Physica D 42 (1990) 228–234
North-Holland

# CO-EVOLVING PARASITES IMPROVE SIMULATED EVOLUTION AS AN OPTIMIZATION PROCEDURE

W. Daniel HILLIS
*Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142-1214, USA*

This paper shows an example of how simulated evolution can be applied to a practical optimization problem, and more specifically, how the addition of co-evolving parasites can improve the procedure by preventing the system from sticking at local maxima. Firstly an optimization procedure based on simulated evolution and its implementation on a parallel computer are described. Then an application of this system to the problem of generating minimal sorting networks is described. Finally it is shown how the introduction of a species of co-evolving parasites improves the efficiency and effectiveness of the procedure.

## 1. Introduction

The process of biological evolution by natural selection [5] can be viewed as a procedure for finding better solutions to some externally imposed problem of fitness. Given a set of solutions (the initial population of individuals), selection reduces that set according to fitness, so that solutions with higher fitness are over-represented. A new population of solutions is then generated based on variations (mutation) and combinations (recombination) of the reduced population. Sometimes the new population will contain better solutions than the original. When this sequence of evaluation, selection, and recombination is repeated many times, the set of solutions (the population) will generally evolve toward greater fitness.

A similar sequence of steps can be used to produce *simulated evolution* within a computer [3, 4, 12, 17–19]. In simulated evolution the set of solutions is represented by data structures on the computer and the procedures for selection, mutation, and recombination are implemented by algorithms that manipulated the data structures. Although the term "simulated evolution" deliberately suggests an analogy to biological evolution, it is understood that the real biological processes are far more complex than the simulation; simulated evolution represents only an idealization of certain aspects of a biological system. Such simulations are sometimes used as tools for understanding biological evolution [15], but this paper will concentrate on the use of simulated evolution for optimization; that is, as a practical method of generating better solutions to problems. Biological systems will serve as a source of metaphor and inspiration, but no attempt will be made to apply the lessons learned to biological phenomena.

As an optimization procedure, the goal of simulated evolution is very similar to that of other domain-independent search procedures such as *generate and test*, *gradient descent*, and *simulated annealing* [13, 16]. Like most such procedures, simulated evolution searches for a good solution, although not necessarily the optimal one. Whether or not it will find a good solution will depend on the distribution of solutions within the space.

These methods are all useful in searching solution spaces that are too large for exhaustive search. As in gradient descent and simulated annealing procedures, simulated evolution depends on information gathered in exploring some regions of the solution space to indicate which other regions of the space should be explored. How well this works obviously depends on the distribution of solutions in the space. The types of fitness spaces for which simulated evolution produces good results are not well understood, but one important type of space for which it works is a space that is independently a good domain for hill climbing in each dimension.

Another attractive property of simulated evolution is that it can be implemented very naturally on a massively parallel computer. During the selection step, for example, the fitness function can be evaluated for every member of the population simultaneously. The same is true for mutation, recombination, and a computation of statistics and graphics for monitoring the progress of the system. In the system described below, we routinely simulate the evolution of populations of a million individuals over tens of thousands of generations. Since these simulations take place on several generations per second, such experiments take only a few hours.

In these simulations, individuals are represented within the computer's memory as pairs of number strings that are analogous to the chromosome pairs of diploid organisms. The population evolves in discrete generations. At the beginning of each generation the computer begins by constructing a phenotype for each individual, using the set of number strings corresponding to an individual (the "genome") as a specification. The function used for the interpretation is dependent upon the experiment, but typically a fixed region within each of the chromosomes is used to determine each phenotypic trait of the individual. Discrepancies between the two bit strings of the pair are resolved according to some specified rule of dominance. This is similar to the diploid "genetic algorithms" studied by Smith and Goldberg [18].

To simulate selection, the phenotypes are scored according to a set of fitness criteria. When the system is being used to solve an optimization problem, the traits are interpreted as solution parameters and the individuals are scored according to the function being optimized. This score is then used to cull the population in a way that gives higher scoring individuals a greater chance of survival.

After the selection step, the surviving gene pool is used to produce the next generation by a process analogous to mating. Mating pairs are selected by either random mating from the entire population, some form of inbred mating, or assortive mating in which individuals with similar traits are more likely to mate. The pairs are used to produce genetic material for the next generation by a process analogous to sexual reproduction. First, each individual's diploid genome is used to produce a haploid by combining each pair of number strings into a single string by randomly choosing substrings from one or the other. At this point, randomized point mutations or transpositions may also be introduced. The two haploids from each mating pair are combined to produce the genetic specification for each individual in the next generation. Each mating pair is used to produce several siblings, according to a distribution normalized to ensure a constant total population size. The entire process is repeated for each generation, using the gene pool produced by one generation as a specification for the next.

The experiments that we have conducted have simulated populations ranging in size from 512 to $\sim 10^6$ individuals, with between 1 and 256 chromosomes per individual. Chromosome lengths have ranged from 10 to 128 bits per chromosome, mutation rates from 0 to 25% probability of mutation per bit per generation, and crossover frequencies ranged from 0 to an average of 4 per chromosome. Using a Connection Machine® [#1] with 65 536 processors, a typical experiment progresses at about 100 to 1000 generations per minute, depending on population size and on the complexity of the fitness function.

## 2. Sorting networks

As an example of how simulated evolution can be applied to a complex optimization problem, we consider the problem of finding minimal *sorting*

[#1]Connection Machine is a registered trademark of Thinking Machines Corporation.

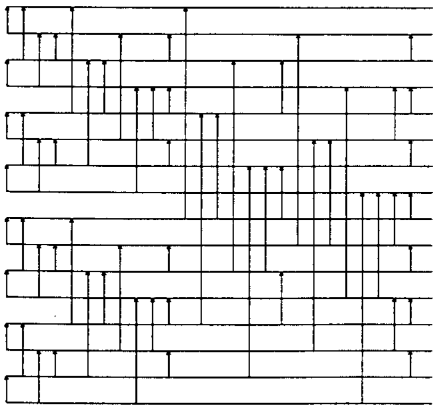
Fig. 1. Sorting network.

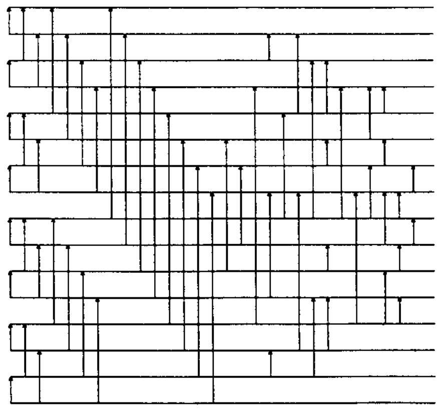
Fig. 2. Green's 60-comparison sorter.

*networks* for a given number of elements. A sorting network [14] is a sorting algorithm in which the sequence of comparisons and exchanges of data take place in a predetermined order. Finding good networks is a problem of considerable practical importance, since it bears directly on the construction of optimal sorting programs, switching circuits, and routing algorithms in interconnection networks. Because of this, the problem has been well studied, particularly for networks that sort numbers of elements that are exact powers of two.

Sorting networks are typically implemented as computer programs, but they have a convenient graphical representation, as shown in fig. 1. The drawing contains $n$ horizontal lines, in this case 16, corresponding to the $n$ elements to be sorted. The unsorted input is on the left, and the sorted output is on the right. A comparison–exchange of the $i$th and $j$th elements is indicated by an arrow from the $i$th to the $j$th line. Two specified elements are compared and they are exchanged if and only if the element at the head of the arrow is less than the element at the tail; the smallest element will always end up at the tail. The sorting network pattern shown in fig. 1 is a Batcher sort [1], which requires $n \log^2 n - 1$ exchanges to sort $n$ elements.

A useful property of sorting networks is that they are relatively easy to test. A sorting network that correctly sorts all sequences of 1 and 0 will correctly sort any sequence, so it is possible to test an $n$-input sorting network exhaustively with $2^n$ tests.

In this section we describe how simulated evolution is used to search for networks that require a small number of exchanges for a given number of inputs. In particular, the case $n = 16$ is of particular interest, and has a long history of successive surprises. In 1962, Bose and Nelson [2] showed a general method of sorting networks which required 65 exchanges for a network of 16 inputs. They conjectured that this was the best possible. In 1964, Batcher [1], and independently, Floyd and Knuth [6], discovered the network shown in fig. 1, which requires only 63 exchanges. Again, it was thought by many to be the best possible, but in 1969, Shapiro [14] discovered a network using only 62 exchanges. Later that year, Green [14] discovered a 60-comparison sorter, shown in fig. 2, which stands as the best known. These results are summarized in table 1. For a lively and more detailed account of these developments, the reader is referred to the book by Knuth [14, pp. 227–229].

There are two ways to cast the search for minimal sorting networks as an optimization problem. The first is to search the space of functional sorting networks for one of minimal length. The second is to search the space of short sequences of comparison/exchanges for ones that sort best. The difficulty with the first approach is that there is no obvious way of mutating a working sorting

Table 1
Summary of number of exchanges required for best known sorting networks with 16 inputs.

| | Best known networks | |
|---|---|---|
| 1962 | Bose and Nelson | 65 |
| 1964 | Batcher, Knuth | 63 |
| 1969 | Shapiro | 62 |
| 1969 | Green | 60 |
| | Networks found by simulated evolution | |
| without parasites | | 65 |
| with parasites | | 61 |

network into another one that is guaranteed to work, so almost all mutations and recombinations will create a network that is outside of the search space. It is much easier in the second approach to produce mutations and variations of a small program that stay within the space of small programs. Mutation can be implemented by changing the position of one of the exchanges, and recombination by splicing the first part of one sorting network with the last part of another. This is essentially the approach we have adopted.

One difficulty with this approach is that even if the solution is in the space of small networks, the easiest paths to the solution may not be. It may be easier, for example, to produce a short correct network by optimizing a slightly longer correct network than by fixing a bug in a short uncorrect network. For this reason, we have taken advantage of the diploid representation of a genotype to allow longer networks to be generated as intermediate solutions.

The genotype of each individual consists of 15 pairs of chromosomes, each consisting of 8 codons, representing the digits of 4 chromosome pairs. Each codon is a 4-bit number, representing an index into the elements, so the genotype of an individual is represented as 30 strings of 32 bits each. The phenotype of each individual (an instance of a sorting network) is represented as an ordered sequence of ordered pairs of integers. There is one pair for each exchange within the network. The elements of the pair indicate which elements are to be compared and optionally exchanged. Each individual has between 60 and 120 pairs in its phenotype, corresponding to sorting networks with 60 to 120 exchanges.

The phenotype is generated from the genotype by traversing the chromosomes of the genotype in fixed order, reading off the pairs to appear in the phenotype. If a pair of chromosomes is homozygous at a given position (if the same pair is specified in both chromosomes), then only a single pair is generated in the phenotype. If the site is heterozygous, then both pairs are generated. Thus the phenotype will contain between 60 and 120 exchanges, depending on the heterozygosity of the genotype. Sixty was chosen as the minimum size so that a completely homozygous genotype would produce a sorting network that matches the best known solution. Because most of the known minimal 16-input networks begin with the same pattern of 32 exchanges, the gene pool is initialized to be homozygous for these exchanges. The rest of the sites are initialized randomly.

Once a phenotype is produced, it is scored according to how well it sorts. One measure of ability to sort is the percentage of input cases for which the network produces the correct output. This measure is convenient for two reasons. First, it offers partial credit for partial solutions. Second, it can be conveniently approximated by trying out the network on a random sample of test cases. After scoring, the population is culled by truncation selection at the 50% level; only the best scoring half of the population is allowed to contribute to the gene pool of the next generation.

To implement recombination, the gamete pool is generated by crossover among pairs of chromosomes. For each chromosome pair in the surviving population, a crossover point is randomly and independently chosen, and a haploid gamete is produced by taking the codons before the crossover point from the first member of each chromosome pair, and the codons after the crossover point from the second member. Thus, there is exactly one crossover per chromosome pair per generation. Point mutations are then introduced in the gamete pool at a rate of one mutation per one thousand sites per generation.

The next stage is the selection of mates. One way to do this would be to choose pairs randomly, but our experience suggests that it is better to use a mating program with some type of spatial locality. This increases the average inbreeding coefficient and allows the population to divide into locally mating demes. The sorting networks evolve on a two-dimensional grid with torroidal boundary conditions. Mating pairs are chosen to be nearby in the grid. Specifically, the $x$ and $y$ displacement of an individual from its mate is a binomial approximation of a Gaussian distribution. Mating consists of the exchange of haploid gametes. After a pair mates, they are replaced by their offspring in the same spatial location, so the genetic material remains spatially local.

Simulations were performed using the procedure on populations of 65 536 individuals for up to 5000 generations. Typically, one solution, or a few equal scoring solutions, were discovered relatively early in the run. These solutions and their variants then spread until they accounted for most of the genetic material in the population. In cases where there was more than one equally good solution, each "species" dominated one area of the grid. The areas were separated by a boundary layer of non-viable crosses. Once these boundaries were established, the population would usually make no further progress. The successful networks tend to be short because the descendant of heterozygotes tended to be missing crucial exchanges (recessive lethals). The best sorting networks found by this procedure contained 65 exchanges.

## 3. The co-evolution of parasites

While the evolution of the sorting networks produced respectable results, it was evident on detailed examination of the runs that a great deal of computation was being wasted. There were two major sources of inefficiency. One was a classical problem of local optima: once the system found a reasonable solution, it was difficult to make progress without temporarily making things worse. The second problem was an inefficiency in the testing process. After the first few generations, most of the tests performed were sorted successfully by almost all viable networks, so they provided little information about differential fitness. Many of the tests were too "easy." Unfortunately, the discriminative value of a test depends on the solutions that initially evolve, and in the case where several solutions evolve, the value of a given test varies from one sub-population to another.

To overcome these two difficulties, various methods were implemented for accelerating progress by encouraging a wider diversity of solutions and limiting the number of redundant test cases. Three general methods were investigated: varying the test cases over time, varying the test cases spatially, and varying the test cases automatically by independent evolution. Because the third case has yielded the most interesting results, only it will be described in detail.

The co-evolution of test cases is analogous to the biological evolution of a host parasite, or of prey and predator. Hamilton has used both computer simulation and mathematical/biological arguments to show that such co-evolution can be a generator of genetic diversity [7–11]. The improved optimization procedure uses this idea to increase the efficiency of the search.

In the improved procedure, there are two independent gene pools, each evolving according to the selection/mutation/recombination sequence outlined above. One population, the "hosts", represents sorting networks, while the other population, the "parasites", represents test cases. (These two populations might also be considered as "prey" and "predator", since their evolution rates are comparable.) Both populations evolve on the same grid, and their interaction is through their fitness functions. The sorting networks are scored according to the test cases provided by the parasites at the same grid location. The parasites are scored according to how well they find flaws in sorting networks. Specifically, the phenotype of

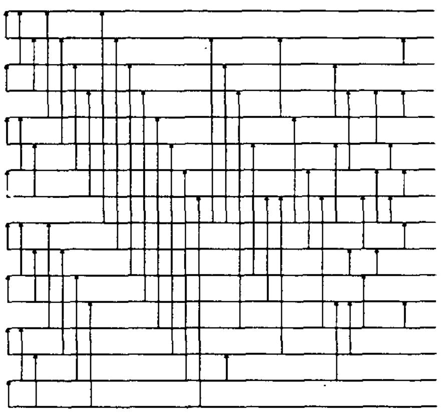

Fig. 3. 61 exchanges.

each parasite is a group of 10 to 20 test cases, and its score is the number of these tests that the corresponding sorting network fails to pass. The fitness functions of the host sorting networks and the parasitic sets of test patterns are complementary in the sense that a success of the sorting network represents a failure of the test pattern and vice versa.

The benefits of allowing the test cases to co-evolve are twofold. First, it helps prevent large portions of the population from becoming stuck in local optima. As soon as a large but imperfect sub-population evolves, it becomes an attractive target toward which the parasitic test cases are likely to evolve. The co-evolving test cases implement a frequency selective fitness function for the sorting networks that discourages large numbers of individuals from adopting the same non-optimal strategy. Successive waves of epidemic and immunity keep the population in a constant state of flux. While systems with a fixed fitness criteria tended to get stuck in a few non-optimal states after a few hundred generations, runs with co-evolving test cases showed no such tendency, even after tens of thousands of generations.

The second advantage of co-evolving the parasites is that testing becomes more efficient. Since only test-case sets that show up weaknesses are widely represented in the population, it is sufficient to apply only a few tests to an individual each generation. Thus, the computation time per generation is significantly less. These two factors taken together make it both more practical and more productive to allow the system to run for larger numbers of generations.

The runs with co-evolving parasites produced consistently better and faster results than those without. Fig. 3 shows the best result found to date, which requires 61 exchange elements. This is an improvement over Batcher's and Shapiro's solutions, and over the results of the simulation without parasites. It is still not the optimum network, since it requires one more sorting exchange than the construction of Green.

These preliminary results are encouraging. They demonstrate that simulated evolution of co-evolving parasites is a useful procedure for finding good solutions to a complex optimization problem. We are currently applying similar techniques to other applications in an attempt to understand the range of applicability. It is ironic, but perhaps not surprising, that our attempts to improve simulated evolution as an optimization procedure continue to take us closer to real biological systems.

## Acknowledgements

The author would like to thank Chuck Taylor, Bill Hamilton, Stuart Kaufmann, Steve Smith, and the reviewers for helpful discussions and comments in the preparation of this paper.

## References

[1] K.E. Batcher, A new internal sorting method, Goodyear Aerospace Report GER-11759 (1964).

[2] R.C. Bose and R.J. Nelson, A sorting problem, J. Assoc. Computing Machinery 9 (1962) 282–296.

[3] D.G. Bounds, New optimization methods from physics and biology, Nature 329 (1987) 215–219.

[4] H.J. Bremermann, Optimization through evolution and recombination, in: Self-Organizing Systems, eds. M.C.

Yovits, G.D. Goldstein and G.T. Jacobi (Spartan, Washington, DC, 1962) pp. 93-106.

[5] C. Darwin, The Origin of the Species by Means of Natural Selection; or, The Preservation of Favoured Races in the Struggle for Life (Murray, London, 1859).

[6] R.W. Floyd and D.E. Knuth, Improved constructions for the Bose–Nelson sorting problem, Notices Am. Math. Soc. 14 (1967) 283.

[7] W.D. Hamilton and J. Seger, Parasites and sex, in The Evolution of Sex, eds. R.E. Micmod and B.R. Levin (Sinauer, Sunderland, MA, 1988) ch. 11.

[8] W.D. Hamilton, Pathogens as causes of genetic diversity in their host populations, in: Population Biology of Infectious Diseases, eds. R.M. Anderson and R.M. May, Dahlem Konferenzen (Springer, Berlin, 1982) pp. 269-296.

[9] W.D. Hamilton, Sex versus non-sex versus parasite, OIKOS (Copenhagen) 35 (1980) 282-290.

[10] W.D. Hamilton, P. Henderson and N. Moran, Fluctuation of environment and coevolved antagonist polymorphism as factors in the maintenance of sex, in: Natural Selection and Social Behavior: Recent Research and Theory, eds. R.D. Alexander and D.W. Tinkle (Chiron Press, New York, 1980) ch. 22.

[11] W.D. Hamilton, Gamblers since life began: barnacles, aphids, elms, Quart. Rev. Biol. 50 (2) (1975) 175–180.

[12] J.H. Holland, Adaption in Natural and Artificial Systems (University of Michigan, Ann Arbor, 1975).

[13] S. Kirkpatrick, C. Gelatt Jr. and M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[14] D. Knuth, Sorting and Searching, Vol. 3. The Art of Computer Programming (Addison–Wesley, New York, 1973).

[15] R. Lewontin and I. Franklin, Is the gene the unit of selection?, Genetics 65 (1970) 707–734.

[16] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller. J. Chem. Phys. 21 (1953) 1807.

[17] I. Rechenberg, Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution (Frommann–Holzboog, Stuttgart, 1973).

[18] R. Smith and D. Goldberg, Nonstationary function optimization using genetic algorithms with dominance and diploidy, in: Genetic Algorithms and Their Applications, Proceedings of the Second International Conference on Genetic Algorithms, July 1987.

[19] Q. Wang, Optimization by simulating molecular evolution, Biol. Cybern. 57 (1987) 95–101.