Simple Subpopulation Schemes †

William M. Spears
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory - Code 5510
Washington, D.C. 20375-5320 USA
E-mail: SPEARS@AIC.NRL.NAVY.MIL

ABSTRACT

This paper considers a new method for maintaining diversity by creating subpopulations in a standard generational evolutionary algorithm. Unlike other methods, it replaces the concept of distance between individuals with tag bits that identify the subpopulation to which an individual belongs. Two variations of this method are presented, illustrating the feasibility of this approach.

1. Introduction

One of the attractive features of an evolutionary algorithm (EA) is that it quickly concentrates effort in promising areas of a search space. Unfortunately, this feature is not always advantageous for many applications of interest. In such cases the attractive feature is described in a negative tone as "premature convergence". Although this phrase is subjective, it refers to the phenomenon in which the EA loses population diversity before some goal is met. Two typical cases are the optimization of multimodal functions and covering problems. In the former case the EA may concentrate effort on some suboptimal peak. In the latter case the problem is even more severe - the EA user wishes to simultaneously find a number of peaks, but the EA has lost all diversity and concentrates on one peak only.

Thus, one recurring theme in EA research is how to maintain population diversity. The mutation operator is often suggested as a solution. However, high mutation rates, although certainly increasing population diversity, are often too destructive. In other words, the price of continuing exploration is that good solutions are often lost. The general consensus, then, is that diversity for the sake of diversity is not the issue. Rather it is the *appropriate* use of diversity, to explore new areas while not destroying the information already learned. One natural and appealing method of accomplishing this is by allowing the EA to evolve subpopulations. Each subpopulation can then explore a

† This paper appears in the proceedings of the 1994 Evolutionary Programming Conference, published by World Scientific.

separate portion of the search space.

In a traditional EA, every generation a population of N individuals reproduces according to their fitness (given by some objective function) and creates a set of N offspring via the application of genetic operators. The N parents are then replaced by the N offspring to produce the next generation. One method for evolving subpopulations in a generational EA is through the use of *sharing* and *restricted mating* [6]. With sharing, the similarity of individuals within the population is used to dynamically modify the fitness of those individuals. This dynamic modification is internal to the EA, thus what is changing is the EA's perception of the objective function, not the objective function itself. The intuition is simple - the peaks in a space are treated as resources and individuals near one peak have to share the resource of that peak. Overcrowding on one peak implies that a resource is overused. In this case the perceived fitness of the peak goes down, reducing selective pressure in that area of the space. On the other hand, peaks with few individuals have their perceived fitness increased, increasing selective pressure in those areas. The net effect is to apportion individuals in rough proportion to the relative height of the peaks in the space. Furthermore, the similarity metric is also used to restrict mating (crossover) to those individuals that are most similar.

Although sharing and restricted mating are rather general ideas, the implementation of those ideas that has received the most attention is by Goldberg and Richardson. [6] In their implementation, sharing and restricted mating together work very well, creating stable subpopulations on many peaks within the search space. Furthermore higher peaks get more individuals than lower peaks, in rough proportion to the relative heights of the peaks, as we would expect with the sharing mechanism. Unfortunately, Goldberg's implementation makes two assumptions. The first is the number of peaks in the space. The second is that those peaks are uniformly distributed throughout the space. However, no sensitivity study indicates how well these assumptions must be met. Finally, the implementation (as described by Goldberg and Richardson) is also expensive. The similarity of each pair of individuals is measured, resulting in $O(N^2)$ similarity comparisons.

One common variation of the generational EA model is to produce a small number of offspring ($<< N$) each generation. This is referred to as a *generation-gap* or *steady-state* model. Since only a few offspring are created at a time, it is now necessary to choose which parents those offspring should replace, since almost all EAs assume a static population size.† Historically, *crowding* is one of the earliest techniques for the maintenance of diversity and creation of subpopulations in a steady-state EA. [5] With crowding, each child will replace an individual that is most similar to itself, from a small set of sampled parents. Computationally this is not an expensive operation. However, when tested on functions with multiple peaks, crowding is only able to create subpopulations on a few of the peaks. [6]‡

---

† It is of course intriguing to imagine a dynamic population size, in which individuals have an expected lifetime. However, little has been done in this area.

‡ Mahfoud suggests, however, that a combination of crowding and preselection may form more

Of course, parallel architectures provide another obvious method for simultaneously exploring separate portions of a search space. In a parallel EA, a topology is imposed on the EA population, typically dividing the population into overlapping subpopulations. In this paper we are concerned with sequential EAs, however. Rather than simply run a parallel EA on a sequential machine, we will concentrate on EAs that are specifically designed for sequential machines. One advantage of the sequential approach is that it is easier to maintain global control over the subpopulations, resulting in subtle yet important semantic differences between the sequential and parallel algorithms. We will discuss this further at the end of this paper.

In summary, the current implementations of sharing and restricted mating work well but make strong assumptions and are expensive. Crowding, on the other hand, is simpler yet appears to lack the necessary power to create a reasonable number of subpopulations. Rather than explore the space between these techniques, another possibility is pursued in this paper - namely, dropping the distance metric altogether.

The motivation for this decision comes from nature. I do not decide I'm Portuguese because I'm in some sense similar to a few of my cousins. Rather, I decide I'm Portuguese because my ancestors were labelled as Portuguese. Although I'm not suggesting that I genetically inherited a Portuguese "label", I certainly culturally inherited that label. What if each EA individual has a label? Similarity then becomes simply a matter of seeing if two individuals have the same label. Of course, this implies that everyone with the same label is equally similar. The results of this paper suggest that the added precision of the distance metric is often not needed.

Labels, then, can replace the distance metric used in the implementations of the above techniques. By implementing restricted mating and sharing with labels, both the efficiency of crowding and the efficacy of restricted mating (and sharing) can be simultaneously accomplished. The remainder of this paper will investigate a class of evolutionary algorithms that assumes that individuals have labels.†

2. Simple Subpopulation Scheme #1

This section will describe an EA that creates subpopulations via the use of labels on individuals. At this time the evaluation of the algorithm will be fairly subjective, namely - how well the algorithm finds multiple peaks in a multimodal space. Such an evaluation is not without precedent - Goldberg and Richardson perform exactly this test. [6] We are also less concerned with specific performance measures on individual problems, and concentrate more on the trends in performance as various factors change (such as the problems or the number of subpopulations). In light of these considerations we examine a set of four one-variable real-valued functions borrowed from Beasley. [1] ‡ Figure 1 illustrates

---

subpopulations. [2,8]

† I'm certainly not the first to suggest this. Perry explores the use of labels as well. [9]

‡ Without loss of generality we will assume we wish to maximize. Also, we follow Goldberg and

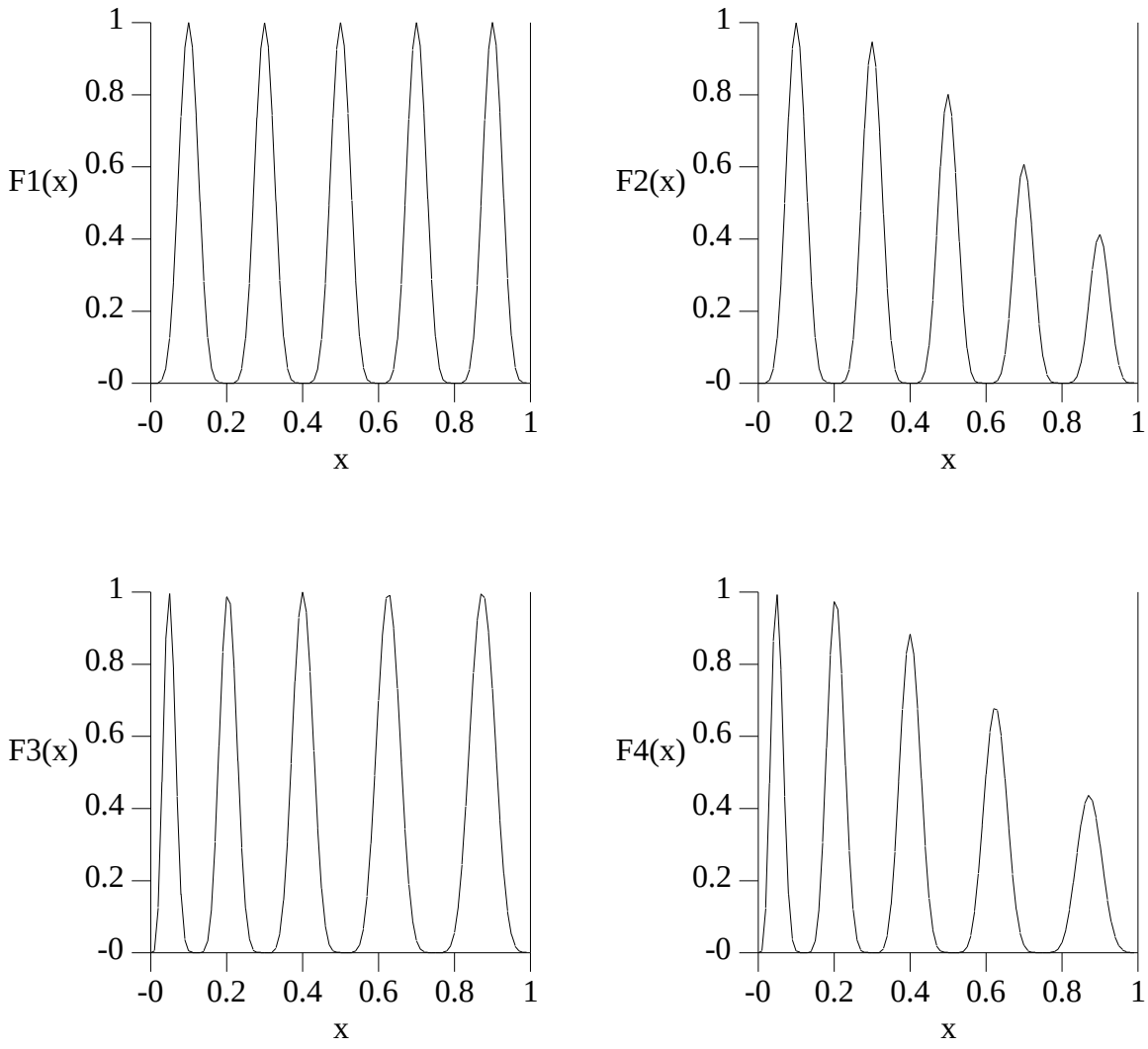


Figure 1: Four test functions

the four functions.

Intuitively, with respect to the creation of subpopulations, F1 should be the easiest function. Since each peak is of equal height, no one peak will dominate the search. Furthermore, the equal spacing between peaks matches the assumption of Goldberg's algorithm. F3, with the unequal spacing, should be more challenging for any algorithm that makes an equal spacing assumption. Finally, F2 and F4 will be most difficult because the highest peak will have the tendency to dominate the search. At the very

---

Richardson and use 30-bit binary encoded individuals for our representation. [6]

least, any algorithm that purports to create subpopulations should be able to locate (and stay at) most of the peaks of each function.

As mentioned before, we will attach a label to each EA individual. Although this label can be arbitrary, we will concentrate on the addition of tag bits to bit string individuals.† Thus, n tag bits can be used to represent 2^n subpopulations. Each individual lies in the unique subpopulation described by its tag bits. Zero tag bits refers to the case of one subpopulation, i.e., a standard EA. Suppose we now add a restricted mating scheme similar to that used by Goldberg. This is easily implemented by allowing mating to occur only between individuals within the same subpopulation.

When this algorithm is run on F1 or F3, it typically forms stable subpopulations on a number of peaks.‡ As the number of tag bits is increased, the number of peaks found increases. The population size, however, prevents one from increasing the number of tags without limit. For example, with a population size of 100, five tags bits (32 subpopulations) implies that each subpopulation has a very small size. If more subpopulations are required, the population size must be increased.

When this algorithm is run on F2 or F4, however, the subpopulations quickly converge to the highest peak. In this case selective pressure focuses attention to the area of the space with the most payoff. Clearly, some form of fitness sharing is required to reduce overcrowding on the higher peaks. Suppose that in every generation we count the number of individuals in each subpopulation. Then we can normalize the perceived fitness by dividing the fitness of each individual by the size of its subpopulation. Thus, a large subpopulation decreases the perceived fitness of a peak, allowing search to be focussed in other areas.

To get an intuitive understanding of how this algorithm will work, let us consider the following thought experiment. Suppose we have a simple function with two peaks, one peak twice as high as the other, and further suppose we allow one tag bit for each individual. Each tag bit bit is randomly initialized, so at the beginning of the run we have two subpopulations of roughly equal size. Due to random sampling both subpopulations could eventually settle in on the higher peak, or both could settle in on the lower peak. However, in some cases (again due to random sampling), each subpopulation will head towards different peaks. If we did not have fitness sharing, the individuals on the higher peak would always get more children than the individuals on the lower peak and eventually the subpopulation on the lower peak would vanish. However, with fitness sharing, the higher peak can support only twice as many individuals as can be supported on the lower peak (since it is only twice as high). In other words, the higher peak can support roughly 2/3 of the individuals, while the lower peak supports 1/3. To see this more

---

† The reader should note that all the techniques introduced in this paper will work on non-bit string representations with minimal change.

‡ Stability is hard to quantify. However, following Deb and Goldberg we will claim stability if a subpopulation survives more than 200 generations with a population size of 100. [4]

clearly, suppose the population is of size 30 and that the fitness of the higher peak is 4, while the fitness of the lower peak is 2. Then, if we have 20 individuals at the high peak, their perceived fitness is 4/20. Meanwhile the remaining 10 individuals at the low peak have perceived fitness 2/10. The fitness sharing mechanism has dynamically adjusted the perceived fitness so that the two peaks have the same perceived height. The result is that both subpopulations can survive in a stable fashion. Furthermore, restricted mating prevents crossover between individuals on the two peaks, which could often result in low fitness individuals.

SSS1 is defined to be the algorithm that combines both the restricted mating and this simple sharing mechanism (see Figure 2).

```
procedure SSS1; {
initialize();
fitness_with_sharing();
until (done) {
      parent_selection();
      mutate();
      shuffle_within_subpops();
      crossover_within_subpops();
      fitness_with_sharing();
}       }
```

Figure 2: The SSS1 algorithm

When tested on functions F1 through F4, SSS1 is able to maintain stable subpopulations on all but the lowest one or two peaks. Lower peaks are dropped because a subpopulation will often have individuals lying on two or more peaks in the early generations. In this case selective pressure within a subpopulation focuses attention on the highest part of the space it can see. Thus the algorithm exhibits the feature that subpopulations can flow to higher portions of the space, in a fashion similar to that of the population in a standard EA. Although SSS1 doesn't cover all peaks, it appears to be useful in situations where the goal is to find multiple high peaks.

SSS1 meets many of the goals outlined earlier. The algorithm is simple and efficient. It does not make the equal spacing assumption. Since different subpopulations may lie on the same peak, the algorithm also does not make strong assumptions about the number of peaks in the space. One should be careful, though, to have more subpopulations than the number of peaks you wish to find. Both restricted mating and sharing can be executed in O(N) time, due to the presence of the tag bits. Diversity is usefully maintained and the EA can divide attention among many areas of high fitness. In comparison with crowding, SSS1 has the same order of complexity, yet yields superior results. It also does not require a generation gap and can work with any standard generation EA. In

comparison with Goldberg's algorithm, SSS1 is simpler to implement, much less costly, and offers similar results. SSS1 does have trouble with low peaks, unlike Goldberg's algorithm. This is largely due to the use of tags. With SSS1 each subpopulation can move from one peak to another. In Goldberg's algorithm, there are no explicit subpopulations, and if the lower peak is sufficiently far from other peaks it can be singled out for attention by using the distance metric. Thus Goldberg's algorithm buys greater precision at a greater cost.

In the next section, we will explore the possibility that SSS1 can be modified to behave more like Goldberg's algorithm, without an increase in complexity. However, first a note on mutation is in order. Since mating is restricted to those individuals in the same subpopulation, the tag bits will not be modified by crossover. Mutation, however, if allowed to operate on the tag bits, can provide a method for moving individuals from one subpopulation to another. This effect, referred to as *diffusion*, is controlled by the amount of mutation on those tag bits. No mutation implies no diffusion while higher mutation rates imply more diffusion. Higher diffusion rates result in lower stability, i.e., subpopulations have a greater probability of disappearing from peaks. For the experiments presented in this paper we do not allow mutation on the tag bits, since we wish to see the maximum stability of the system.

3. Simple Subpopulation Scheme #2

As mentioned above, SSS1 can drop lower peaks because competition within a subpopulation will allow it to move to a higher peak. One possibility is to use very small subpopulations. Unfortunately, extremely small subpopulations can lead to poor search. Furthermore, small subpopulations can easily die due to stochastic effects. Another idea is to restrict mating even further. Instead of mating with only those in your subpopulation, suppose one can mate only with neighboring individuals (in the same subpopulation). Since the whole subpopulation is subject to the same sharing normalization, individuals on a lower peak should be able to "piggyback" off the individuals on the higher peak. A useful analogy is to consider a remora on a shark. Let us consider the shark to be more fit, individually, than the remora. Competition between the two would be to the disadvantage of the remora. However, by "attaching" the remora to the shark, we help remove the competition between the two organisms. The (perceived) fitness of the remora has been increased, due to its association with the shark. As a result, the remora has a much higher chance of survival.

The concept of a neighbor adds a topology to the population, an idea that is familiar to parallel EA researchers (see Davidor for a discussion). [3] In this case an obvious topology is a ring, so that every individual in a serial population has two neighbors (one to its right and one to its left). SSS1 with the topology added is referred to as SSS2 in this paper (see Figure 3).

As would be expected, results with SSS2 on F1 and F3 are very similar to the results with SSS1. On F2 and F4, however, SSS2 is markedly better at maintaining

```
procedure SSS2; {
initialize();
fitness_with_sharing();
until (done) {
      parent_selection();
      mutate();
      crossover_neighbors_within_subpops();
      fitness_with_sharing();
}      }
```

Figure 3: The SSS2 algorithm

subpopulations at lower peaks, for a given population size and number of tag bits. Stable subpopulations are consistently formed at the fourth peak, and often formed at the fifth peak. Computationally, SSS2 is actually more efficient that SSS1 (because shuffling is no longer performed), but exhibits behaviour very similar to Goldberg's algorithm.† The reader should note, however, that SSS2 differs from SSS1 in another fashion as well. Unlike SSS1, zero tags bits in SSS2 does not yield a standard EA.

Despite the success of SSS2, it is rather unusual in that it uses topology to restrict mating, but not to effect sharing. In other words, sharing still only relies on tag bits, but not on the topology, whereas restricted mating relies on both the tag bits and the topology. An obvious modification is to treat sequential clusters of individuals with identical tags as subpopulations in their own right. For example, suppose there are 15 individuals. The first five have tags of 111, the second five have tags of 100, and the last five have tags of 111. With SSS1 or SSS2 this is treated as two subpopulations. Now we have three, with the boundaries occurring where the tag bits change. One effect of this modification is that n tag bits can result in far more than 2^n subpopulations. In fact, one tag bit may be sufficient. What we have lost, however, is some control over the number of subpopulations that can arise, and their size. Of course, this too may be an advantage in some situations. This new algorithm (SSS3) will be further explored in future work.

4. Higher Dimensional Problems and Subpopulation Dynamics

As mentioned earlier, F1 through F4 are one-variable real-valued problems. We have also experimented with higher dimensional problems, and while both SSS1 and SSS2 successfully create stable subpopulations, it is hard to visually depict this behaviour when the number of dimensions exceeds two. However, we can show perspective plots of two-variable problems and display the EA population on that plot. The

---

† Actually, if one considers genotypic implementations, SSS2 maintains more subpopulations on lower peaks than does Goldberg's algorithm. [4]

last page of this paper is an example of the performance of SSS2 on an interesting two-variable problem. SSS2 was run until 500 generations, with a population of 100 and 16 subpopulations. The large dots represent some of the individuals from the last generation. As can be seen, SSS2 located five of the six peaks.

Although this paper is in part motivated by the application of EAs to multimodal function optimization and covering problems, it is also worthwhile to examine the dynamics of these subpopulation algorithms. For example, if sharing is working reasonably well, we would expect the size of subpopulations to roughly reflect the fitness of the particular peaks they reside on. Thus subpopulations on higher peaks should be larger than subpopulations on lower peaks. As an example we ran SSS2 on function F4 with a population of 200 and 16 subpopulations, and monitored the size of those subpopulations from generation to generation. Although it is impossible to reasonably display the results for all 16 subpopulations, it is feasible to present a subset. In Figure 4 we show the time evolution of four subpopulations during that run.

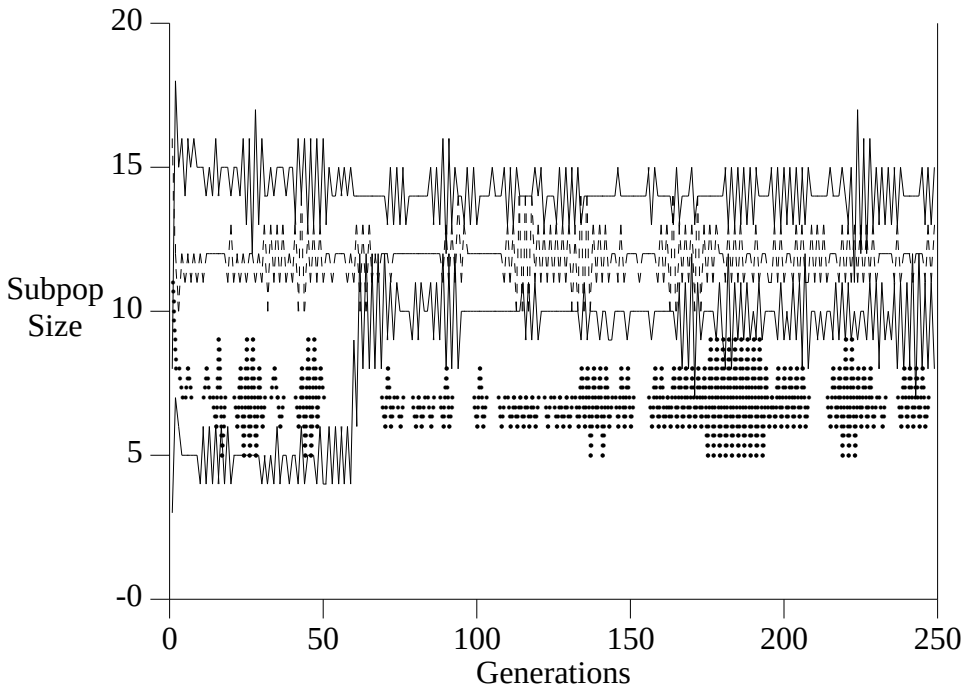


Figure 4: Time evolution of subpopulation size

Figure 4 illustrates a number of features. If the peaks were of equal fitness, we would expect to see roughly 200/16 = 12.5 individuals per subpopulation. However, since this is not the case, we should expect some subpopulations to have more than 12.5 individuals, while others have less. Interestingly, even with only the four subpopulations,

one can distinguish five different levels for the subpopulation sizes, one level for each of the five peaks. It is also interesting to note that we can see one subpopulation flow from one size level to another, indicating that a subpopulation has flowed to a higher peak, as discussed earlier in this paper. Finally, Figure 4 also illustrates that the rather simple fitness sharing mechanism we have used results in a fair amount of variance in the size of a subpopulation over time. One implication of this observation is that it might be better to perform sharing over a few generations, rather than on a generation by generation basis (providing some averaging to reduce the variance). We intend to explore this option in the future.

5. Summary and Discussion

This paper considers a new method for maintaining diversity and creating subpopulations in a standard generational EA. The distance metric assumed by the standard implementations of crowding, restricted mating, and sharing is replaced by tag bits that identify the subpopulation to which an individual belongs. Two variations of this method are presented. In the first (SSS1) the tag bits are used to restrict mating and provide an efficient sharing mechanism. SSS1 appears to be useful at finding the high peaks in the search space. SSS1, however, often moves from lower peaks to higher. By adding a topology to help restrict mating further, it is shown that the resulting algorithm (SSS2) has the ability to more easily maintain individuals on lower peaks. Another modification of this algorithm, SSS3, is discussed, although not implemented. In SSS3, the topology can be used for both restricted mating and sharing.

The SSS algorithms are efficient because they do not have to make distance comparisons. Interestingly, it is also possible to make Goldberg's implementation of sharing more efficient by sampling. [7] In other words the distance of each individual from the rest is estimated by using a subset of the remaining individuals. It will be intriguing to find out whether this change results in behaviour similar to that of the SSS algorithms.

Finally, this work has similarities to the EA research performed on parallel architectures. In a parallel EA, a topology is imposed on the EA population, resulting in subpopulations. However, there are some important differences between the parallel approaches and our sequential approach. In the SSS algorithms, the fitness of an individual and the subpopulation size are dynamic, based on the other individuals (and subpopulations). As a consequence, the SSS algorithms can concentrate effort on more promising peaks, while still maintaining individuals in other areas of the search space. This is typically not true for parallel EAs implemented on MIMD or SIMD architectures. When using a MIMD architecture, subpopulations are dedicated to particular processors and the subpopulations remain a constant size. In SIMD implementations, one or two individuals reside on a processor, and subpopulations are formed by defining overlapping neighborhoods. However, due to the overlap, one particular subpopulation will eventually take over the whole population. The advantages and disadvantages of these semantic differences will be explored in the future.

Acknowledgements

I would like to thank Diana Gordon, the Machine Learning Group at NRL-AIC, and the INFT910 class at George Mason University for their valuable feedback on all aspects of this work.

References

1. Beasley, D., Bull D. R., and Martin, R. (1993) *A sequential niche technique for multimodal function optimization.* Evolutionary Computation Journal, Volume 1, Number 2, Summer 1993.

2. Cavicchio, D. J. (1970) *Adaptive search using simulated evolution.* Doctoral Thesis, University of Michigan, Ann Arbor.

3. Davidor, Y. (1991) A naturally occurring niche & species phenomenon: The model and first results. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 257-263. La Jolla, CA: Morgan Kaufmann.

4. Deb, K. and Goldberg, D. E. (1989) An investigation of niche and species formation in genetic function optimization. *Proceedings of the Third International Conference on Genetic Algorithms*, 42-50. Fairfax, VA: Morgan Kaufmann.

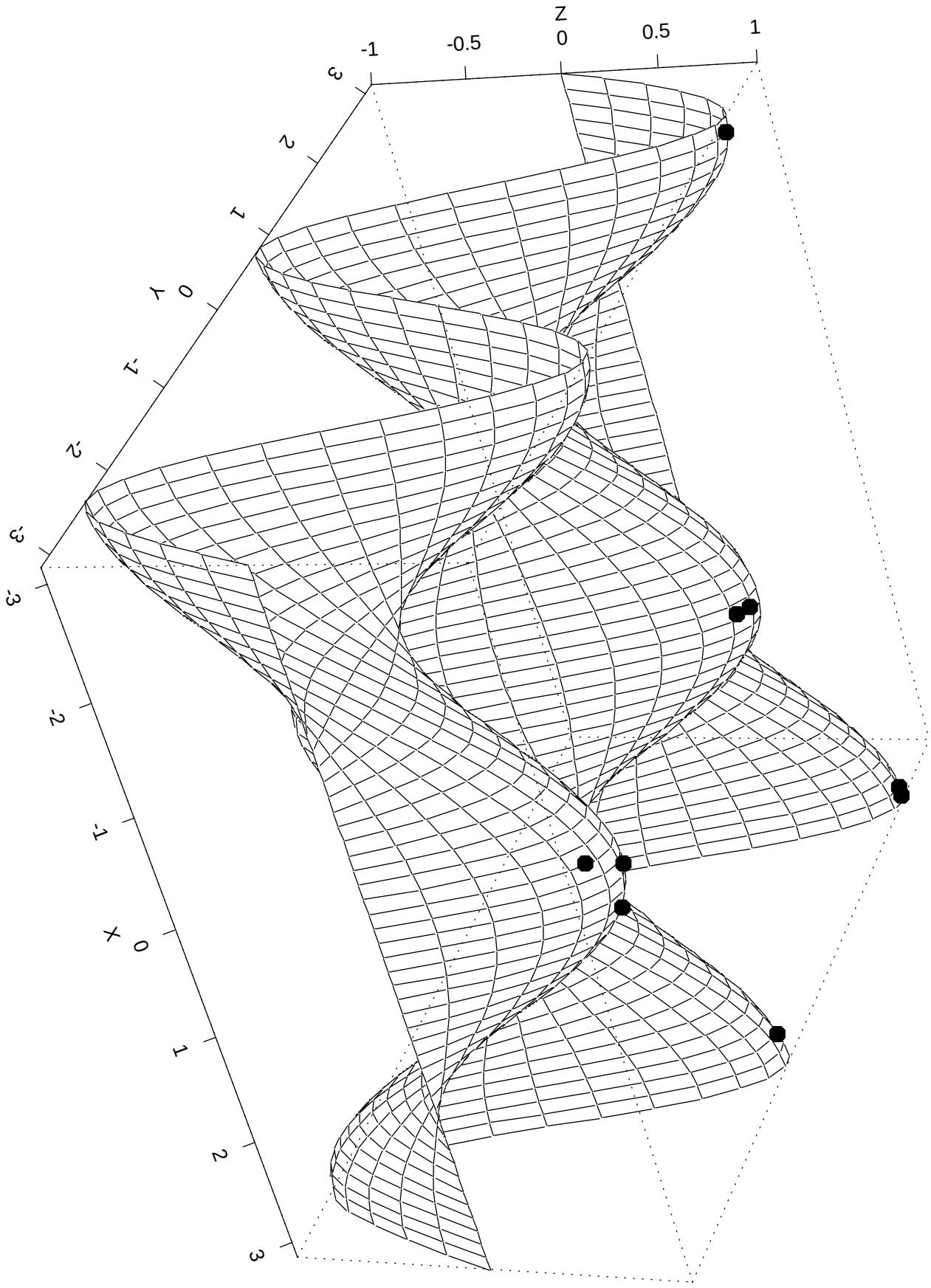
5. De Jong, K. A. (1975) *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral Thesis, Department of Computer and Communication Sciences. University of Michigan, Ann Arbor.

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

7. Goldberg, D. E., Deb, K., & Horn, J. (1992) Massive multimodality, deception, and genetic algorithms. *Proceedings of the Parallel Problem Solving from Nature Conference*, 37-46, North-Holland.

8. Mahfoud, S. W. (1992) Crowding and preselection revisited. *Proceedings of the Parallel Problem Solving from Nature Conference*, 27-36. Brussels, Belgium.

9. Perry, Z. (1984) *Experimental study of speciation in ecological niche theory using genetic algorithms*. Doctoral Thesis, University of Michigan, Ann Arbor.


Z
-1
-0.5
0
0.5
1
3
2
1
Y
0
-1
-2
-3
-3
-2
-1
X
0
1
2
3