How Effective are Multiple Populations in Genetic Programming

William F. Punch
Michigan State University GARAGe
3115 Engineering Building, East Lansing MI, 48824
punch@cps.msu.edu
517-353-3541

Abstract

The use of multiple populations in Genetic Programming is an area that is just beginning to be investigated. To date a number of conflicting reports have been generated with respect to the effectiveness of multiple populations in GP. We report here that these conflicting reports may be due a problem-dependent nature found in GP that has not been reported in GAs. This paper will review: what both multiple populations and speed-up mean in the areas of GA/GP, some conflicting results that have been reported in the GP literature on whether multiple populations gives speed-up to GP problems, and offer an answer as to why different GP problems show different kinds of speed-up when using multiple populations.

1.0 Introduction

Parallel processing is an issue that has often been examined in Genetic Algorithm (GA) research [Lin 1994, Manderick and Spiessens 1989, Mulhenbein 1989, Punch 1993, Tanese 1989, Pettey 1987]. It is worth noting that there are at least three senses of parallelism in the GA literature:

- *implicit parallelism*: This was a term first coined by Holland [Holland 1975] in some of the early descriptions of GAs. Implicit parallelism is the ability of a GA to process approximately n^3 schemata for every n individuals evaluated in the population.
- *distributed parallel processing*: This is the decrease in processing time a GA requires to process the same number of individuals as more processors are used. In the past, GAs have been described as *embarrassingly parallel*. By this, it is meant that a typical GA can easily be ported to run on multiple processors because of the simple processor communication required.
- *multiple population processing*: This is also a decrease in processing as found in the distributed case. However, this processing change is due to the fact that a single population of size *X* requires more evaluations to reach some level of performance than *n* populations, each with *X*/*n* individuals if each population occasionally exchanges solutions with other populations. This change can be realized on either a single processor or multiple processors.

The distributed processing and multiple population cases can be conflated when multiple populations are run on separate processors (see coarse-grain parallelism in Section 2.0). Such a conflation leads to the claim that a parallel GA has *superlinear speedup*. Speedup is the measure of time decrease used by an algorithm as more processors are applied. Speedup is generally indicated as a relation, such as *linear speedup*. Linear speedup means that for every processor used, a linear decrease in time is observed in the run time of the algorithm. *Superlinear speedup* means that there is a larger than linear decrease in time for processor used. The use of the term superlinear speedup is problematic, especially in the area of parallel processing. Parallel processing researchers have claimed that superlinear speedup is not possible in an algorithm where the only difference between single processor vs. multiple processor experiments is the number of processors. Summarized as Amdahl's law [Amdahl 1967], the argument is as follows. Suppose one records the time t that an algorithm takes to complete on a single processor. If, instead of being distributed across multiple processors, the algorithm is broken up into smaller segments (according to some parallelization approach) and those segments are run sequentially on the same single processor with t_{si} indicating the time needed for each segment. It is not possible that the sum time of all t_{si} be less than t since the same amount of work has to be done! The key point is the phrase "the same amount of work has to be done". In the multiple population case fewer evaluations **are** required to solve a problem than for a similar single population case (that, after all, is the point). Clearly this means that less work is needed to reach the required level of performance. Thus GAs cannot truly achieve superlinear speedup since less work is done by the sum of all subprocessors than that of a single processor. However, the increase in multipopulation performance is indeed "greater than linear".

2.0 Parallel Processing in GAs

Parallelization is an area that has been much investigated in GAs for two reasons: GAs are very easy to parallelize and GAs typically require a lot of processing time to solve any real-world problem. As a result there are a number of approaches to parallelizing GAs, depending on the particular kind of problem being solved (see [Lin 1994] for more details):

- *micro-grain parallelism*: This is the simplest form of GA parallelism. Here, we essentially parallelize only the evaluation function.
- *fine-grain parallelism*: This form of parallelism is used to reduce problems of premature convergence by using a spatial distribution of individuals combined with a crossover operator based on locale.

- *coarse-grain parallelism (multiple populations):*. In this form of parallelism, the population is divided into autonomous populations

As mentioned in Section 1.0, the coarse-grain parallel case is interesting because it simultaneously uses two approaches that speed GA processing: the reduction in work obtained by having multiple populations and the reduction in time obtained by having those populations distributed across multiple processors.

2.1 Multi-Population GAs

Since GAs are so easily parallelized, the most interesting aspect of parallel GAs is the reduction in work associated with multiple populations. While there are many factors which affect performance in multiple population GAs, the most important are the frequency of exchange, the number (and quality) of individuals exchanged, and the topology of near neighbors in the exchange. These have the most effect on the reduction in work/time.

We have conducted a number of experiments that explore these various parameters. For example, we evaluated the effectiveness of a number of different topologies and exchange approaches using a simple graph partitioning problem [Lin 1994]. The initial parallel architectures explored were all ring-based architectures with a population running on its own separate processor. Of the 8 architectures evaluated, a number of conclusions could be made. First, as the number of populations increase (that is, as the total population size was divided into smaller populations, thus keeping the total number of individuals the same for all experiments) the number of optimal solutions increased. Second, as the number of populations was increased, we observed a super linear speedup in solution time, indicating that not only was there a speedup from distributed processing, but also from a reduced workload due to multiple populations. Third, some exchange topologies were more effective than others

Finally, we began some experiments with a more radical topology termed the *injection architecture* topology. Rather than a ring topology for exchanges, the populations were arranged in a hierarchy. This hierarchy had two interesting properties. First, as you traversed the hierarchy from the root to the leaf populations, the representation used by those populations was made coarser, less fine-grained. Thus the leaf populations used a fairly coarse (abstract) representation of the problem, while it's parents used a more refined representation. Only the root node population used a full detailed representation. Second, the exchange of individuals was one-way, from coarse to fine grained populations. The effect was to have coarse-representation populations search a smaller, more abstract, space and then *inject* what appeared to be promising solutions into more fine grain representation populations for more detailed examination.

We further examined the effectiveness of the injection architecture topology on some more difficult, real-world problems. One such problem was the design of composite material beams, optimized to absorb energy for a fixed size [Punch 1995]. The beam was represented as a matrix of 24 layers of composite material, with each layer having 20 cells of material to be assigned in the layer. The GA represented the beam as a 480 element string, where each string element indicated a material to be assigned to some part of the matrix (the element size depended on the number of materials that could be used). As the evaluation function (a form of finite element analysis) of the beam was computationally expensive, we compared a micro-grained parallel approach to a simple ring topology multipopulation approach. As expected, while the micro-grained approach gave very nearly linear speedup, the ring topology gave more than linear speedup, again indicating a reduction in work due to the multiple populations. Furthermore, we used an injection architecture topology where "coarseness" meant that submatrices of the beam were represented as a single element. Again, the injection architecture outperformed the ring architecture approach. We have subsequently used injection architecture to generate designs of composite material wings [Malott 1996] and flywheels [Eby 1997, Eby 1998].

3.0 Genetic Programming

Our success with parallel GA architectures led us to examine the effectiveness of multiple populations approaches in genetic programming (GP) [Koza 1992]. Since GP approaches share many of the same features as GAs, it seemed likely that the work done on multiple populations of GAs would apply. While it is clear that distributed processing would work as in GAs, the question was whether multiple populations would have the same effect.

3.1 Genetic Programming Parallelization

The first experiments we conducted were based on two problems, one a standard machine learning problem, the "ant path" problem, and a new problem we introduced which was inspired by Holland's royal road [Jones 1994], termed the "royal tree" problem.

The ant problem starts with a 32x32 matrix, where all of the matrix elements are initially empty. Some "path" through the matrix squares are filled with "food" for the ant to eat. The problem is to start the ant at a standard point in the matrix, and given a set time period (typically something like 400 steps), see how many of the "food" particles that ant can pick up by walking over food elements.

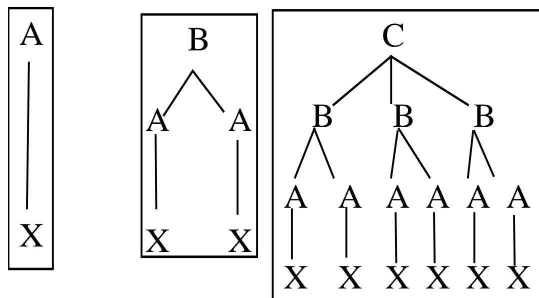

Figure 1: Example royal trees

The royal tree [Punch 1996] is a problem that we developed to be used as a "standards" function for testing the effectiveness of GPs. It consists of a single base function that is specialized into as many cases as necessary, depending on the desired complexity of the resulting problem. We define a series of functions, *a*, *b*, *c*, etc. with increasing arity. (An *a* function has arity 1, a *b* has arity 2, and so on.) We also define a number of terminals *x*, *y*, and *z*. For any depth, we define a ``perfect" tree as shown in Figure 1. A level-a tree is an *a* root node with a single *x* child. A level-b tree is a *b* root node with two level-a trees as children. A level-c tree is a *c* root node with three level-b trees as children, and so on. A level-e tree has depth 5 and 326 nodes, while a level-f tree has depth 6 and 1927 nodes.

The raw fitness of the tree (or any subtree) is the score of its root. Each function calculates its score by summing the weighted scores of its direct children. If the child is a perfect tree of the appropriate level (for instance, a complete level-c tree beneath a d node), then the score of that subtree, times a *FullBonus* weight, is added to the score of the root. If the child has the correct root but is not a perfect tree, then the weight is *PartialBonus*. If the child's root is incorrect, then the weight is *Penalty*. After scoring the root, if the function is itself the root of a perfect tree, the final sum is multiplied by CompleteBonus. Typical values used are: *FullBonus* = 2, *PartialBonus* =1, *Penalty* = 1/3, and *CompleteBonus* =2. The score base case is a level-a tree, which has a score of 4 (the a---x connection is worth 1, times the FullBonus, times the CompleteBonus).

The reasoning behind this ``stair-step" approach to the function is based on the reasoning originally used by the royal road. Many combinations of solutions can be found through genetic combination, but each *proper* combination gives a big jump in evaluation credit. The *FullBonus* is provided to give a large credit to those trees that find the correct, complete royal tree child. Since a deeper royal tree, such as a level-f tree, has a number of complete royal trees as children, each complete subtree found gives a large credit to that particular solution. The *PartialBonus* is used to give credit for finding the proper, direct child for a node, even if that direct child is not the root of a royal tree. This pressure is not as great as the *FullBonus*, but it is an effective incentive since the score is determined recursively down the tree and each node receives some credit when if finds its proper, direct children. If a node does not have the correct, direct children, it is penalized by *Penalty*, making the *FullBonus* and *PartialBonus* even more effective. Finally, if the resulting tree itself is complete, then a very large credit is given.

The reasoning behind the increase in arity required at each increased level of the royal tree is simple, we wanted to make the problem difficult for GP to solve since *increasing* arity as we climb to the next level dramatically increases the difficulty of the problem, and provides a measure of how well a GP can perform. For example, it is extremely difficult to climb to a level-f tree and we have never succeeded in climbing to level-g.

We ran three sets of experiments on both a level-e royal tree and ant problem using a single population, a ring population and an injection architecture. These experiments were conducted using the lilgp [Punch 1994] GP programming system from the MSU GARAGe. Each experiment ran a maximum of 500 generations. The optimal value for the level-e royal tree is 122,880, and for the ant it was 89. Each problem was run 16 times. The total population size is 1000 for all experiments (1 population of 1000, or the sum of all populations equal to 1000). The results of those experiments are shown in Table 1, Table 2 and Table 3. These results are reported as the number of *Wins* and *Losses*. The Wins are reported as *W:(x,y)* where *x* represents the number of optimal solutions found before 500 generations, and *y* is the average generation in which the optimal solution was found. The Losses are reported as *L:(q,r,s),* where *q* is the number of losses (no optimal solution found before 500 generations), *r* is the average best-of-run fitness, and *s* is the average generation when the best-of-run occurred.

The remaining parameters are: max-nodes(1000), max-depth(25), crossover(80% internal, 10% external, fitness-overselect), reproduction(5 %, fitness-overselect), mutation (5%, fitness, grow-method, max-depth 4). For multiple populations, exchanges were done every 10 generations, with the exchange replacing the 2 worst solutions with the 2 best solutions from its neighbor.

These results are suprising given our previous results with multiple population GAs. For the ant problem (except for the case of proportional selection with mutation) multiple populations gave *poorer* results. These results are even more dramatically different for the royal tree problem, where no optimal result was *ever* found in 64 runs of various multiple population approaches.

To confound things more, Koza & Andre [Andre and Koza 1996] reported at nearly the same time that, for the 5-parity problem they achieved super-linear speedup on a 64-node

transputer, indicating that they got both the multi-population and distributed processing speedup.

Table 1: Single Population Results

	Ant	Royal Tree
Over Selection, no mutation	**W:(7,15) L:(9,78,198)**	**W:(1,145) L:(15,6144,47)**
Prop. Selection, no mutation	**W:(2,265) L:(14,68,208)**	**W:(0,0) L:(16,71,85)**
Over Selection, with mutation	**W:(10,109) L:(6,73,300)**	**W:(8,233) L:(8,9064,159)**
Prop. Selection, with mutation	**W:(7,112) L:(9,67,158)**	**W:(0,0) L:(16,71,92)**

Table 2: Multi-population results (ring of 5 populations)

	Ant	Royal Tree
Over Selection, no mutation	**W:(4,160) L:(12,68,312)**	**W(0,0) L:(16,10005,338)**
Prop. Selection, no mutation	**W:(7,286) L:(9,71,257)**	**W:(0,0) L:(16,83,62)**
Over Selection, with mutation	**W:(6,208) L:(10,74,313)**	**W(0,0) L:(16,16284,373)**
Prop. Selection, with mutation	**W:(7,240) L:(9,73,181)**	**W:(0,0) L:(16,76,181)**

Table 3: Injection architecture results, 4 nodes feeding into 1 final result node

	Ant	Royal Tree
Over Selection, no mutation	W:(2,297) L:(14,70,326)	**W(0,0)** L:(16,20764,395)
Prop. Selection, no mutation	W:(8,270) L:(8,70,272)	**W:(0,0)** L:(16,81,152)
Over Selection, with mutation	W:(2,116) L:(14,70,304)	**W(0,0)** L:(16,18354,405)
Prop. Selection, with mutation	W:(6,309) L:(10,74,256)	**W:(0,0)** L:(16,83,192)

3.2 Resolving the Differences in Parallel GP Results

We examined a number of issues to determine the cause of the observed discrepancies. For example, Andre & Koza exchange a much higher percentage of individuals than we did, but as expected this did not change the results. We tried other topologies and a host of different configurations to no avail. Finally we ran the same kind of experiment on a larger set of GP problems. We redid similar experiments on the regression problem. The regression problem is essentially a curve fitting problem. 20 points are proposed on the curve to be fit, and the GP attempts to generate a function that hits all 20 of the points on the curve to some small tolerance. We performed the experiment with some larger population sizes (4900), and more populations (7). We did a single population and a ring of 7. Results for this experiment are shown in Table 4.

Table 4: Comparison of a single population , size 4900, vs. a ring of 7x700 populations for the regression problem

	Results	
Single Population	**W:(6,46)**	**L:(9,16,181)**
Ring of 7x700	**W:(14,36)**	**L:(2,16,180)**

Clearly the ring parallel processing did speedup the problem, as it did in the even-5 parity problem but unlike it did with the royal tree and ant problem.

There appeared to be some problem-dependent characteristics that affected whether multiple populations increased the performance of GP. We began to investigate this problem-dependent aspect and focused on two characteristics:

- the number of potential solutions. In the regression problem or the 5-parity problem there are an infinite number of solutions, while the ant problem has a limited (though large) number of solutions, and the royal road has exactly one solution. The number of solutions may affect how well the interacting populations can "bootstrap" each other. This is due to the fact that each population is in fact smaller, and will therefore have a more difficult time finding the one "correct" solution. Since each population must roughly find the same "one" solution, interacting populations may not be as effective as one large population.
- the level of deception. For similar reasons, highly deceptive problems may prevent multiple small populations from finding good solution. Deception may lead each population down "garden path" solution paths which, when exchanged with other populations will only serve to deceive them as well. A single, large population may well be able to overcome such difficulties

4.0 THE SEQUENCE PROBLEM

We tested the effect of these two factors by creating a new problem called the sequence problem. The function set consisted of 4 functions *a*, *b*, *c*, *d* , *e* and *f*. and only one terminal *x*. Functions *a* and *b* were single argument functions, while *c* was of arity 3, *d* of arity 4, *e* of arity 5 and *f* of arity 6. The goal was simply to create a sequence of the form *a-b-a-b-a-b-a-b-a-b-a-b-a-b-a-x* under various scoring conditions that would allow us to evaluate our hypotheses.

4.1 Single vs. Multiple Solutions

To test whether the number of potential solutions affected multiple population performance, we used the following two scoring functions for the sequence problem.

1. Single Solution. Only the exact target sequence would be accepted as a valid result and that target sequence was given a score of 15. Any subsequence of the right "type" but not the full length was given a score equivalent to the length of that sequence (for example *a-b-a-b-a* was given a score of 5). Any tree containing any other function besides *a* or *b* was given a score of 0.

2. Multiple Solutions. A tree is counted correct as long as the target sequence was contained somewhere in the tree. That is, the functions *c* through *f* were ignored in the scoring and only the occurrence of the target was required in the tree. In the case of multiple sequences, the longest correct sequence was used to score the tree.

Obviously there is only a single correct tree in the first scoring case, while there are an infinite set of trees in the second scoring case. The problem turns out to be relatively simple, so the populations were reduced to 1x100 in the single population and 5x20 in the multiple population case. Other parameters remained the same as found in the previous problems.

No effect is really shown, all approaches seem to solve the problem well. If any effect does appear, it is that multiple populations performed better than a single population when multiple solutions exist.

Table 5: Single vs. multiple solutions results for the sequence problem using both single and multiple populations.

	Single Population (100)	Ring Populations (5x20)
Multiple Solutions	**W: (120, 37.0)** **L: (8,10.6, 87)**	**W: (128, 25.4)** **L: (0,0,0)**
Single Solution	**W: (128, 10.3)** **L: (0,0,0)**	**W: (128, 9.2)** **L: (0,0,0)**

4.2 Deception

We then examined the deception hypothesis by modifying the scoring function in the following way. We use the "single solution" scoring as indicated above with some modifications. As before, we are looking for the single target sequence (score 15) as a solution, however we no longer penalize the *c-f* functions. If the tree contains any c-f function then the tree receives a score of 5, otherwise it receives the score of the longest sequence. This is clearly deceptive since the "true" solution should not contain any *c-f* function. This yields the following results:

Table 6: Deceptive scoring results for the sequence problem with single and multiple populations.

	Single Population (2000)	Ring Populations (5x400)
Deceptive Sequence	**W: (16, 16.3)** **L: (16, 8.3, 13.0)**	**W: (0,0)** **L: (32, 7.5, 33.5)**

The deceptive scoring approach dramatically exhibits the effect found in the ant and royal road problems. The single population approach performs much better than the equivalent multiple population approach, and no speedup is found.

5.0 DISCUSSION

In looking for possible problem-specific factors that would affect how well multiple population approaches perform, we posited two possibilities. The first is that multiple-solution problems would be more amenable to multiple populations than single-solution problems. The second is the non-deceptive problems would be more amenable to multiple populations than deceptive problems. Given the results reported above it would seem that at least when the factors are combined in a problem, that problem does better with a single population.

Overall, the "real" factor involved is how "hard" the problem is. If the populations, as related to the difficulty of the problem, are given enough resources to make progress on their own, then their exchange of solutions will increase their progress. If, however, the populations are not truly capable of making individual progress, then the exchange of information does not help as much (if at all). Thus many combinations of parameters could be used to get the above effect.

Consider a final example. We ran the same regression problem shown in Table 4 with different parameter settings, in particular we modified the population size vs. the tree depth. The results are shown in Table 7.

Table 7: Regression problem with modified tree depth for single and multiple populations.

	Max Depth 17	Max Depth 25
Single Pop	**W: (7, 47.3)** **L: (9, 16.4, 180.7)**	**W: (11, 69.0)** **L: (21, 15.7, 178.2)**
Ring Pop	**W: (14, 42.6)** **L: (2, 16.6, 179.5)**	**W: (11, 110.5)** **L: (21, 13.9, 160.0)**

The depth 17 example showed multiple population improvement, while a population size with a greater tree depth of 25 showed no real improvement. By increasing the search space (greater tree depth), each individual population can do less well than in the "simpler" depth 17 case, making interaction between populations less profitable.

The conclusion to make is that proper understanding of the "difficulty" of the problem, in combination with the available problem-solving resources, is critical for multiple population improvement.

REFERENCES

[Amdahl 1967] Amdahl, G. (1967), ``Validity of the Single processor Approach to Achieving Large Scale Computing Capabilities," *Proceedings AFIPS Conference*, Vol. 30, pp.483-485, Thompson Books, Washington, D.C.

[Andre and Koza 1996] Andre, D. and J.R. Koza (1996) "Parallel Genetic Programming: A Scalabel Implementation Using the Transputer Network Architecture", *Advances in Genetic Programming 2*, editors Peter J. Angeline and Kenneth E. Kinnear, pp 317-338, MIT Press

[Eby 1997] David Eby, R. C. Averill, Boris Gelfand, William F. Punch, Owen Mathews, Erik D. Goodman, "An Injection Island GA for Flywheel Design Optimization" Invited Paper, Proc. EUFIT '97, - 5th European Congress on Intelligent Techniques and Soft Computing, forthcoming, Sept., 97.

[Eby 1998] David Eby, R. C. Averill, William F. Punch III, Erik D. Goodman "Evaluation of Injection Island GA Performance on Flywheel Design Optimization", January 15, 1998, to appear in Proceedings, Third Conference on Adaptive Computing in Design and Manufacturing, Plymouth, England, April, 1998, Springer Verlag.

[Holland 1975] J. Holland (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

[Jones 1994] Jones, T (1994). ``A Description of Holland's Royal Road Function," *Evolutionary Computation*, 2(4), pp. 409-415

[Koza 1992] Koza, J.R.(1992), *Genetic Programming*. Bradford/MIT Press.

[Lin 1994] Lin, S.-C., W. F. Punch, and E. D. Goodman (1994) ``Coarse-grain parallel genetic algorithms: Categorization and new approach." *Sixth IEEE SPDP*, pp 28--37, October.

[Manderick and Spiessens 1989] Manderick B., and P. Spiessens (1989), ``Fine-Grained Parallel Genetic Algorithms," *Third International Conference on Genetic Algorithms*, pp. 428-433, June.

[Malott 1996] B. Malott, R.C. Averill, S.-C. Ding, W.F. Punch, and E.D. Goodman, "Use of Genetic Algorithms for optimal Design of Laminated Composite Canti lever Sandwich Panels with Bending-Twisting Coupling", presented at AIAA SDM (Structures, Dynamics and Materials), Apr 96.

[Mulhenbein 1989] Muhlenbein, H. (1989) ``ParallelGenetic Algorithms, Population Genetics and Combinatorial Optimization," *Third International Conference on Genetic Algorithms*, pp. 416-421, June.

[Pettey 1987] C. Pettey, M. Leuze, and J. Grefenstette, (1987) "A Parallel Genetic Algorithm," *Proc. Second ICGA*, July, pp. 155-161.

[Punch 1996] Punch, W.F., Doug Zongker and E. Goodman (1996), "The Royal Tree, a Benchmark for Single and Multiple Population Genetic Programming", *Advances in Genetic Programming 2*, editors Peter J. Angeline and Kenneth E. Kinnear, pp299-316, MIT Press

[Punch 1995] Punch, W. F., R. C. Averill, E. D. Goodman, S.-C. Lin, and Y. Ding (1995)``Design Using Genetic Algorithms---some results for Composite Material Structures." *IEEE Expert*, 10(1), pp 42-49, February.

[Punch 1994] W. Punch and D. Zonger, "The lilgp Programming Environment", http://garage.cps.msu.edu

[Punch 1993] W. Punch, E. Goodman, M. Pei, L. Chai-Shun, P. Hovland and R. Enbody (1993), "Further Research on Feature Selection and Classification Using Genetic Algorithms," *Proc. Fifth ICGA*, June, pp. 557-564.

[Tanese 1989] R. Tanese (1989), "Distributed Genetic Algorithms," *Proc. Third ICGA*, June, pp. 434-440.