in the generation at which communication takes place, as shown in the figures. Also, as evolution is progressively repeated, a cycle in which the number of types of tags decreases, and then temporarily increases due to communication takes place repeatedly.

It can be seen that by the time that the 60th generation is reached, the entropy has fallen to the small value. At that time, the tags have only several types (see also Fig. 8). Good individuals seem to be generated by causing individuals that exhibit good results in each deme to evolve from these several types of tags, transferring them as sub-programs for the evaluation, and then combining them.

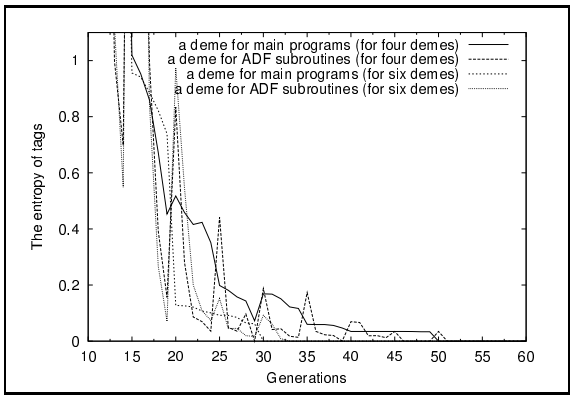
Note that the entropy value for six demes has fallen to almost zero by the 30th generation and it retains near to zero after that, while for four demes it is in a cycle of increases and decreases. As a simple application of our method to the Ant problem does not yield good result, this does not apply to that problem. Considering the fact that the results of six demes better than four demes, we can induce the hypothesis that the entropy must become smaller value to acquire better result.

We might know the performance of the final result from the entropy as we saw the above. However, from this test alone, it cannot be said unreservedly that an entropy value that is small from the outset produces good results. We think that if the number of types of tags is reduced from a very early generations, it is not possible to maintain diversity, and hence the results cannot be improved. In order to improve the results using this method, it is desirable to execute this method through generations while maintaining a certain degree of the diversity, and then reduce it once a certain generation has been reached.

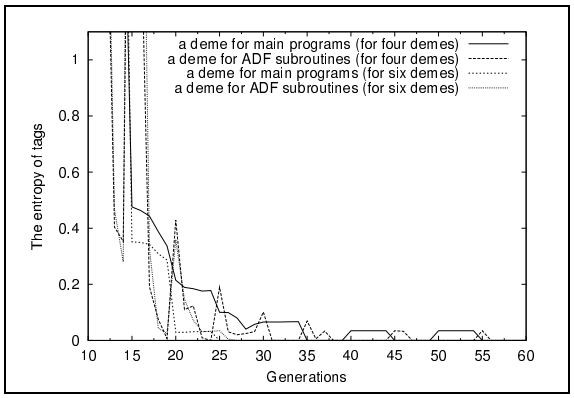
6. Conclusion and Future Works

We have proposed a new version of parallel distributed GP. The experiments showed that our proposed method performed better than the usual ADF method. We observed that the more demes we used, the better the results were. Moreover, we proposed the entropy in order to know how our method works.

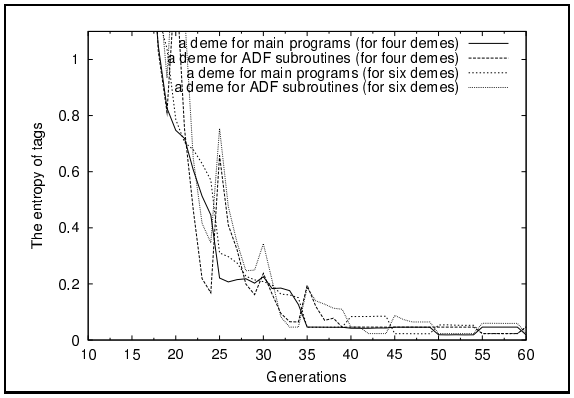
As a future research, we are trying other partitions of demes and more demes to produce more interesting results. Also, we are researching the relation between the entropy and the performance of our method. Fur-



(a) Mackey-Glass problem.



(b) Multiplexer problem.



(c) Ant problem.

Fig. 9 The time series of the entropy on three problems.

thermore, we will try to obtain the best performance from this method by using entropy as an index.

References

Andre, D. and Koza, J. (1998). "A parallel implementation of genetic programming that achieves super-linear performance", *Journal of Information Science*, Vol. 106, No. 3, pp. 201-218(18)

Banzhaf, W., Nordin, P., Keller, R., and Francone, F. (1998). *Genetic Programming - An Introduction*, Academic Press / Morgan Kaufmann, San Francisco, CA.

Cover, T. and Thomas, J. (1991). *Elements of Information Theory*, John Wiley.

Gordon, V. and Whitley, D. (1993). "Serial and parallel genetic algorithms as function optimizers", *Proc. of the Fifth International Conference on Genetic Algorithms*, (Stephanie Forrest (Ed.)), pp.177-183, Morgan Kaufmann Publishers, San Mateo, CA.

Iba, H. and Niwa, T. (1996). "Distributed Genetic Programming, Empirical Study and Analysis", *Genetic Programming 1996: Proc. of the First Annual Conference*, (Koza, J., Goldberg, D., Fogel, D., and Riolo, R. (Ed.)), pp. 339-344, MIT Press, Stanford University, CA.

Koza, J. (1994). "Genetic Programming II: Automatic Discovery of Reusable Subprograms", The MIT Press.

Martin, W., Lienig, J., and Cohoon, J. (1997). "Island (migration) models: evolutionary algorithms based on punctuated equilibria", *Handbook of Evolutionary Computation*, IOP and Oxford University Press, pp. C.6.3:1-16.

Punch, W. (1998). "How Effective are Multiple Poplulations in Genetic Programming", *Genetic Programming 1998: Proc. of the Third Annual Conference*, pp. 308-313, Morgan Kaufmann, San Francisco, CA.

Tanese, R. (1989). "Distributed Genetic Algorithms", *Proc. of the Third International Conference on Genetic Algorithms*, J. David Schaffer (Ed.), pp. 434-439, Morgan Kaufmann Publishers, San Mateo, CA.