speedup to be less than 1. In that case, the anomaly is called a *detrimental anomaly*. Lai and Sahni (1984) and Li and Wah (1986) have studied in detail how these anomalies might occur. See also Wah, Li, and Yu (1985).

13.3 GENETIC ALGORITHMS

13.3.1 Evolution and Genetic Algorithms

Genetic algorithms have their theoretical roots in biology and are an attempt to mimic the process of natural evolution in a population of individuals. While the exact mechanisms behind natural evolution are not very well known, some aspects have been studied in significant depth. One of the principal research areas in biology for some time now has been the study of chromosomes: the information carriers containing the characteristics of a living being. Although there is some uncertainty surrounding the way in which chromosomes dictate the properties of living organisms, there is little doubt that they in fact do uniquely determine them.

Evolution is a process that works at the chromosome level, through the reproductive process, rather than on the living individuals themselves. When individuals in a population reproduce, portions of the genetic information of each parent (portions of the parents' chromosomes) are combined to generate the chromosomes of their offspring. In this way, the offspring carry genetic-makeup information that is a blend of their parents' information, and exhibit a corresponding blend of the parental characteristics. Combining chromosomes from the parents to produce offspring is the predominant mechanism by which chromosome patterns change. This inheritance of some portion of the characteristics from each parent is termed *crossover*.

In addition, an occasional random change may occur in the chromosome pattern of a particular individual: an effect termed *mutation*. Mutations may lead to individuals whose chromosome patterns are significantly different from either parent. A mutation may enhance survivability or impair it, compared to the individual's parents. The most significant aspect of mutation is that it causes an abrupt change in the individual's viability that is not directly related to that of the parents. Since there appear to be many more ways to degrade an individual's survivability than to improve it, massively occurring mutations tend to degrade the viability of the population over time. Rarely occurring mutations will occasionally produce a favorable change, yet will not overwhelm the population with unfavorable changes.

As with all changes to an organism's chromosomes, there will be a change in the way the organism interacts with its environment. On occasion these changes improve the viability of the adapted organism relative to its ancestors, leading to an increased likelihood of its surviving and passing on some of its chromosome information to its offspring. Naturally, there will be occasions as well in which the changes reduce the organism's viability and thereby decrease its chances of passing that information to its offspring. Mutation is generally believed to play a significant, although infrequent, role in natural evolution as compared to crossover.

When the mutation rate is relatively low, the unfit individuals tend to die out before passing their traits on to their offspring. Conversely, the more fit individuals generally succeed in passing their traits to future generations. However, when the mutation rate is relatively large, large numbers of individuals whose characteristics are randomly different from the characteristics of the preceding generation are introduced. This leads to degeneration of the population in nature, and to instability, or nonconvergence, in the computer implementation of genetic algorithms.

Over time, changes in chromosomes produce variants that may differ significantly from the members of the original organism strain. In general, changes that enhance the viability of the strain in some way tend to predominate over time. For example, in species of birds in which the female provides the primary care for the eggs in a nest, the feather coloration of the females is generally subdued in comparison to that of the males. In this manner, the females blend into their surroundings rather than call attention to themselves as the males do. The lifespan of individuals who blend into their surroundings while sitting motionless on a nest for long periods tends to be greater than the lifespan of those who call a predator's attention to themselves. As such, females tend to pass their subdued-coloration chromosomes to more offspring. Eventually, the population becomes weighted toward those characteristics that tend to be associated with increased lifespan.

The theory proposed by Charles Darwin in 1859 in his classic work on evolution, the theory of natural selection, is more commonly known as "survival of the fittest." Just as in the preceding bird example, in any population of a species the "stronger" or "more fit" individuals are more likely to survive and subsequently to pass on to their offspring those traits that led to their survival. It is certainly possible for an extremely fit individual to fail to survive long enough to reproduce! An example would be the individual who just happens to be in "the wrong place at the wrong time" and is prematurely turned into a food source for another species. As the pundits have noted, the race does not always go to the fastest, or the fight to the strongest, but on average it sure pays to bet on those outcomes.

The repeated application of the principles of crossover and mutation, each time followed by the production of a "next generation" based on the relative fitness of the reproducing members, is termed a genetic algorithm. Just as in nature, this approach toward optimization of a "population of solutions" tends over time to produce good solutions. The next sections describe in detail how genetic algorithms may be implemented.

Genetic algorithms are computational approaches to problem solving that are modeled after the biological process of evolution. The algorithm first creates an initial population of solutions (*individuals*). These solutions are then evaluated, using an application-specific criterion of *fitness*, to characterize them from "most fit" to "least fit." A subset of the population is selected, using criteria that tend to favor the more fit individuals. This subset is then used to produce a new generation of offspring. Finally, a small number of individuals in this next generation are subjected to random *mutations*.

The processes of selection, crossover, and mutation are then repeated, producing a number of generations. The underlying presumption is that, as in nature, individuals in a population will tend to evolve and improve in fitness so long as the more fit individuals in a given generation are used to produce the next generation. While genetic algorithms are not guaranteed to find the optimum solution, they are generally excellent at quickly finding solutions that are pretty good.

13.3.2 Sequential Genetic Algorithms

The following is a pseudocode outline of the components of a sequential genetic algorithm:

```
generation_num = 0;
initialize Population(generation_num);
evaluate Population(generation_num);
set termination_condition to False;
while (not termination_condition) {
    generation_num++;
    select Parents(generation_num) from Population(generation_num - 1);
    apply crossover to Parents(generation_num) to get
        Offspring(generation_num);
    apply mutation to Offspring(generation_num) to get
        Population(generation_num);
    evaluate Population(generation_num) and update termination_condition;
}.
```

Several issues have been somewhat glossed over to this point in the discussion. Among them are

- How one represents an individual or potential solution in terms of "chromosomes"
- How one produces the initial population of potential solutions
- How one evaluates each potential solution to determine its fitness
- How one determines the condition(s) under which the repetition terminates
- How one selects individuals to become "parents" of the next generation, and the actual production of next-generation individuals

Once these have been explored, the issues involving parallelizing the sequential approach will be considered.

13.3.3 Initial Population

To demonstrate the first steps in the genetic algorithm method, let us consider the simple problem of determining the maximum of the function

$$f(x, y, z) = -x^2 + 1{,}000{,}000x - y^2 - 40{,}000y - z^2$$

over an integer domain of $-1{,}000{,}000$ to $+1{,}000{,}000$ for each variable. (The variables, x, y, and z are restricted to integer values for additional simplicity.)

It so happens that there is a direct solution to this problem that facilitates verification of the results obtained by other techniques. In practice, of course, since they do not guarantee the optimum solution, one would only apply search and optimization techniques to problems for which a simple closed-form solution did not exist. Using simple algebra, f can be factored into a form in which the maximum may be seen by inspection

$$f(x, y, z) = 2504 \times 10^8 - (x - 500{,}000)^2 - (y + 20{,}000)^2 - z^2$$

In this form, it is apparent that the maximum of $f(x, y, z)$, 250,400,000,000, occurs when x is 500,000, y is $-20{,}000$, and z is 0, since those coordinate values make the squared terms zero and any other coordinate values reduce f by subtracting something from the 2504×10^8 term. But let us suppose that such a direct solution is unavailable. (As indicated, we use this function as our test case so that the computer solution can be readily checked against the actual solution; any function could be used.)

Due to the computational size of the problem, the "immediately obvious" exhaustive search approach of simply calculating the function value corresponding to every possible coordinate is impractical. There are $(2{,}000{,}001)^3$ coordinates to be evaluated, and even if the function value corresponding to each could be computed in 100 ns (highly optimistic for any reasonable evaluation function), it would take more than 200,000,000 hours on a single processor. Even if we were able to speed up the process by a factor of 10,000 through parallel implementation of this algorithm, we still would have to wait more than two years for a solution.

Data Representation. The first step in determining the initial population of potential solutions is to choose a suitable data representation for an individual. In early genetic algorithm work, the potential solutions were simply strings of 1's and 0's. More recent work has extended the representations to include floating point, Gray code, and integer strings (Michalewicz, 1996). For simplicity, we will use binary strings.

The ith potential solution will consist of a three-tuple (x_i, y_i, z_i), in which each component of the three-tuple has one of the 2,000,001 possible integers in the interval $-1{,}000{,}000 \le \text{component\_value} \le +1{,}000{,}000$. Since

$$2^{20} < 2{,}000{,}001 \le 2^{21}$$

21 bits are needed to represent each of the three components. A single potential solution (i.e., a chromosome, or individual) will simply be the 63 bits consisting of the concatenation of the (x_i, y_i, z_i) representations.

Suppose $x = +262{,}408_{10}$, $y = +16{,}544_{10}$, and $z = -1{,}032_{10}$. Using the "sign plus magnitude" representation, the binary representation for each of the numbers is

$$x = 001000000000100001000$$
$$y = 000000100000010100000$$
$$z = 100000000010000001000$$

where the leftmost bit of each number is the sign (0 denoting positive values, 1 denoting negative values) and the remaining bits represent the magnitude. Note that this choice of representations ensures that the magnitudes must fall between $-1{,}048{,}575$ and $+1{,}048{,}575$, or $\pm 2^{20}$. Concatenating into a binary string, we get the representation of a potential solution

$$001000000000100001000000000100000010100000100000000010000001000$$

Now that we have determined the length of a potential solution, 63 bits, we may easily introduce an initial population of them using a pseudorandom generator.

Evaluation. Next, we use the values of x, y, and z in our function to assess the fitness of this individual:

$$f(x, y, z) = -x^2 + 1{,}000{,}000x - y^2 - 40{,}000y - z^2$$

$$\begin{aligned} f(262408, 16544, -1{,}032) &= -(262{,}408)^2 + 1{,}000{,}000 \times 262{,}408 - (16{,}544)^2 - 40{,}000 \\ &\quad \times 16{,}544 - (-1{,}032)^2 \\ &= -68{,}857{,}958{,}464 + 262{,}408{,}000{,}000 - 273{,}703{,}936 - 661{,}760{,}000 - 1{,}065{,}024 \\ &= 192{,}613{,}512{,}576 \end{aligned}$$

This evaluation process would be repeated for every individual in the population, calculating the function value, or fitness, corresponding to each. These individuals are then ranked according to how well they did at maximizing the function. That is, individuals whose fitness, as evaluated by this function, is greater than 192,613,512,576 would be ranked more fit than this individual, while those evaluating to a lower value would be ranked as less fit.

Constraints. Naturally, since our variable domain is restricted to ±1,000,000, any individual whose x, y, or z values fall outside that interval is considered to be of such low fitness that there is no reason even to compute its function value in anticipation of its possibly becoming a parent. That is, its equivalent in nature would be that of a stillborn individual. There are other ways to handle restricted domains on individuals. The defect that caused the individual to fall outside the ±1,000,000 interval could be "surgically repaired" (changing one or more bits to correct the defect). Alternatively, the generated individual can be mapped into the restricted domain prior to fitness evaluation with some simple scaling of each coordinate:

$$\text{Scaled coordinate} = -1{,}000{,}000 + \frac{\text{coordinate}}{2^{21}} \times (2{,}000{,}000)$$

Solution constraints on individuals as applied to genetic algorithms are discussed further in Michalewicz (1996).

Number of Individuals. Finally there is the issue of how many individuals should be placed in the initial population. In general, a small number of individuals decreases the likelihood of obtaining a good solution in a reasonable number of generations, while a large number increases the computation required for each generation. It is not uncommon for an initial population to be made up of 20 to 1000 pseudorandom potential solutions. For this example, that would mean the production of 20 to 1000 pseudorandomly generated binary strings, each of which is 63 bits long.

13.3.4 Selection Process

Again, modeling after nature, it should be possible for any individual to be selected to become a parent of an offspring in the next generation. However, as in nature, a more fit individual is presumed to have a somewhat greater chance of being selected than one that is less fit. This bias toward the more fit individuals is termed *selective pressure*.

At first glance it might appear that a high degree of selective pressure (e.g., choosing only the most fit portion of the population to produce offspring for the next generation) would be highly desirable. However, for problems in which there may be local optima, this may result in the process converging rapidly to a local optimum and totally missing the global optimum. The opposite situation, too little selective pressure, results in very slow convergence or even nonconvergence; neither situation is desirable.

Based on its having demonstrated relatively satisfactory performance in many applications, one of the leading candidates for use in applying selective pressure is *tournament selection*.

Tournament Selection. In this approach each individual is equally likely to be chosen at random from the general population and entered into a tournament. A set of k such individuals is chosen for each tournament, and the fitness of each is evaluated. The most fit individual from this set of k individuals is deemed to be the overall tournament winner and will become a parent to the next generation of individuals. In all, n such tournaments are conducted to determine the n individuals that will be used to produce the next generation, which keeps the population size the same as the preceding population size.

Clearly, when k is 1 there is no selective pressure at all; each member of the population is equally likely to be used to produce the next generation. When k is large, the amount of selective pressure is likewise large, since only the individual determined to be the most fit out of k randomly chosen individuals will be selected to become a parent. More typically, a value of 2 is used for k, reminiscent of a medieval joust (or tournament) by two knights.

13.3.5 Offspring Production

Once the more fit individuals have been selected from the present generation through a series of tournaments, their chromosomes must be combined to determine the composition of the individuals who will make up the population of the next generation. The process of combining portions of the chromosomes from each parent to produce the chromosomes of the children is crossover. While there are several common crossover forms, our discussion will be limited to single-point crossover.

Single-Point Crossover. In single-point crossover, a randomly located cut is made at the pth bit in the m-bit chromosome patterns of each parent, A and B. This cut divides each parent's pattern into two parts. Child number 1 has its pattern made from the first p bits of parent A's pattern, followed by the last $m - p$ bits of parent B's pattern. Child number 2 has its pattern made from the first p bits of parent B, followed by the last $m - p$ bits of parent A's pattern. As the term implies, portions of the parents' chromosome patterns *cross over* in the process of forming the children. This is illustrated in Figure 13.2. For example, consider two individuals that have been selected to be parents. Each has 63-bit chromosome patterns

A = 001100101000000100001 010010100000010101010 000111111100000011000

and

B = 000001000000101000001 101000001010101010101 000110001010101010000.

If the cut is made after the fifteenth bit, the two portions of the parents' chromosomes are

Parent A (first 15 bits): 001100101000000

Parent A (last 48 bits): 100001010010100000010101010000111111100000011000

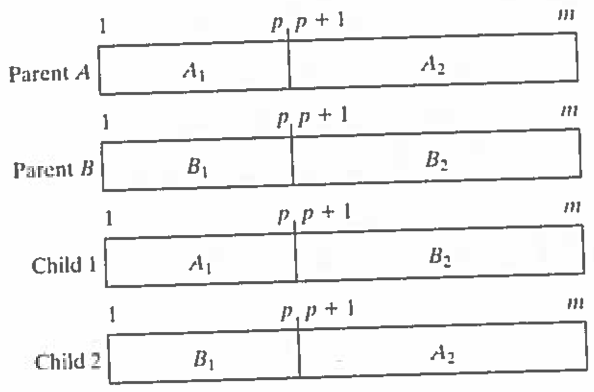

Figure 13.2 Single-point crossover.

Parent B (first 15 bits): 000001000000101
Parent B (last 48 bits): 000001101000001010101010101000110001010101010000

Applying crossover, a pair of children are created:

Child 1: 001100101000000 000001101000001010101010101000110001010101010000

and

Child 2: 000001000000101 100001010010100000010101010000111111100000011000

The children's *genes*, the individual bits of their chromosomes, are a blend of the genes of their parents. Due to the selection process having a bias toward choosing the more fit individuals from the current generation, the children are characterized by a blend of more fit parents' characteristics. As in nature, crossover slowly tends to force the population toward stronger (i.e., more fit) individuals. Since this is a gradual, relatively predictable change, crossover is usually encouraged in genetic algorithms and typically occurs at each generation.

While we have discussed only single-point crossover in this example, it should be noted that researchers have investigated other alternatives. Some of these include multipoint crossover, in which several cuts are made and smaller portions of the chromosomes are intermingled, and uniform crossover, in which each bit, or gene, is randomly selected from either parent. The use of gene sharing from a pool of parents in which each gene of an offspring is selected from a parent randomly chosen from the pool has also been investigated.

Mutation. Unlike crossover, mutation occurs at the isolated gene level within a chromosome. This can be thought of as similar to the effects that disease, radiation, or the ingestion of various controlled substances might have on a living organism. Mutation tends to bring about major unpredictable changes in the fitness of an individual.

In the initial example of maximizing a function, a single-bit change in the leftmost position of the 21-bit pattern of x, y, or z, would induce a sign change in that component of a coordinate, but the magnitude would remain the same. A single-bit change in the second position from the left (the position weighted by 2^{19}) would cause a change in magnitude of

524,288 for x, y, or z. However, a single-bit change in the rightmost position would cause a change in magnitude of only 1.

Since the effect of a single-bit mutation on an individual's x, y, or z component may range all the way from minimal (± 1) to extreme (±1,000,000), depending upon which bit is changed, it is clear that mutation can be an extremely powerful factor in influencing an individual's fitness. When the probability of mutation is high, the chances of a relatively fit individual changing characteristics (and thereby being dramatically degraded in fitness) are likewise high. In the extreme, high rates of mutation may result in the nonconvergence of the genetic algorithm.

Since the changes in an individuals's fitness due to mutation may be very dramatic and much less predictable than the effects of crossover, the chances of mutations occurring are generally kept small in a genetic algorithm. This allows mutated individuals to be introduced into the population, but at a rate slow enough to prevent them from continually shifting the population away from convergence on an optimum solution. Through selection and crossover, the effects of mutations are gradually incorporated into the population as a whole. When it is beneficial (i.e., the mutation improves the fitness of an individual), the mutated individual's likelihood of being selected to help produce the next generation of individuals is increased. This automatically results in the propagation of beneficial mutations and the dying out of the nonbeneficial mutations.

13.3.6 Variations

The preceding sections by no means exhaust the variations that have been explored in producing members of the next generation. Instead of producing the next generation solely from crossover and mutation on the current generation, one might

- Carry over a few of the most fit individuals from this generation to be a part of the next generation
- Randomly create a few new individuals in each generation rather than only generating randomized individuals at the initiation of the algorithm
- Allow the population size to vary from one generation to the next

13.3.7 Termination Conditions

Several strategies have been employed in terminating genetic algorithms, ranging from a simple count of the number of generations to attempts to determine how close the solution is to convergence. In the simplest case, the production of successive generations is carried out s times. While this is an easy enough termination condition to implement, it suffers from two obvious deficiencies. In one extreme the population may have converged already to a solution in far fewer than s generations, while in the other it may still have a long way to go before converging satisfactorily.

Since the optimal solution is unknown a priori, one cannot simply measure the difference between the current best solution and the optimum to decide whether to terminate. On the other hand, one can borrow a technique from the field of numerical analysis in which the results of successive iterations are examined, and a decision to terminate is then based on the degree of improvement between successive generations. Just as various numerical algorithms may exhibit oscillation around a solution rather than further convergence, so may genetic algorithms. Since such oscillation has been experimentally confirmed both in the situation where the difference in probability of selection of individuals is nearly the same (minimal selective pressure) and in the high-mutation-rate situation, some care must be taken in relying solely on this form of termination.

Yet another termination condition may be based on the degree of similarity of the individuals within the population. Since their similarity increases as the population of individual-solution candidates converges toward an optimal solution, measuring the diversity in the population serves to measure convergence as well. It is not uncommon for a genetic algorithm to incorporate multiple termination conditions.

13.3.8 Parallel Genetic Algorithms

Conceptually, the problem of adapting genetic algorithms to multiple processors is straightforward. Two approaches immediately come to mind:

1. Let each processor operate independently on an isolated subpopulation of the individuals, periodically sharing its best individuals with the other processors through *migration*, or
2. Let each processor do a portion of each step of the algorithm — selection, crossover, and mutation — on the common population.

Isolated Subpopulations. In the first approach each processor independently generates its own initial subpopulation of individuals. Each processor then carries out k generations of individuals:

- Handling the evaluation of fitness
- Selecting the individuals from its subpopulation to be used to produce its next generation
- Performing the crossover and mutation computations on its subpopulations.

After k such generations ($k \geq 1$), the processors share their best individuals with the other processors.

Migration Operator. When migration is incorporated, changes in a population come not only from inheriting portions of one's parents' genes, albeit with occasional random mutations, but also from the introduction of new species into the population. In nature, this movement between subpopulations is often a survival response, although humankind has (sometimes unwittingly) accelerated the process by providing a transportation mechanism. A few examples of transported species include the rabbits introduced into Australia and the zebra mussels introduced into the St. Lawrence River system and the Great Lakes region of North America. In genetic algorithms, the migration operator is responsible for several tasks needed to implement the exchange of individuals between subpopulations. Such tasks include

- Selecting the emigrants
- Sending the emigrants
- Receiving the immigrants
- Integrating the immigrants

Sending and receiving selected individuals can be done relatively easily in a message-passing parallel environment. It is the selecting and integrating of the individuals that provides interesting results. Selecting the best individuals to migrate from each subpopulation, and integrating the received individuals into the population in place of the worst individuals, will cause a faster convergence. Therefore, it would seem likely that by passing copies of a number of the more fit individuals from one subpopulation to another subpopulation, replacing the worst individuals would achieve convergence even faster. However, selecting and integrating in such a manner eventually places too much selective pressure on the population and might hinder the results produced. Once again, selective pressure should not be so great that the results progress toward a local optimum instead of the global optimum.

Introducing migration also introduces a communication overhead. As with other parallel computations using message passing, when the communication time begins to dominate the total computation time, the performance of the parallel algorithm suffers. Therefore, both the frequency and the volume of information communicated between slave processes should be considered.

Migration Models. The most popular approaches for modeling migration are

- The island model
- The stepping-stone model

In the *island model*, individuals are allowed to be sent to any other subpopulation, as illustrated in Figure 13.3. There are no restrictions on where an individual may migrate. In the *stepping-stone model*, migration is limited by allowing emigrants to move only to neighboring subpopulations. This concept is illustrated in Figure 13.4. The stepping-stone model reduces communication overhead by limiting the number of destinations to which emigrants may travel, and thereby limiting the number of messages. The island model allows more freedom, and in some ways represents a better model of nature. However, there is significantly more communication overhead and delay when implementing such a model.

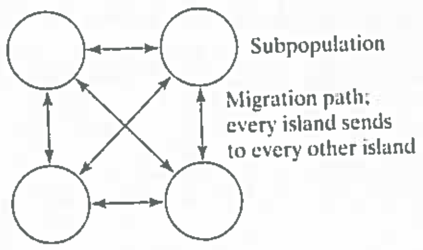

Figure 13.3 Island model.

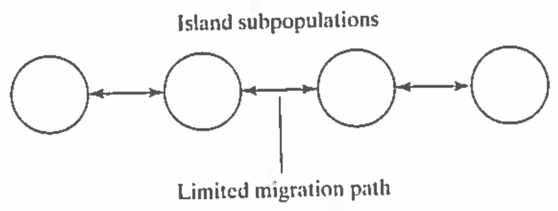

Figure 13.4 Stepping-stone model.

Parallel Implementation. The following pseudocode describes the implementation of the slave process:

```
set generation_num to 0;
initialize Population(generation_num);
evaluate fitness of Population(generation_num);
while (not termination_condition) {
   generation_num++;
   select Parents(generation_num) from
      Population(generation_num - 1);
   apply crossover to Parents(generation_num) to produce
      Offspring(generation_num);
   apply mutation to Offspring(generation_num) to get
      Population(generation_num);
   apply migration to Population(generation_num);
   evaluate Population(generation_num);
}
```

Each slave program executes the preceding pseudocode to produce results that are returned to the master program. The master program then performs some analysis before producing the final results.

The island and stepping-stone models both have advantages and drawbacks. The obvious advantage is that, except for the communication of best individuals, which occurs only once in every k generations, both are embarrassingly parallel in nature. That is, by using p processors, each operating independently on (population_size/p) subpopulations of individuals, one could potentially speed up the computation of the k generations by a factor of p.

Such isolated parallelism has even been used to attempt to explain the (apparently) missing links in the fossil record. Cohoon et al. (1987) proposed an implementation based on the theory of "punctuated equilibria," which provided a foundation for the theory that new species are likely to form quickly in subpopulations following a change in the environment. These changes might occur, for example, as a result of migration introducing new genetic material into the population. Cohoon et al. observed that when a substantial number of individuals migrated between isolated subpopulations, new solutions were found shortly after the migration occurred, just as the fossil records seem to indicate that major developments occurred in nature over relatively short periods.

Since the overall parallelism is interrupted every k generations so that information about the most fit individuals may be communicated, somewhat less than a p-fold speedup is possible under this approach. Whether information about the most fit individuals is communicated to all the other processors or only to ones deemed adjacent will affect the actual speedup. The interprocessor communication architecture also may be a factor.

Under the stepping-stone model, even the interprocessor communication that occurs after the kth generation is reduced by restricting communication to a processor's nearest neighbors, rather than permitting each processor to communicate with every other processor.

As computationally advantageous as the embarrassingly parallel approach may be, it suffers from a flaw: the relatively isolated nature of the population subsets. The isolation of the subsets increases the possibility that each will converge to its own local optimum rather than to a global optimum and perhaps completely miss the global solution. In addition, the reduced number of individuals making up each population subset increases the number of generations needed to achieve convergence.

As an aside, the isolation/local-optima phenomenon in genetic algorithms is believed to have occurred in nature as well! Geologists studying plate tectonics (the large sections of the earth's crust that move around, over, and into each other) have produced computer projections of past plate movements. One of the plates that has long been isolated from contact with others makes up the bulk of the Australian continent. Biologists have long noted the number of species unique to that region, including wingless birds and large hopping mammals. Evolution, occurring in parallel in isolated regions, does not necessarily result in similar solutions in each of the regions! Rather, differing local optima may well be the result.

Of course, the obvious mechanism to reduce the effective isolation, increased interprocessor communication, introduces its own disadvantage. An increase in the sequentially occurring (communication) portion of the algorithm and a decrease in the effective speedup over the case using multiple, but isolated, processors is the result.

Parallelizing a Common Population. As an alternative to this isolated subpopulation parallelism, one could simply implement the genetic operators in parallel on the global population. For example, if there were a way to provide information on the individuals to every processor, then the processors could perform selection, crossover, and mutation in parallel.

Clearly, if we utilize tournament selection, the processors may operate on independent pairs of individuals. Similarly, each processor could work independently on a subset of the selected individuals to perform the crossover and mutation operations and evaluate the resulting fitnesses. Unfortunately, while this effectively parallelizes the genetic operations, it may well do little to speed up the overall computation! Much depends upon the time needed to compute the various operations (selection, crossover, and mutation), compared to the time to distribute the population information to all the processors plus the time to communicate the results of the selection process. In many cases genetic operators require little intensive computation, and the resulting overall process speedup is effectively communication limited.

Shared Memory Systems. If the individuals were stored in shared memory, each processor has access to the population and may read/modify individuals in the process of carrying out the genetic algorithm. By assigning individuals to specific processors, it may be possible to minimize the memory-access contention that would otherwise require synchronization, thereby avoiding most of the serializing effects. Aside from synchronization between generations, which would introduce some serialization, each processor is free to carry out its computations on its local subpopulation in parallel with the others. Since communication between processors occurs at essentially the speed of accessing memory, one would expect the solution time of a genetic algorithm on such a system to be reduced compared to that of a distributed processor system.

Distributed Processor Systems. If all the workstations in a distributed processor system lie on the same subnet, then their interprocessor communications will be competing for the network resources. The simple task of communicating each processor's most fit individuals to the others will involve contention for network bandwidth and result in serializing a portion of the genetic algorithm. Minimizing the serial portions through the use of isolated subpopulations has historically been the approach of choice to maximize the parallelism.

On the other hand, it may not be necessary to minimize the subpopulation intercommunication if it is feasible to segregate the processors into a number of subnets. Doing this makes each subnet capable of carrying on communication among its processors independent of the other subnets. Effectively, this parallelizes (at the subnet level) at least a portion of the interprocessor communication in addition to parallelizing (at the individual processor level) the computation required by the genetic algorithm's operators.

Yet another mechanism for achieving at least partial parallelism in communication is to interconnect the processors in a two- or three-dimensional ring, or torus. This limits the communication to what occurs between processors deemed adjacent by virtue of their interconnections and corresponds to the stepping-stone model of isolated subpopulations. By limiting the interprocessor communication to the migration of individuals between adjacent subpopulations, while permitting each processor to work independently on its subpopulation in parallel with the others, much of the serializing effect of communication is eliminated. Marin, Trelles-Salazar, and Sandoval (1994) showed that near-linear speedup could be attained with a half dozen processors and claimed that it can scale up to larger workstation networks as well.

13.4 SUCCESSIVE REFINEMENT

There are many alternative searching algorithms that could have been employed to find a good solution to our optimization problem. One of the most intuitive applies successive refinement to the search grid. Initially, a grid spacing is chosen that permits a brute-force examination of the search volume in very little time. In the context of the problem just looked at, the grid size might have started with an examination of every ten-thousandth point. Since the volume spanned $-1{,}000{,}000$ to $+1{,}000{,}000$ on each edge, a search increment of 10,000 would mean that we only evaluate about $200 \times 200 \times 200$ points, or perform a little under 10^7 function evaluations. If the best K points are retained, and a cube centered on each is evaluated using a finer grid spacing, say an increment of size 100, it will require about $K \times 10^6$ additional function evaluations.

Again, retaining the K best points in each of the K subvolumes and exhaustively searching a grid of size 100^3 around each will result in about $K^2 \times 10^6$ additional function