Pareto, Population Partitioning, Price and Genetic Programming

W. B. Langdon, Email: W.Langdon@cs.ucl.ac.uk

Dept. of Computer Science,
University College London,
Gower Street, London WC1E 6BT, UK

Switchboard tel: +44 (0) 171 387 7050 xtn. 3701
Dept. Office Tel: +44 (0) 171 380 7214
Fax: +44 (0) 171 387 1397

7 April 1995

Abstract

A description of a use of Pareto optimality in genetic programming is given and an analogy with Genetic Algorithm fitness niches is drawn. Techniques to either spread the population across many pareto optimal fitness values or to reduce the spread are described. It is speculated that a wide spread may not aid Genetic Programming. It is suggested that this might give useful insight into many GPs whose fitness is composed of several sub-objectives.

The successful use of demic populations in GP leads to speculation that smaller evolutionary steps might aid GP in the long run.

An example is given where Price's covariance theorem helped when designing a GP fitness function.

1 Pareto Optimality

A number of issues have arisen from work on evolving data structures; pareto optimality, population partitioning using demes (Section 2) and Price's covariance theorem (Section 3), which could be applied more widely in genetic programming. Section 4 suggests some ways in which GP run times might be reduced.

Having evolved stacks and queues data structures using genetic programming [Lan95], I am currently working on the evolution of list data structures. Aho etal [AHU87] define a list which supports ten different operations, whereas the stack and queue both support five.

Each operation is a separate program tree within a composite individual and has its own fitness sub-score. Each sub-score is treated as a separate objective; pareto [Gol89] tournaments are used to select which individuals to breed from or remove from the population.

The advantage of pareto comparison is it allows programs to be compared using multiple objectives, without forcing an arbitrary weighting of the objectives relative to each other.

Whilst working on the queue using a scalar fitness function, it was noted that as the population evolved it appeared to trade improvement in a score on one operation with reduction in another. Changes made by crossover were likely to be lost, even if they increased the fitness of one operation, unless the increase was big enough to offset any corresponding fall in fitness caused by a reduction in one of the other components of fitness. Improvements in one operation need not cause a reduction elsewhere but it did seem to be an important effect.

Pareto tournament selection with a steady state GA (i.e. non-generational) is not elitist as there is no longer a simple definition of what the best member of the population is. Programs which score highly on one part of the fitness function may be displaced from the population by others which scores less well on that objective but do better on others.

Figures 1 to 8 (Appendix A) show initial results for the evolution of the best score (as a percentage of the perfect score) on each of the ten list operations. In general, each point on each of these lines could be a different individual. In contrast, the dotted line shows the best-of-generation sum of the ten operations and so each point does refer to an individual (such a sum, is a simple scalar value for each individual within the population).

These graphs make obvious the continual discovery and loss of individuals which score highly on a given objective. There are many cases where means of producing a high score on a particular objective is discovered multiple times, being lost from the population between each. How this effects genetic programming as a search algorithm requires further investigation. As mentioned above, this non-monotonic behaviour is not unique to pareto selection but has been observed when using a scalar fitness function, when considering the individual components of this scalar.

1.1 Size of the Pareto Front

Figure 9 shows the number of points on the pareto front as a population evolves. Each non-dominated fitness value (i.e. value on the pareto front) is equally fit and so could be considered to be a "niche".

Oei etal [OGC91] considered linear GA's with complete generational replacement and state the "naive" use of tournament selection to spread a GA population across equally fit niches will cause the population to behave chaotically. They predict the number of niches within a population of equally fit individuals will fall, being given by the formula:

$$k = \frac{1}{\frac{1}{k_0} + \frac{2t}{\pi^2 N}} \tag{1}$$

Where k_0 is the initial number of niches, t number of generations and N is the population size.

Figure 9 shows a general downward trend in the number of different non-dominated fitness values in the population (i.e. niches) after the initial high rate of fitness improvement slows. This loss of variation with time in finite populations of equally fit individuals is also known as "genetic drift".

Where a tournament group contains two, or more, non-dominated individuals, the remainder of the population can be used to rank them. Thus an individual which is dominated by few others will be ranked higher than one dominated by many. N.b. this exerts a divergent selection pressure on the population as individuals are preferred if there are few others that dominate them. Following [HNG93] the pareto rank is estimated by comparison with a sample of the population rather than all of it. When a sample of up to 81 individuals is used the number of occupied points on the pareto front rises rather than falls and reaches a much higher level, ≈ 1000, which persists at least up to generation 100.

1.2 Elitism

As has been shown pareto optimality does not ensure the population will retain the best (on any given score) individual, however this can be imposed upon the population. Figures 7 and 8, show the behaviour after adding a rule that an individual may not be deleted if it has the single highest score for any of the ten operations. Whilst these traces look better behaved, no conclusion about the behaviour of the bulk population or even the best-sum-of-generation can be draw before performing more comparative runs.

It is rather galling to realise about 90% of tournaments are finally resolved by chance. This means on average only 32% of fitness evaluations lead to a deterministic choice of which individuals to select for breeding or removal from the population. This figure is not quite as bad as it seems because the figure of 90% includes the cases where one or two of the members of each tournament group are discarded using fitness values leaving a random choice from the two or three remaining.

The graphs are from single runs and so must be treated with caution however where graphs of mean behaviour over may runs are plotted, this non-monotonic behaviour can be obscured by the averaging process.

2 Demes

Disjoint populations have been used many times within linear genetic algorithms but very little work has been reported on using them within GP. However both myself [Lan95] and John Koza [KA95] report improved performance from geographically structuring the population into semi-isolated regions or demes.

Inman Harvey suggests [Har92] that better results may be achieved with variable length (but still linear) GAs if crossover takes place primarily between individuals that are nearly the same length.

Perhaps part of the reason for the success of demic populations in genetic programming is that the localization of the population causes the GA to quickly converge so that breeding takes place between like individuals. However the population as a whole remains diverse. It is expected that not only will the programs be of similar lengths but also genotypes will be similar in other ways.

In my own work, I allowed self-crossover. In the bulk population, the chances of a program being selected twice as the parent of a given individual are remote but with small demes the chances are much higher. This would be expected to encourage the rapid local convergence of a demic GP.

If it is found that the beneficial effect of demes is due to local convergence encouraging the breeding of like with like, two measures could be taken to use this:

1. Selecting parents which are a like,

2. Selecting crossover points to minimise the difference between the offspring and its parents.

Both involve the replacement of a random process with a more directed (but still stochastic) one. The definition of similarity could be on the basis of either genotype or phenotype (fitness). The work on DNA sequencing could be a useful starting point for approximate matching of trees, when program trees become very large. Note this is exactly the opposite of GA niches or fitness sharing, where the GA population is encouraged to become spread out (either in fitness or genotype space).

3 Price's Covariance Theorem

Price's Selection and Covariance theorem [Pri70] was devised for application to biological evolution, however as Lee Altenberg [Alt94] has pointed out that it can be applied to Genetic Programming (indeed to any genetic algorithm or population based stochastic search technique where genes are passed at random to offspring in the next generation and the expected number of children allocated to each individual is proportional to its fitness).

$$\Delta Q = Cov(z, q)/\overline{z} \tag{2}$$

Where:

ΔQ is the expected change in the frequency of a gene (or fixed linear combination of genes) from one generation to the next,

z_i number of children produced by individual i, typically proportional to the fitness of i,

q_i the number of copies of the gene (or combination of genes) in individual i,

$\overline{z}$ in the mean value of z, typically 1.9.

Thus the expected (in the statistical sense) change in the number of a particular terminal or function (or combination) in the next generation can be calculated from the covariance of the frequency it occurs in the current population with the fitness of the individuals.

I have found this a useful check on the fitness function. E.g. a program that is required to push a value on the stack must use both that value and a write operation, i.e. it must contain both the "arg1" and "write" primitives. Calculating the covariance of their frequency in the initial population with fitness, explained (with one fitness function) why the number of arg1 and write primitives fell rapidly. In later generations the covariance changed sign and their numbers increased. However in some runs they became extinct in 5–10 generations, so preventing a working program from being evolved. This highlighted the importance of the fitness function in the initial population as well as later, where it was working as expected.

Another approach is to extend the syntax of the GP language and require the push program tree to include those primitives which are known to be required.

4 Reducing Fitness Evaluation Times

As Handley [Han94] has demonstrated where the primitives are free from side effects considerable performance improvements can result from caching results produced by subtrees. The read and write primitives *do* have side effects, never the less caching can be used if parts of the program are known to be free of side effects. For example a modest runtime saving of 32% was achieved when evolving the queue by using a cache of values calculated by the ADF. This was despite the fact that the ADF was called from code known to contain side effects so its argument could not be predicted in advance. A single cache of 1000 words was shared by all fitness evaluations during the GP run and hit ratios of $\approx$ 80% where obtained.

It might prove practicable to reduce the time required to evaluate an individual program's fitness by using stored knowledge of the fitness of its ancestors and how it was produced from them. For example if the crossover (or mutation) point is known to have been located in a part of the parent which is not executed then it cannot have affected the programs execution and so its fitness is also unchanged. Thus the fitness of the child can be copied from the parent with out ever running it.

Reductions in fitness evaluation time might be possible by exploiting intermediate results calculated by program's ancestors. For example where fitness is defined by running a number of independant fitness cases, the scores on each test might be saved. When an offspring is generated and the crossover point is known to be in code that is only used in some of the fitness cases only those need be run. The fitness function might be designed to make this more likely.

There has been much made of the quantity of redundant code in GP individuals. Where code is run many times during fitness evaluation it may be advantageous to use optimization techniques, such as used by some high level language compilers, to produce an executable version of the code. This need not be in machine code but could be in the GP language, albeit shorter. This is slightly different from edit [Koz92] as the compilation phase only speeds the execution of the program (i.e. the phenotype). The genetic material of the program is unchanged.

It has been argued that GP obeys Holland's schema theorem with fit (parts of) programs spreading through the population. Fitter programs being produced by crossover assembling them from fit subprograms. The fitness of a subprogram being given by:

$$\frac{\sum \text{fitness programs containing subprog}}{\text{No. copies of subprog}} \tag{3}$$

Perhaps Graphs, similar to those used by Handley, could be used to keep track of how often program subtrees are used and the fitness of the individuals in which they occur with a view of analyzing the dynamics of subprogram fitness. So far it has not proved fruitful to use Fisher's theorem [Fis58] to relate this to the variance of fitness.

5 Conclusions

There are techniques to control the spread of a genetic programming population between different, but equivalent, fitness values.

Further work is required to establish whether breeding between widely separated GP programs helps GP or whether breeding between similar programs would produce smaller improvements at each stage but more progress in the long run.

A Pareto Population Graphs

Solid lines in Figures 1 to 8 show the best-of-generation score (as a percentage of the perfect score) on each of the ten list operations. Dotted line shows the best-of-generation sum of the ten operations.

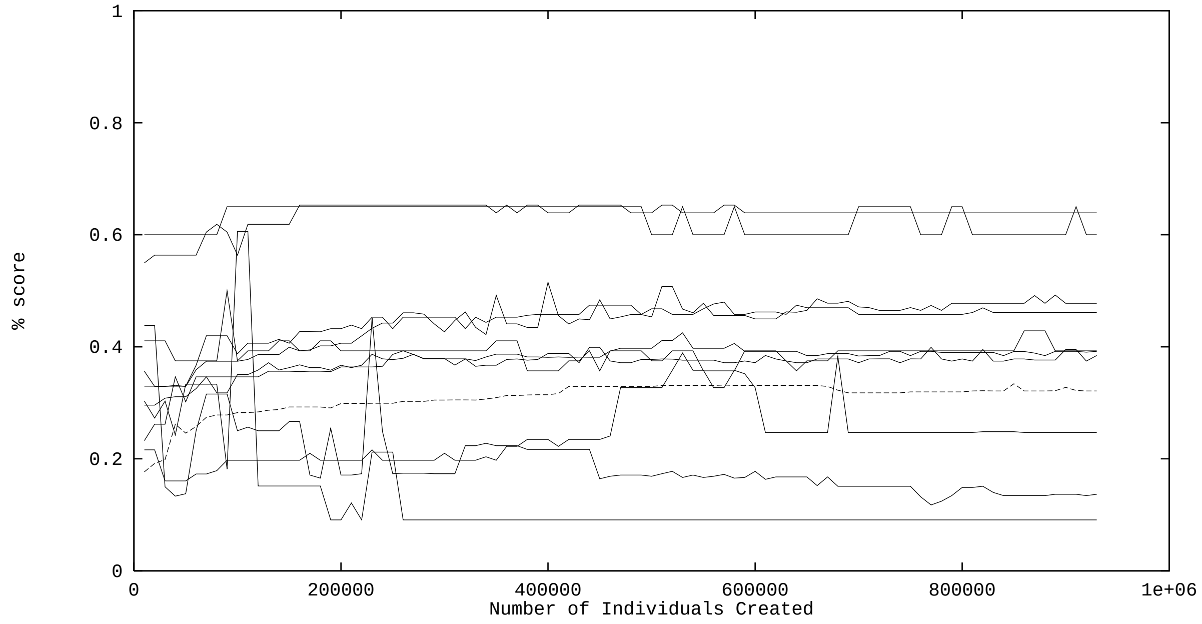

Figure 1: Variation in maximum raw score, no comparison set, no niche sharing, not elitist, pop=10,000, no demes

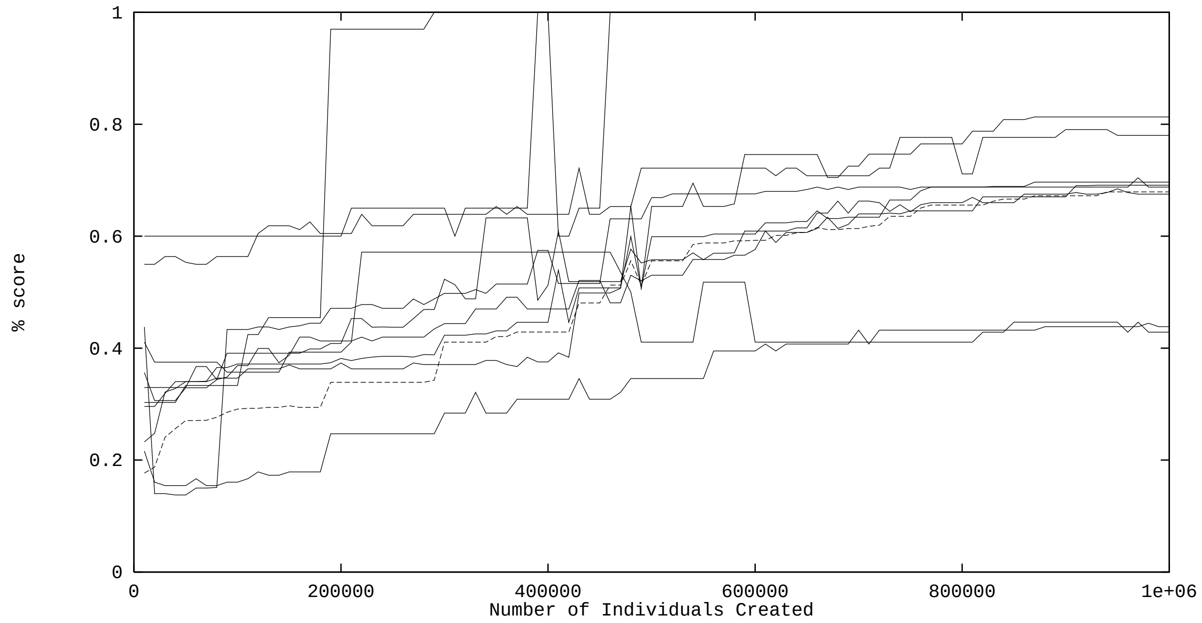

Figure 2: Variation in maximum raw score, no comparison set, no niche sharing, not elitist, pop=10,000, 3×3 demes

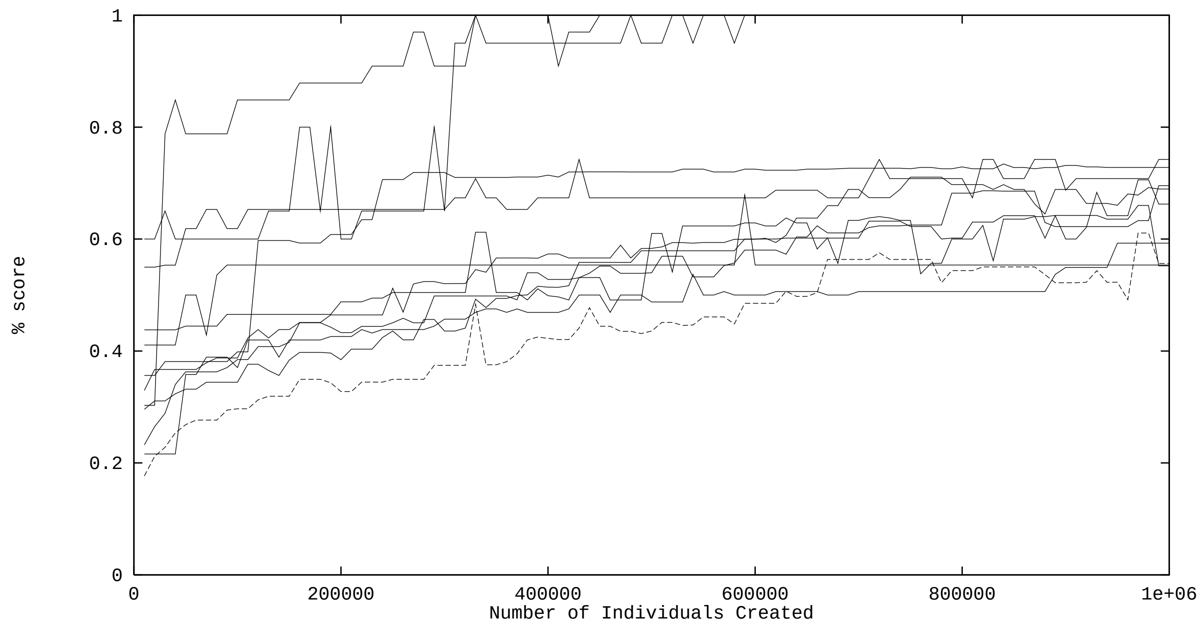

Figure 3: Variation in maximum raw score, comparison set, no niche sharing, not elitist, pop=10,000, no demes

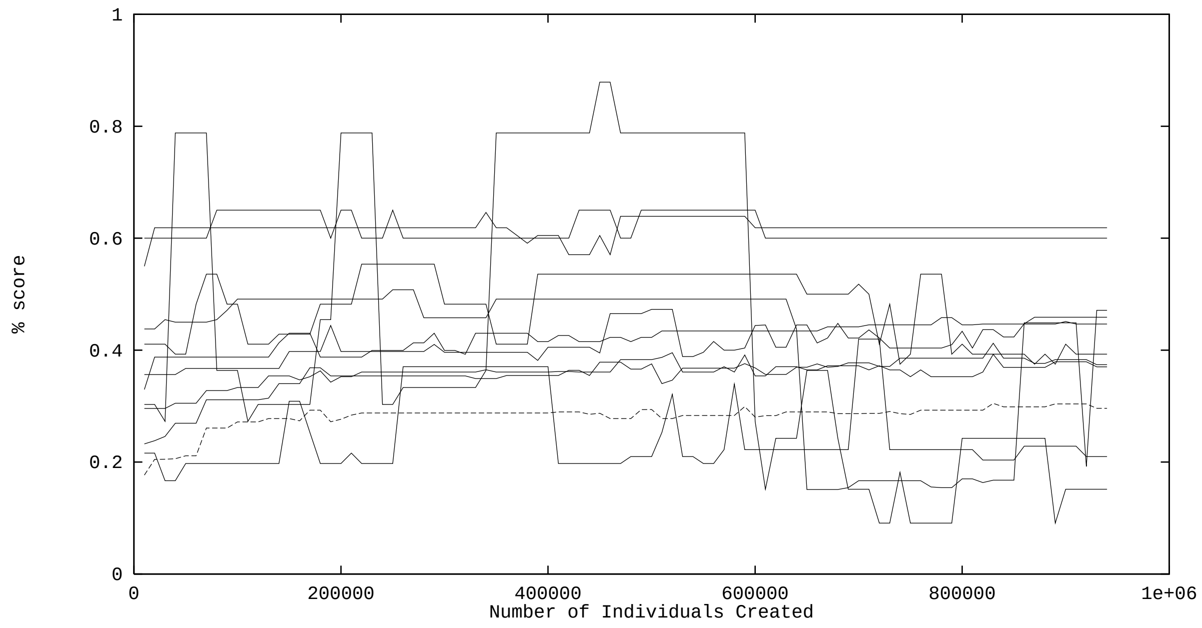

Figure 4: Variation in maximum raw score, comparison set, no niche sharing, not elitist, pop=10,000, 3×3 demes

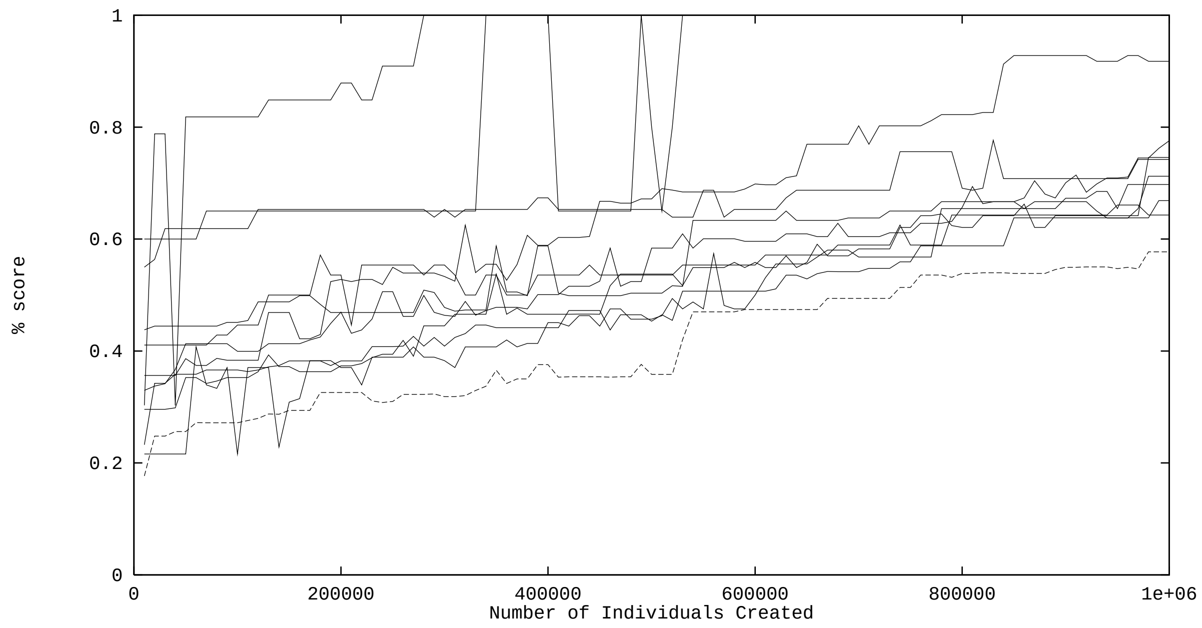

Figure 5: Variation in maximum raw score, comparison set, niche sharing, not elitist, pop=10,000, no demes

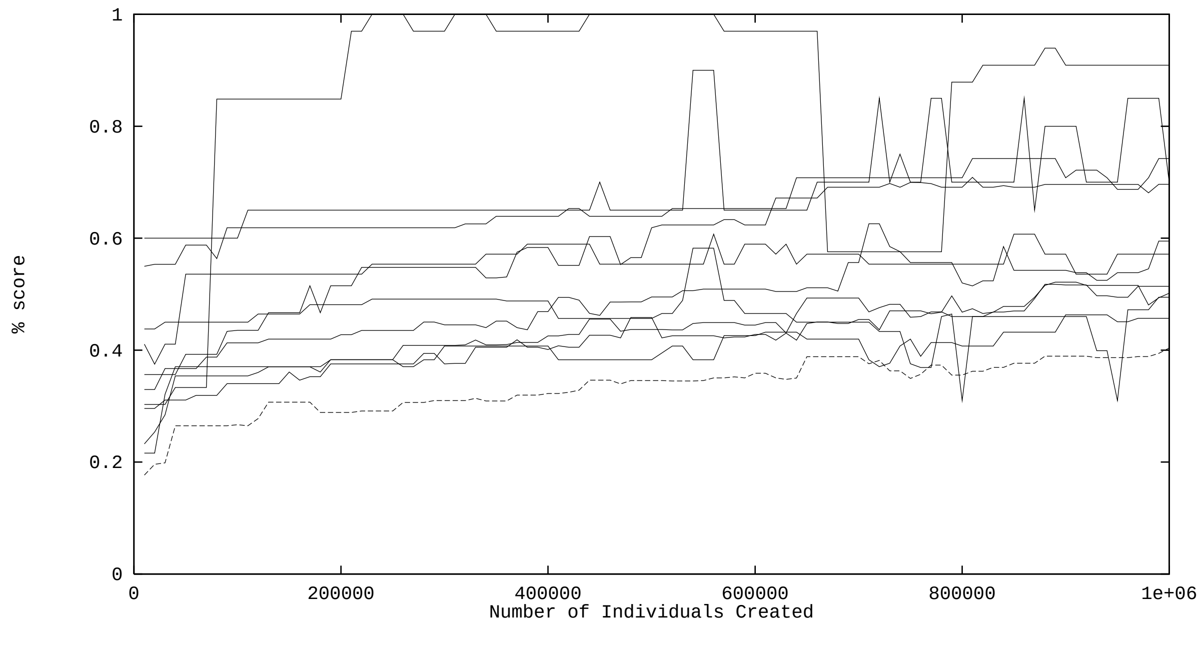

Figure 6: Variation in maximum raw score, comparison set, niche sharing, not elitist, pop=10,000, 3×3 demes

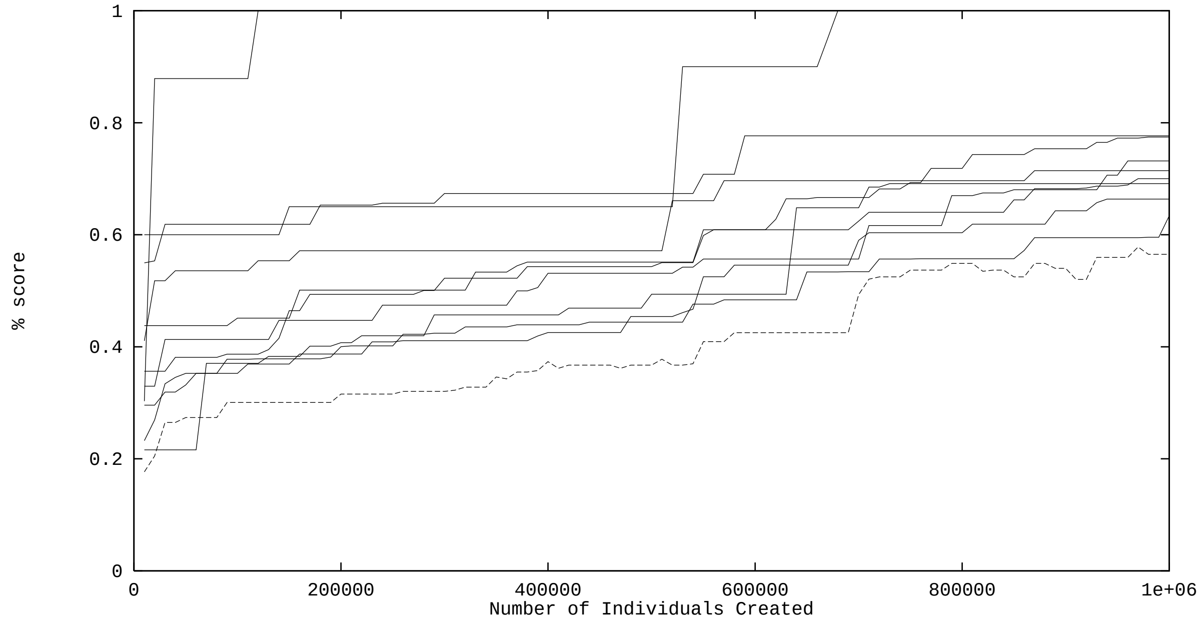

Figure 7: Variation in maximum raw score, comparison set, no niche sharing, elitist, pop=10,000, no demes

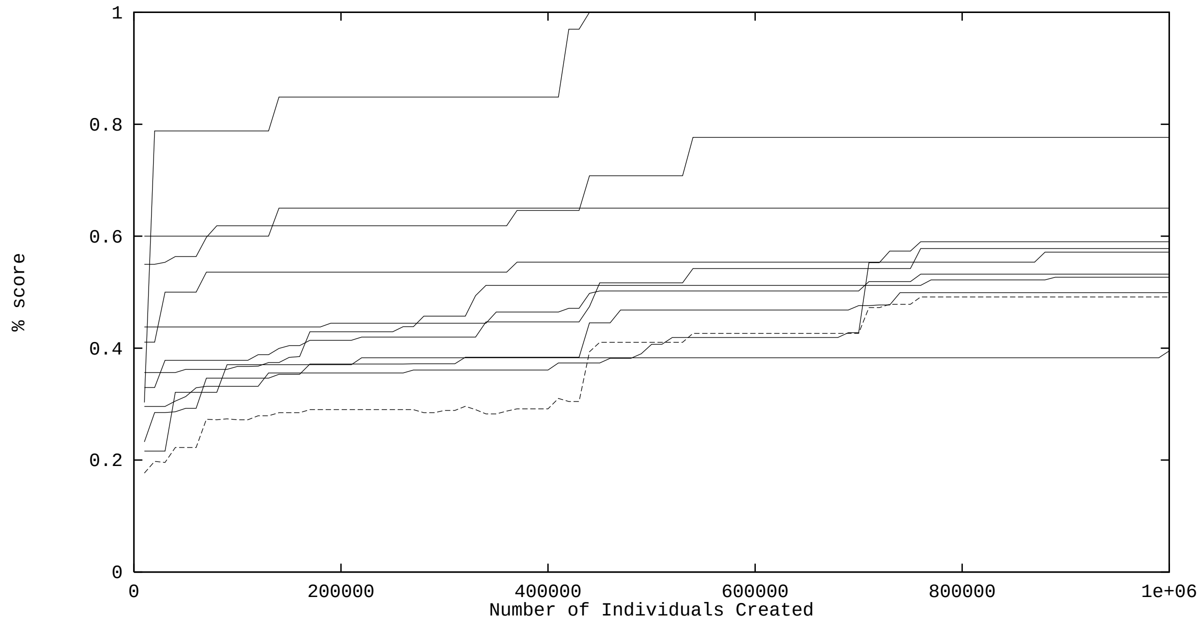

Figure 8: Variation in maximum raw score, comparison set, no niche sharing, elitist, pop=10,000, 3×3 demes

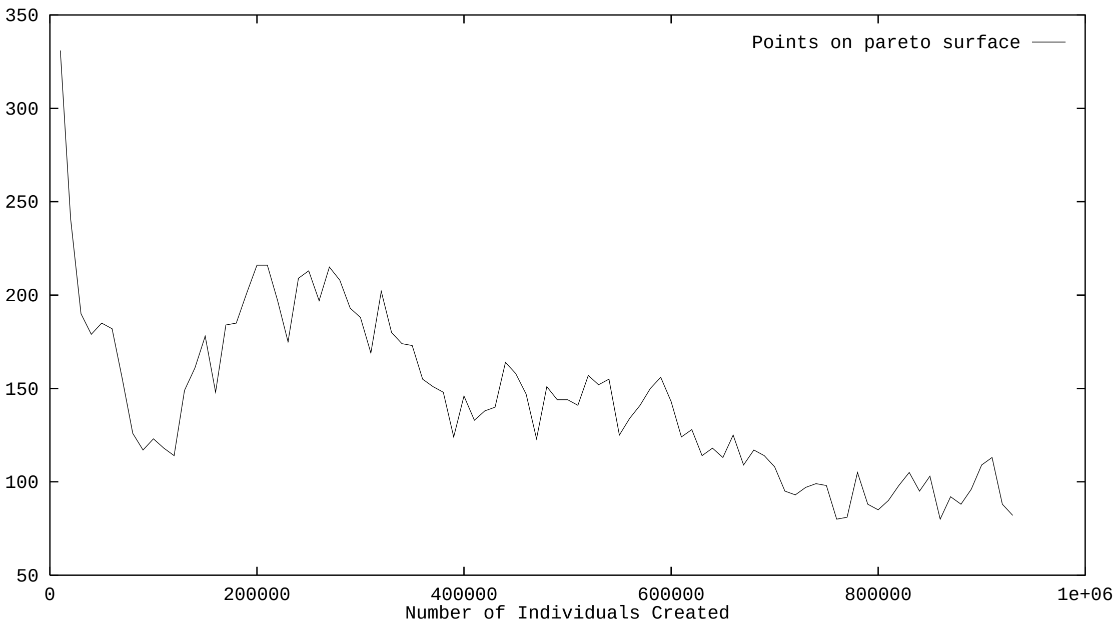

Figure 9: Number of different non dominated fitness values, no comparison set, no niche sharing, not elitist, pop=10,000, no demes

References

[AHU87] A V Aho, J E Hopcroft, and J D Ullman. *Data Structures and Algorithms.* Addison-Wesley, 1987.

[Alt94] Lee Altenberg. The evolution of evolvability in genetic programming. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming.* MIT Press, 1994.

[Fis58] Ronald A. Fisher. *The Genetical Theory of Natural Selection.* Dover, 1958. Revision of first edition published 1930, OUP.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison Wesley, 1989.

[Han94] S. Handley. On the use of a directed acyclic graph to represent a population of computer programs. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence.* IEEE Press, 1994.

[Har92] Inman Harvey. Species adaptation genetic algorithms: A basis for a continuing saga. In F.J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems, Proceeding of the first European Conference on Artificial Life (ECAL)*, pages 346–354. MIT Press, 1992.

[HNG93] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. Multiobjective optimization using the niched pareto genetic algorithm. IlliGAL Report no. 93005, Illinois Genetic Algorithm Laboratory, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 South Mathews Avenue, Urbana, IL 61801-2296, July 1993.

[KA95] John R. Koza and David Andre. Parallel genetic programming on a network of transputers. Technical Report CS-TR-95-1542, Stanford University, Department of Computer Science, January 1995.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT press, 1992.

[Lan95] W. B. Langdon. Evolving data structures using genetic programming. Research Note RN/1/95, UCL, Gower Street, London, WC1E 6BT, January 1995. Accepted for presentation at ICGA-95.

[OGC91] Christopher K. Oei, David E. Goldberg, and Shau-Jin Chang. Tournament selection, niching, and the preservation of diversity. IlliGAL Report No. 91011, University of Illinois at Urbana-Champaign, Urbana, Il 61801, December 1991.

[Pri70] George R. Price. Selection and covariance. *Nature*, 227, August 1:520–521, 1970.