Crowding and Preselection Revisited

Samir W. Mahfoud
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, IL 61801

IlliGAL Report No. 92004
April 1992

Published in *Parallel Problem Solving From Nature, 2*,
R. Manner and B. Manderick (Eds),
Elsevier Science Publishers (North Holland): Amsterdam, pp. 27-36, 1992

Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois at Urbana-Champaign
117 Transportation Building
104 South Mathews Avenue
Urbana, IL 61801

Crowding and Preselection Revisited

Samir W. Mahfoud
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, IL 61801
mahfoud@gal4.ge.uiuc.edu

Abstract

This paper considers the related algorithms, crowding and preselection, as potential multimodal function optimizers. It examines the ability of the two algorithms to preserve diversity, especially multimodal diversity. Crowding is analyzed in terms of the number of replacement errors it makes. Different strategies for reducing or eliminating error are proposed and examined. Finally, a variation of preselection is presented which approximates crowding, virtually eliminates replacement error, and restores selection pressure.

1 Introduction

Ever since the introduction of genetic algorithms, the maintenance of diversity has been an often-recurring issue. One early work that considered the maintenance of population diversity was Cavicchio's (1970) dissertation. Cavicchio introduced several of what he called *preselection* schemes, and claimed that one of them was highly successful at preserving population variance. Another early work was De Jong's (1975) dissertation. De Jong presented a more general technique which he referred to as the "crowding factor model" or more simply, *crowding*.

To date, preselection has largely remained untouched except in passing mention in a few studies. While crowding has seen sporadic application, it is viewed as being of limited use in multimodal function optimization, where the selective preservation of diversity is crucial (Deb, 1989).

While it appears from De Jong's and Cavicchio's descriptions of crowding and preselection that these algorithms should be successful at preserving population diversity, this is not the case in practice. Stochastic errors in the replacement of population members work to create a significant amount of genetic drift. Genetic drift causes a genetic algorithm to converge to one region of the search space, even when many different regions are equally fit (Goldberg & Segrest, 1987). This study is a preliminary effort at analyzing and combating drift in crowding and preselection by introducing measures which reduce or eliminate the replacement error. A series of adjustments is made to the basic crowding algorithm, resulting in a modified, effective form of preselection or approximate crowding.

2 Crowding

Crowding (De Jong, 1975) follows the simple genetic algorithm except that only a fraction of the population reproduces and dies each generation. A percentage of the population, specified

by the *generation gap* (G), is chosen via fitness-proportionate selection to undergo crossover and mutation. $G \times n$ individuals from the population must now be chosen to die (to be replaced by the ones selected for reproduction). Each element that gets inserted in the population finds the element it replaces as follows. A random sample of CF individuals is taken from the population, where CF is called the *crowding factor*. Of the CF elements, the one most similar to the element being inserted gets replaced. Similarity is defined using bitwise (genotypic) matching. This study uses roulette-wheel selection to accomplish fitness-proportionate selection, and uses sampling without replacement for applying genetic operators to selected individuals, for choosing the CF individuals, and for picking individuals to die.

Crowding is inspired by a corresponding ecological phenomenon. That is, similar individuals in a natural population, often of the same species, compete against each other for limited resources. Dissimilar individuals tend to occupy different niches, so they typically do not compete. The end result is that in a fixed-size population at equilibrium, new members of a particular species replace older members of that species. Ideally, the overall number of members of a particular species does not change.

Crowding does not model the method by which a population arrives at a stable mixture of species, but instead strives to maintain the diversity of the pre-existing mixture. To achieve this end, one could simply retain the initial population. Although this would preserve existing diversity, it would not promote useful diversity. Here, we define useful diversity as the presence of representatives from all species, including the optimal individuals of each. Species differ according to the local optimum of the search space in which they lie. Each such optimum is considered a niche. Our goal, then, is multimodal function optimization.

Originally, the maintenance of bitwise diversity to prevent premature convergence was the goal of crowding. The algorithm was successful to some extent at maintaining such diversity, but stochastic errors introduced by low CF and by other factors forced the algorithm to gradually drift towards fixed bit positions. Additionally, the maintenance of bitwise diversity is a questionable goal in itself. For example, low mutation rates will prevent the fixing of bit positions but will not prevent premature convergence. High mutation rates will prevent both the fixing of bit positions and premature convergence, but will not allow any convergence. The effects of mutation are not considered further in this study. Factors other than mutation can yield a similar effect, as demonstrated later.

3 Preselection

Five years prior to De Jong's work, Cavicchio (1970) mentioned the use of what he called "preselection schemes". Like in crowding, the goal of these schemes is to preserve diversity. However, Cavicchio stated that computing a difference measure for each new population element would be too expensive. Instead, one could estimate that a parent would be one of the members of the population closest to the new element. The validity of this assumption is demonstrated later.

The best preselection scheme works as follows. If a child has higher fitness than the worse parent, it replaces that parent. Unfortunately, the effect of this scheme is not evident in Cavicchio's work, since many other factors, including self-modifying parameters and normal selection, are also present. Here, we isolate the preselection scheme and test a variation that is expected to approximate crowding.

4 The Test Functions

The two test functions used in this paper are F1 and F2 of Deb's (1989) study of multimodal function optimization. They are defined as follows:

$$F1(x) = \sin^6(5\pi x)$$

$$F2(x) = e^{-2(\ln 2)(\frac{x-.1}{.8})^2}\sin^6(5\pi x)$$

F1 and F2 are periodic functions, each having five equally-spaced maxima in the interval $[0, 1]$. In F1, all peaks are of equal height. In F2, they are all of different height. Figures 1 and 2 display these functions graphically.

The parameter, x, is coded as a 30-bit binary string, with 30 zeros representing 0.0, and 30 ones representing 1.0. Binary integer strings are converted to real numbers through division by $2^{30} - 1$.

5 Performance Criteria

In evaluating the performance of his reproductive plans, De Jong operated under the assumption of unimodal function optimization. The ability to quickly adapt the population elements toward optimal regions of the search space was the prime criterion. This resulted in the use of two performance measures: on-line and off-line performance. On-line performance is defined as the average fitness of all past and current population elements. Off-line performance is a running fitness average of the best elements of all populations up to the current one. De Jong also used a secondary criterion for his crowding scheme, the number of bits that had become fixed.

These criteria are not of much use in this study of multimodal function optimization. Both on-line and off-line performance measures are overly concerned with speed of convergence, rather than quality of final results. Off-line performance is actually a second type of on-line performance measure, since current results are heavily degraded by past ones. Here, we are more concerned with maintaining optima and minimizing drift. As mentioned earlier, we do not strive to maintain all diversity, but to preserve multimodal diversity. Speed is a secondary consideration that, as later becomes apparent, need not be sacrificed. The number of fixed bits, though not used as a criterion to develop algorithms, is nonetheless measured, and is discussed in section 7.

The two related performance criteria used in this study are the number of peaks an algorithm maintains and the number of replacement errors it makes. A peak is considered maintained if at least one population element (which has a fitness of at least 80% of the peak's optimal value) exists in the region of the search space corresponding to that peak. On the easy test problems of this study, the quality of final solutions will usually be similar across algorithms. We are thus more interested in the quantity of diverse, but good, final solutions; this corresponds to the number of peaks maintained.

The number of replacement errors constitutes our second performance criterion. Recall that both crowding and preselection replace existing population elements with new ones. We will define a replacement error as the replacement of a member of one peak by a member of another. The peaks of each test function are easy to distinguish, since they occur periodically every increment of .2 along the x axis. The primary goal of this study is to develop methods which minimize the number of replacement errors, and which, as a result, maintain the maximum number of peaks.

Speed is not a fundamental concern here, although it is often an important consideration. In short, varying the generation gap in crowding has an insignificant effect on algorithmic speed, while higher crowding factors significantly slow the algorithm. Preselection algorithms are faster than crowding, even crowding with low CF. They are typically similar in speed to a simple genetic algorithm.

6 Algorithms and Experimental Results

Six algorithms are tested, all of which are variations of crowding or preselection. Algorithm 0 represents a basic crowding algorithm, with typical parameter settings. Algorithms 1 through 4 are cumulative attempts to eliminate unnecessary parameters, and to reduce the number of replacement errors in the crowding algorithm. Algorithm 5 is an improved version of preselection which approximates crowding.

All algorithms use a population size of 100 and a mutation rate of zero. Strings are binary-coded and 30 bits in length. Populations are randomly initialized and are then run for $20,000$ function evaluations. 100 runs are performed for each algorithm.

Figures 1 and 2 show the distribution of population elements for a sample run of each algorithm on the test problems. Figure 3 shows the number of peaks each algorithm maintains over time, averaged over 100 runs. Figure 4 shows the number of replacement errors as each algorithm progresses, averaged over 100 runs. Figure 5 displays the number of diverse bit positions each algorithm maintains over time, also averaged over 100 runs.

6.1 Algorithm 0

Algorithm 0 (A0) is our starting point, and represents the variation of crowding used in Deb's (1989) study. Crossover probability (p_c) is .9, $G = .1$, and $CF = 3$.

A0 maintained two peaks each run, on both test functions (see Figures 1 through 3). This is consistent with Deb's results, where crowding nearly always maintained exactly two peaks on each test function. Arriving at an explanation for this behavior is not too difficult. A0 makes an average of 6600 replacement errors on F1 and an average of 5900 replacement errors on F2 (see Figure 4). Since crowding does a replacement for each function evaluation, nearly one out of three replacements is in error. This is an open invitation to genetic drift, so it is apparent why more than two peaks are not maintained.

With such high variance, should A0 be maintaining only one peak? Let us examine the mechanism by which an element is lost from one peak and gained on another. Suppose an element from peak a is awaiting insertion in the population. Crowding chooses CF candidates from the population and replaces the closest one. Assuming crowding makes no errors in comparison, an element of peak a will be replaced if it is among the CF. If an element of peak a is not among the CF, an element from some other peak will be lost. As long as all peaks other than a together contain CF or more elements, these other peaks are vulnerable to loss of members. However, when peak a contains $n - 1$ of the population's elements, and peak b contains the final one, peak b will not lose that element, since for $CF \geq 2$, an element of a will be present among the CF. Note that crowding with $CF = 1$ is effectively a simple genetic algorithm, where all peaks but one are eventually lost (Deb, 1989; Goldberg & Segrest, 1987).

6.2 Algorithm 1

Algorithm 1 (A1) is a simplification of A0. The intent of A1 is to get rid of parameters whose fine-tuning often detracts from a thorough analysis of an algorithm and its behavior. The use of crossover probabilities less than 1.0 somewhat delays the mixing of a population, but is not an effective method of preserving diversity. Likewise, generation gaps that are too high result in the replacement of too much of the population, with behavior approaching that of a simple genetic algorithm. De Jong himself noted that crowding becomes less effective as G increases. However, he avoided generation gaps less than .1, probably to allow a limited amount of parallelism. We will consider parallelism later.

A1 is identical to A0, except that $p_c = 1.0$, and $G = 1/(2n) = .02$. G is set to the lowest possible value which still allows the binary operator, crossover, to function. The settings for these two parameters resemble the settings used in GENITOR (Whitley, 1988), a one-at-a-time, generation-and-replacement genetic algorithm. As expected, A1 produces results nearly identical to those produced by A0 (see Figures 1 through 5).

6.3 Algorithm 2

Having fixed most parameters, we can now concentrate on reducing the number of errors. In the analysis of A0, we assumed perfect comparison. Perfect comparison requires that the algorithm never make an error when deciding which of the CF elements is closest to the one being inserted. For the binary-coded strings of this study, bitwise or genotypic comparison is far from ideal. Only the three most significant bits of each string are likely to be useful for distinguishing different peaks of the search space. Genotypic comparison is thus trying to detect about three bits of signal in the presence of about 27 bits of noise.

One solution is the *phenotypic* comparison measure used in the original study of sharing functions (Goldberg & Richardson, 1987). Phenotypic comparison uses decoded parameters rather than raw, binary strings. For F1 and F2, the decoded parameter is the real number, x. Although Deb (1989) tried phenotypic sharing, he did not attempt phenotypic crowding. Another potential solution is to use genotypic comparison, along with Gray (Hamming-distance) codes.

Algorithm 2 (A2) is identical to A1, except for its use of phenotypic comparison. As shown in Figures 1 through 3, A2 maintains two to three peaks, a marked improvement. The number of replacement errors has also dropped significantly (Figure 4).

6.4 Algorithm 3

A major source of error must still be eliminated: the sampling error due to low CF. De Jong noted that while higher crowding factors lose fewer bits, they also degrade on-line and off-line performance. In addition, high crowding factors limit parallelism. Nevertheless, we are not yet concerned with speed or parallelism, but rather with maintaining maxima, and with accuracy of final results.

It is not difficult to calculate the CF required to perform correct replacements with some specified probability, if one makes a few assumptions regarding the fitness function and the initial distribution of elements. Algorithm 3 (A3) settles for no fewer than 100% correct replacements. It examines every population element to find the closest match.

A3 is identical to A2, except that $CF = n$. Although this adds an order of complexity to the previous algorithms, it is useful for demonstrative purposes. A3 is similar in complexity to the method of sharing functions (Goldberg & Richardson, 1987), which also cycles through the

whole population, in this case to compute each element's shared fitness. Goldberg notes that a version of sharing which estimates shared fitness using population sampling (as in crowding), is a promising possibility. Although Deb compared genotypic crowding using sampling to full phenotypic sharing (no sampling), no one has yet compared equally powerful versions of both.

The percentage of replacement errors in A3 drops to nearly zero (about 0.1%). Errors are still made a fraction of the time, when the phenotypically closest element is at the boundary of a neighboring peak. A3 consistently maintains population elements at all five peaks of the search space (Figures 1 through 3). However, A3 also distributes population elements all across the search space, including inside the low-fitness valleys of F1 and F2. It is not immediately apparent whether A3 is exhibiting any convergence pressure, or if it locates peaks due to random distribution of population members. While A3 chooses two good solutions for reproduction each generation, it also overwrites the most similar elements of the population, whether they are better, worse, or equally fit.

Preliminary runs of A3 on a simple, linear, unimodal function indicate that the algorithm exhibits very slight, if any, selection pressure. This is not surprising if one examines the method by which traditional crowding (with low CF) converges. Crowding proceeds by selecting elements for reproduction according to fitness. If the closest element in the population is consistently replaced, crowding will very likely replace an element similar in fitness, resulting in little or no improvement. However, if a certain percentage of the time a dissimilar element is replaced in error, an improvement is likely since the element to be inserted is probabilistically of superior fitness. Crowding thus relies on replacement errors for its convergence. We later examine a method for restoring convergence pressure.

6.5 Algorithm 4

A curious, but not unexpected thing happens when we raise CF to n. A high percentage of the time, one or both parents are replaced. On F1 and F2, a parent was replaced 83% of the time. This suggests a method to approximate crowding: always replace the closest parent. This method has the pleasant side-effect of drastically reducing algorithmic complexity and run time to the level of the simple genetic algorithm. The CF parameter is no longer needed, since a parent rather than an element of a sample is replaced.

Algorithm 4 (A4) is identical to A3, except that instead of sampling the whole population to find the phenotypically closest element, it instead samples only the parents. There are two possible replacements of two parents by their two offspring: offspring 1 replaces parent 1 and offspring 2 replaces parent 2, or offspring 1 replaces parent 2 and offspring 2 replaces parent 1. The pair of replacements which yields the greatest sum of phenotypic similarity between offspring and replaced parent is used.

Although the number of replacement errors remains very low (0.7%), most solutions have gravitated towards the valleys rather than the peaks of the two functions. The algorithm actually exhibits some very slight reverse selection pressure, with cumulative effects. Somewhat surprising at first, this behavior can be explained as follows. Recall that crowding chooses two elements for reproduction according to fitness. While in A3, these chosen elements might or might not be replaced by their offspring, in A4 these elements are always replaced by their offspring. A4 is in effect killing off the best elements of the population, and replacing them by elements which, though similar, are more often worse than better. This results in the slight downward selection pressure displayed. A3 does not display such downward pressure, due to the small percentage of the time (about 17%) when a parent is not the closest element in the population. A3, however, does not display much, if any, upward selection pressure.

6.6 Algorithm 5

What remains is to add some selection pressure to A4. Note that A4 resembles Cavicchio's preselection, except that fitness-proportionate selection is used, parents are always replaced regardless of offspring fitness, and a similarity measure is used to decide which of the two offspring replaces which parent.

Since the fitness-proportionate selection of A3 and A4 results in little to no favorable selection pressure, A5 dispenses with it. Instead, A5 pairs all population elements randomly each generation, without replacement. This means that each population member reproduces once each generation. A5 has the additional effect of adding true parallelism, since each generation, all population elements can proceed simultaneously. The algorithm is also simplified since the G parameter is no longer necessary.

To add back selection pressure, A5 only replaces a parent if the competing offspring is better, as in preselection. A5 differs from preselection in that it processes two parents and two offspring at a time, using the phenotypic similarity measure discussed under A2, to determine which offspring competes against which parent. Preliminary results have shown that replacing an arbitrary parent or replacing the worse parent, as preselection does, results in too many replacement errors to achieve multimodal function optimization.

As shown in Figures 1 through 3, A5 is successful at clustering solutions about all five peaks of both F1 and F2. As in A3, replacement errors are almost nonexistent (0.2%).

7 Extensions

The maintenance of bitwise diversity, as illustrated in Figure 5, was earlier referred to as a questionable goal. Let us now examine an instance of why this is so. Figure 5 shows that the traditional crowding techniques, A0 and A1, have lost a few bits; this is consistent with De Jong's results. A2, which uses phenotypic comparison, has lost a large majority of its bits. With phenotypic comparison, difference in Hamming space is not important, but difference in decoded parameter space is. A3 has gained back a lot of bitwise diversity due to its near elimination of replacement error. A4 and A5, on the other hand, seem to be anomalies, since they never lose a single bit. Let us take a closer look at one point to which A5 has converged. Notice that the optimum, $x = .5$, exists on a Hamming cliff, with one neighboring solution containing 29 zeros, and another containing 29 ones. It is not surprising that no bits are lost, since the final population contains both of these neighboring solutions. This form of bitwise diversity is not very useful.

Some would argue for the use of Gray codes (Caruana & Schaffer, 1988). One untested possibility is to revert to genotypic comparison, but to use the more complementary Gray codes.

The primary argument for maintaining bitwise diversity is that such diversity will somehow prevent premature convergence. As suggested above, and as can be demonstrated with the use of low mutation rates, premature convergence may still occur. What one would like is a mechanism for controlling convergence. One possibility is the addition of simulated annealing controls (Mahfoud & Goldberg, 1992).

The simple algorithms outlined here are prime candidates for Markov chain analysis. Exhaustive analysis is possible for reasonable population sizes and numbers of optima. The functions and parameter settings in this paper are not too unreasonable, since the resulting Markov matrices are sparse. Markov chain analysis is currently underway.

A comparison of the crowding technique, A5, to the method of sharing functions (Goldberg & Richardson, 1987) would also be useful. Sharing functions allocate individuals to peaks based on fitness or peak height. Intuitively, one can expect A5 to maintain the pre-existing number of individuals at each peak. If initialization is uniform, one can expect the algorithm to allocate individuals to peaks based on the percentage of the search space leading up to a peak. However, A5 should not strictly be performing parallel hillclimbing, since it is possible for a cross between elements of two peaks to lead to a better, but previously unexplored region of the search space.

8 Conclusion

Crowding and preselection are naturally motivated genetic algorithms which strive to maintain population diversity, and which can be applied to multimodal function optimization. Prior to this study, the two techniques were not developed to their full potential; replacement errors diminished their effectiveness. Even now, there may still be room for improvement.

Preselection is an unfortunate name for parental replacement techniques because it implies performing a type of selection before regular selection takes place. In this study, preselection is the only type of selection which occurs.

We have presented an algorithm which virtually eliminates replacement errors, is effective at multimodal function optimization, is simple, is fast, and is parallel. This algorithm incorporates a similarity measure like that of crowding, along with the selection pressure of preselection. The algorithm, earlier called A5, will henceforth be known as *deterministic crowding*.

Recently, many attempts at preserving diversity have abandoned so-called *panmictic* methods in favor of those which enforce some kind of geographic or mating isolation. A conclusion that can be drawn from this paper is that panmictic methods still possess unexplored potential.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant ECS-9022007. I would like to thank David Goldberg for many valuable discussions related to this paper.

References

Caruana, R. A., & Schaffer, J. D. (1988). Representation and hidden bias: Gray versus binary coding for genetic algorithms. *Proceedings of the Fifth International Conference on Machine Learning*, 153-161.

Cavicchio, D. J. (1970). *Adaptive search using simulated evolution.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International, 36*(10), 5140B. (University Microfilms No. 76-9381).

Deb, K. (1989). *Genetic algorithms in multimodal function optimization* (Masters Thesis and TCGA Report 89002). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E., & Richardson, J. J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, 41-49.

Goldberg, D. E., & Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, 1-8.

Mahfoud, S. W., & Goldberg, D. E. (1992). *Parallel recombinative simulated annealing: a genetic algorithm* (IlliGAL Report 92002). Urbana: University of Illinois, Illinois Genetic Algorithms Lab.

Whitley, D., & Kauth, J. (1988). GENITOR: a different genetic algorithm. *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, 118-130.

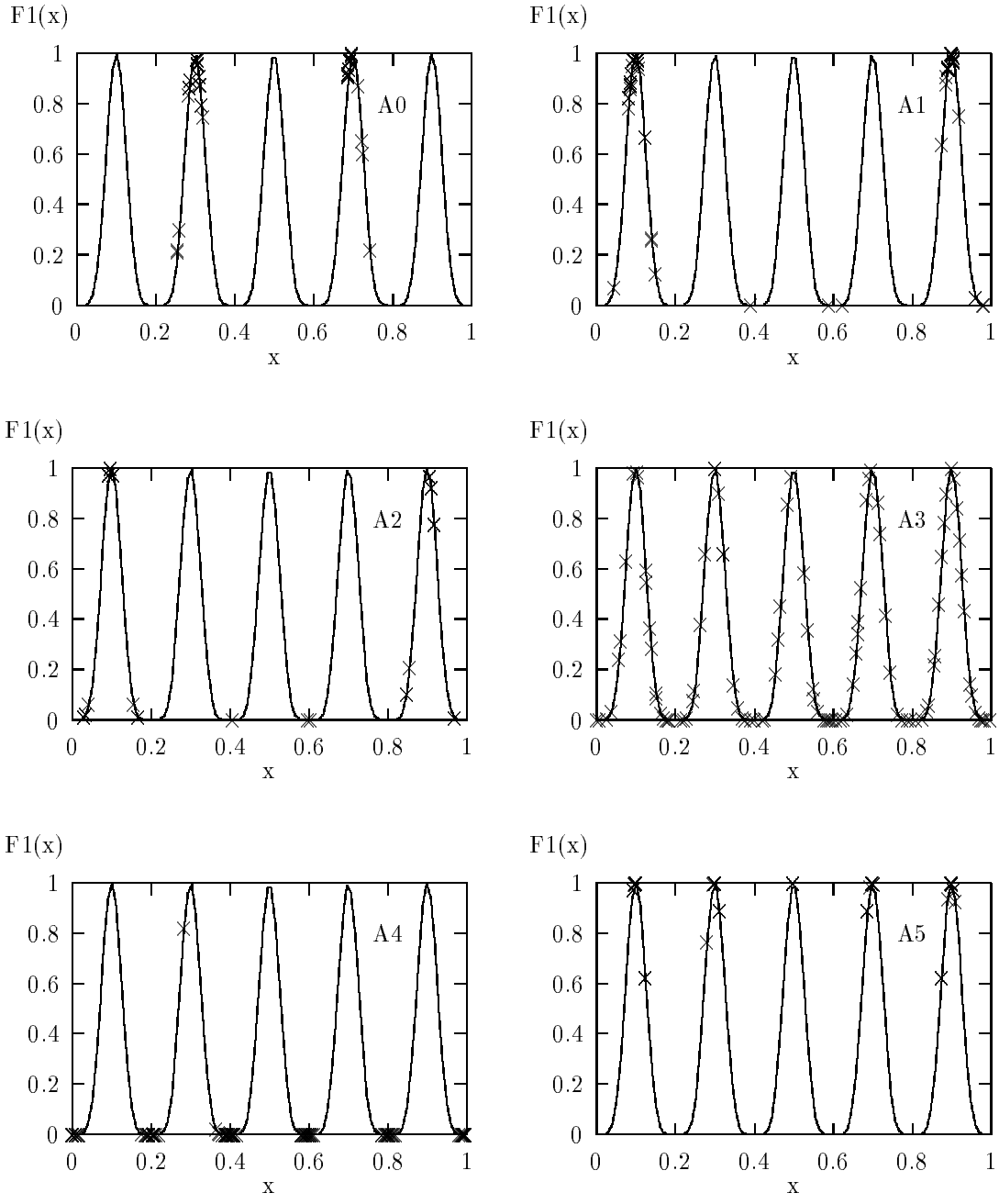


Figure 1: The final distribution of 100 population elements is shown for one run of each algorithm, A0 to A5, on function F1. Each algorithm is run for 20,000 function evaluations.

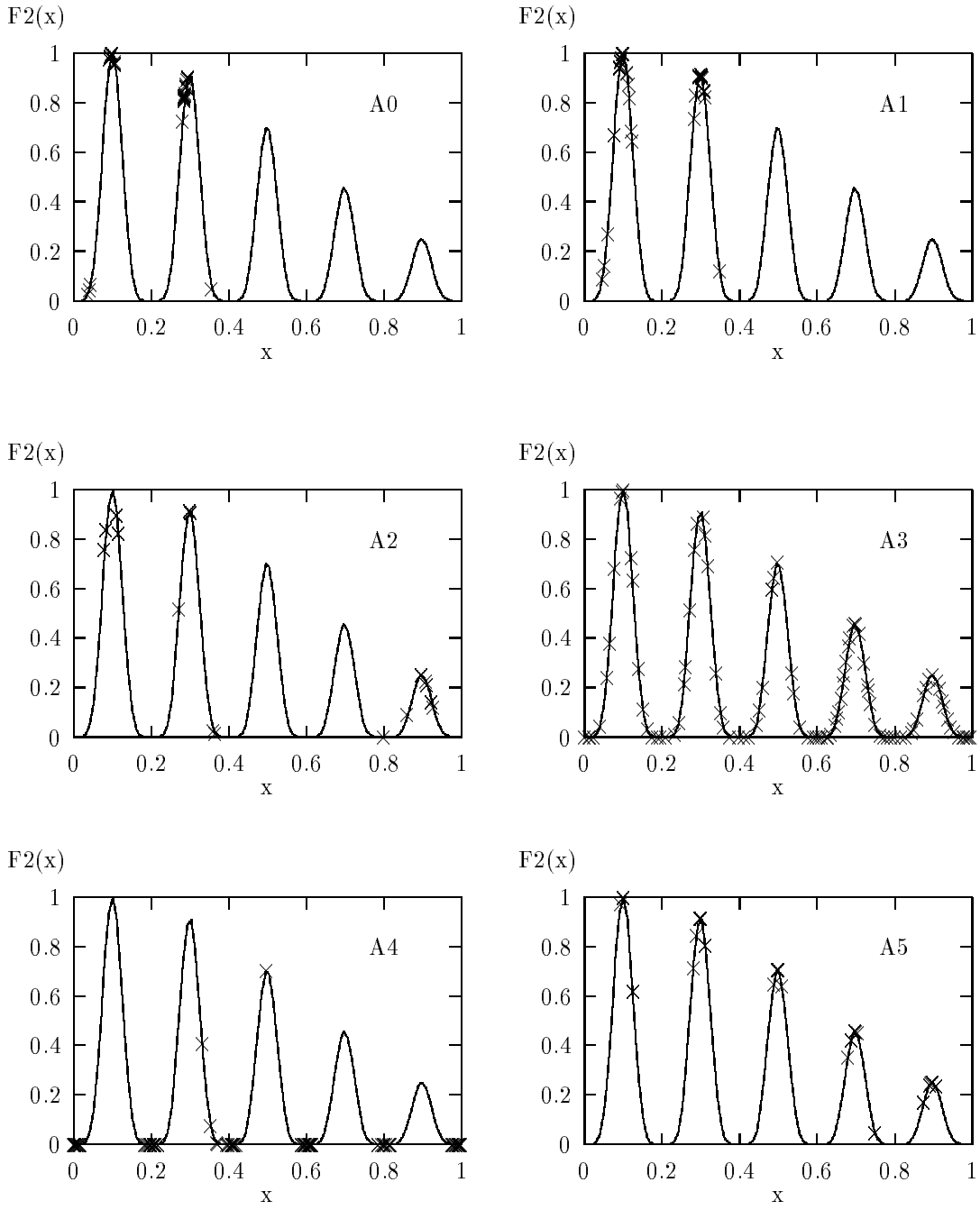
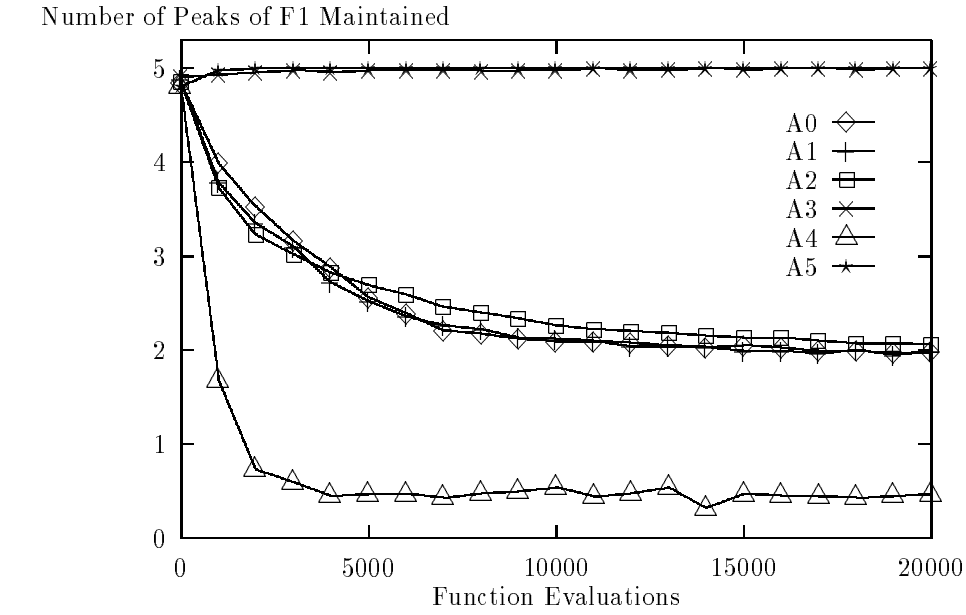


Figure 2: The final distribution of 100 population elements is shown for one run of each algorithm, A0 to A5, on function F2. Each algorithm is run for 20,000 function evaluations.



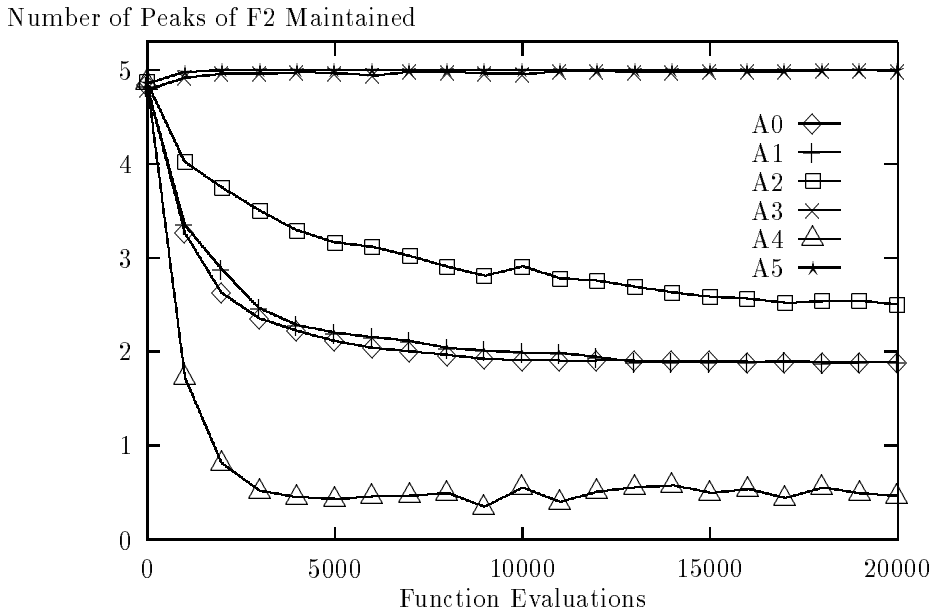


Figure 3: The number of peaks maintained by each algorithm, A0 to A5, is shown as a function of time, averaged over 100 runs. A peak is considered maintained if some population element exists in the region of the peak, whose fitness is at least 80% of the peak's height.

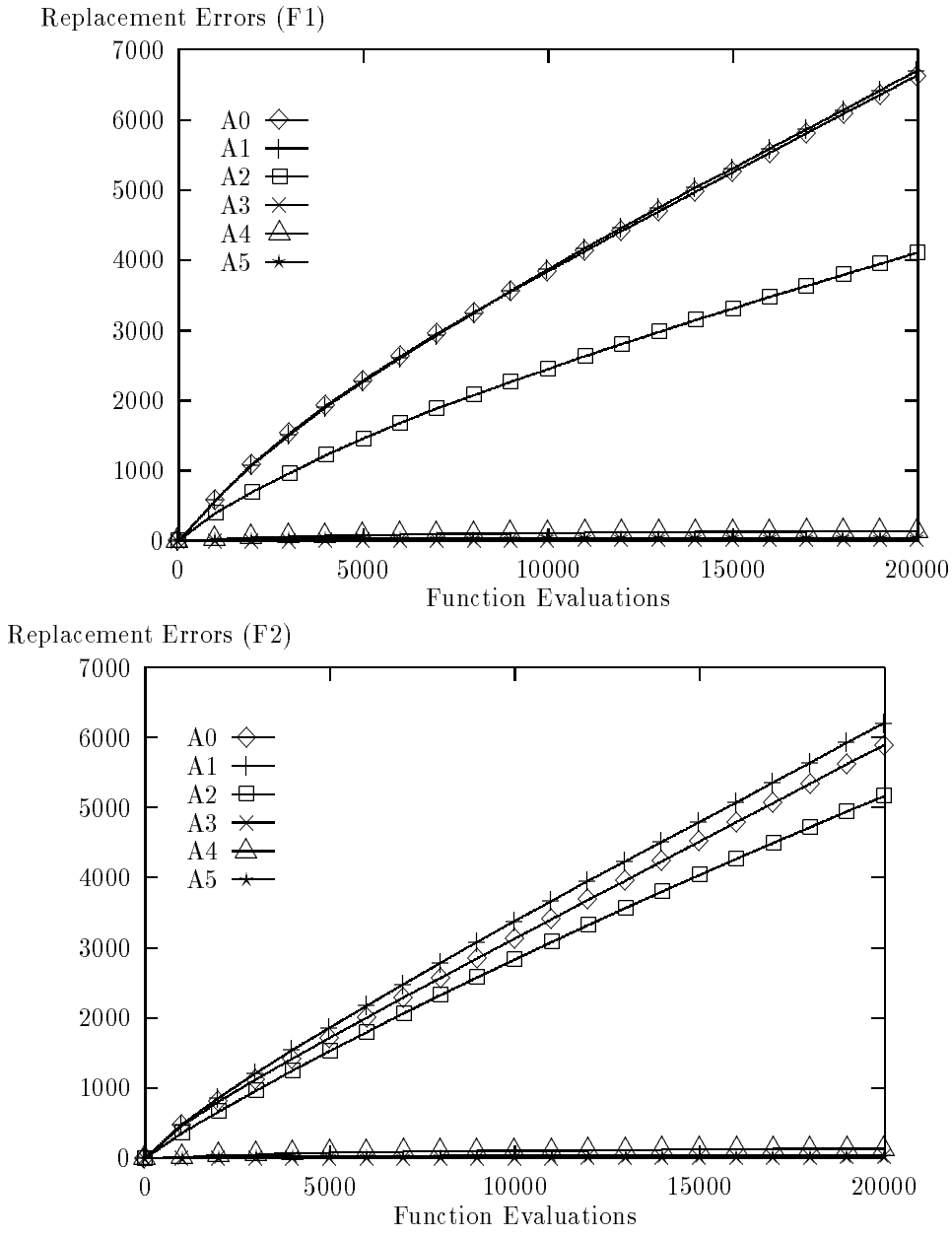


Figure 4: The total number of replacement errors for each algorithm, A0 to A5, is shown as a function of time, averaged over 100 runs.

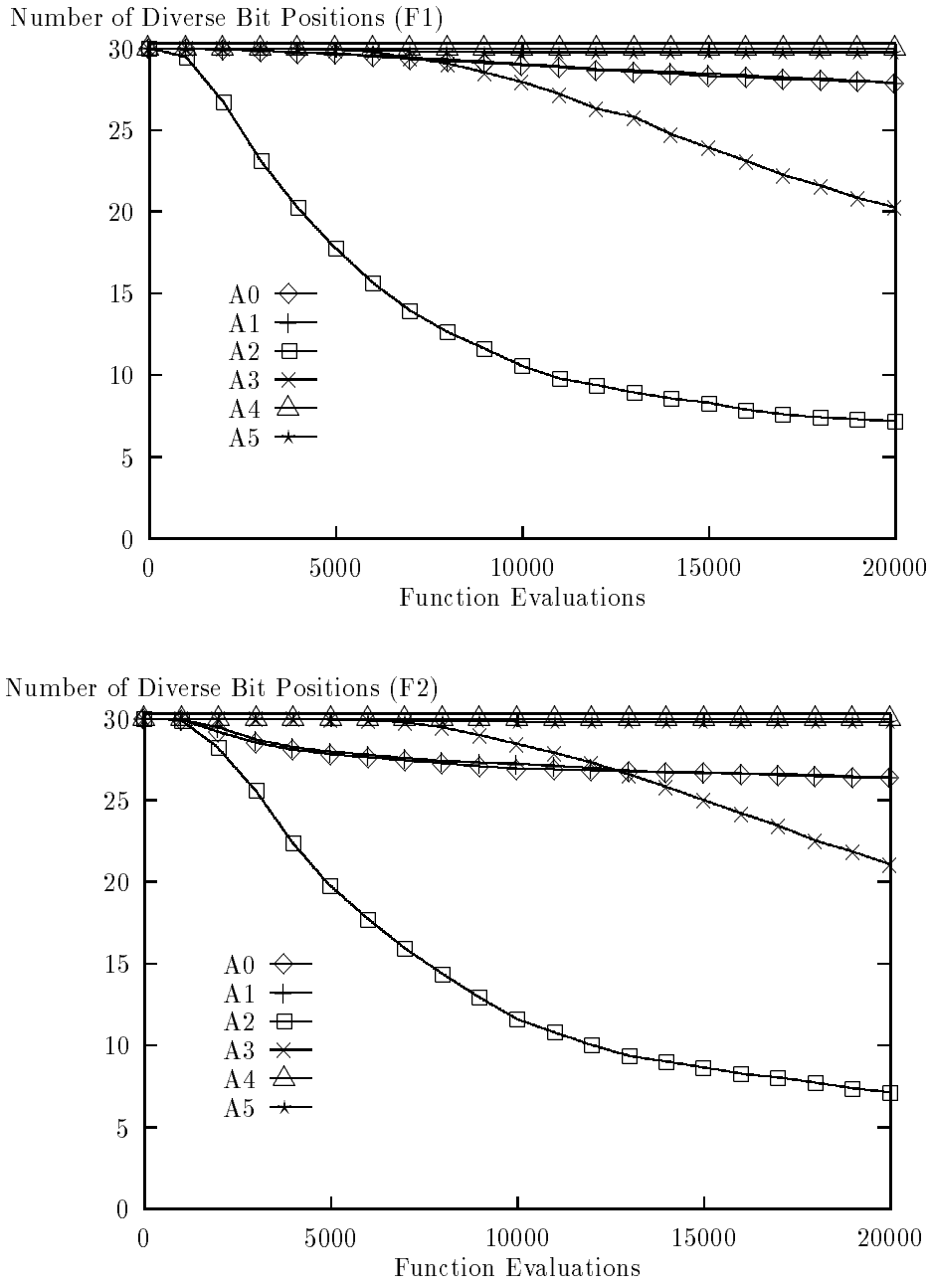


Figure 5: Bitwise diversity maintained by each algorithm, A0 to A5, is shown as a function of time, averaged over 100 runs. A bit position is considered diverse if both possible values occur at least once in the population.