CHAPTER 3

# COMBINATORIAL OPTIMIZATION

It is humbling to learn that there are entire classes of optimization problems for which most methods—including those described previously—are utterly useless.

We will pose these problems as maximizations except in Section 3.3, although in nonstatistical contexts minimization is often customary. For statistical applications, recall that maximizing the log likelihood is equivalent to minimizing the negative log likelihood.

Let us assume that we are seeking the maximum of $f(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$, where $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ and $\boldsymbol{\Theta}$ consists of $N$ elements for a finite positive integer $N$. In statistical applications, it is not uncommon for a likelihood function to depend on configuration parameters that describe the form of a statistical model and for which there are many discrete choices, as well as a small number of other parameters that could be easily optimized if the best configuration were known. In such cases, we may view $f(\boldsymbol{\theta})$ as the log profile likelihood of a configuration, $\boldsymbol{\theta}$, that is, the highest likelihood attainable using that configuration. Section 3.1.1 provides several examples.

Each $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ is termed a *candidate solution*. Let $f_{\max}$ denote the globally maximum value of $f(\boldsymbol{\theta})$ achievable for $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, and let the set of global maxima be $\mathcal{M} = \{\boldsymbol{\theta} \in \boldsymbol{\Theta} : \ f(\boldsymbol{\theta}) = f_{\max}\}$. If there are ties, $\mathcal{M}$ will contain more than one element. Despite the finiteness of $\boldsymbol{\Theta}$, finding an element of $\mathcal{M}$ may be very hard if there are distracting local maxima, plateaus, and long paths toward optima in $\boldsymbol{\Theta}$, and if $N$ is extremely large.

## 3.1 HARD PROBLEMS AND NP-COMPLETENESS

Hard optimization problems are generally combinatorial in nature. In such problems, $p$ items may be combined or sequenced in a very large number of ways, and each choice corresponds to one element in the space of possible solutions. Maximization requires a search of this very large space.

For example, consider the *traveling salesman problem*. In this problem, the salesman must visit each of $p$ cities exactly once and return to his point of origin, using the shortest total travel distance. We seek to minimize the total travel distance over all possible routes (i.e., maximize the negative distance). If the distance between two cities does not depend on the direction traveled between them, then there are

*Computational Statistics*, Second Edition. Geof H. Givens and Jennifer A. Hoeting.

$(p-1)!/2$ possible routes (since the point of origin and direction of travel are arbitrary). Note that any tour corresponds to a permutation of the integers $1, \ldots, p$, which specifies the sequence in which the cities are visited.

To consider the difficulty of such problems, it is useful to discuss the number of steps required for an algorithm to solve it, where steps are simple operations like arithmetic, comparisons, and branching. The number of operations depends, of course, on the size of the problem posed. In general the size of a problem may be specified as the number of inputs needed to pose it. The traveling salesman problem is posed by specifying $p$ city locations to be sequenced. The difficulty of a particular size-$p$ problem is characterized by the number of operations required to solve it in the worst case using the best known algorithm.

The number of operations is only a rough notion, because it varies with implementation language and strategy. It is conventional, however, to bound the number of operations using the notation $\mathcal{O}(h(p))$. If $h(p)$ is polynomial in $p$, an algorithm is said to be polynomial.

Although the actual running time on a computer depends on the speed of the computer, we generally equate the number of operations and the execution time by relying on the simplifying assumption that all basic operations take the same amount of time (one unit). Then we may make meaningful comparisons of algorithm speeds even though the absolute scale is meaningless.

Consider two problems of size $p = 20$. Suppose that the first problem can be solved in polynomial time [say $\mathcal{O}(p^2)$ operations], and the solution requires 1 minute on your office computer. Then the size-21 problem could be solved in just a few seconds more. The size-25 problem can be solved in 1.57 minutes, size 30 in 2.25 minutes, and size 50 in 6.25 minutes. Suppose the second problem is $\mathcal{O}(p!)$ and requires 1 minute for size 20. Then it would take 21 minutes for size 21, 12.1 years (6,375,600 minutes) for size 25, 207 million years for size 30, and $2.4 \times 10^{40}$ years for size 50. Similarly, if an $\mathcal{O}(p!)$ traveling salesman problem of size 20 could be solved in 1 minute, it would require far longer than the lifetime of the universe to determine the optimal path for the traveling salesman to make a tour of the 50 U.S. state capitals. Furthermore, obtaining a computer that is 1000 times faster would barely reduce the difficulty. The conclusion is stark: Some optimization problems are simply too hard. The complexity of a polynomial problem—even for large $p$ and high polynomial order—is dwarfed by the complexity of a quite small nonpolynomial problem.

The theory of problem complexity is discussed in [214, 497]. For us to discuss this issue further, we must make a formal distinction between *optimization* (i.e., search) problems and *decision* (i.e., recognition) problems. Thus far, we have considered optimization problems of the form: "Find the value of $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ that maximizes $f(\boldsymbol{\theta})$." The decision counterpart to this is: "Is there a $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ for which $f(\boldsymbol{\theta}) > c$, for a fixed number $c$?" Clearly there is a close relationship between these two versions of the problem. In principle, we could solve the optimization problem by repeatedly solving the decision problem for strategically chosen values of $c$.

Decision problems that can be solved in polynomial time [e.g., $\mathcal{O}(p^k)$ operations for $p$ inputs and constant $k$] are generally considered to be efficiently solvable [214]. These problems belong to the class denoted P. Once any polynomial–time algorithm has been identified for a problem, the order of the polynomial is often quickly reduced

to practical levels [497]. Decision problems for which a given solution can be checked in polynomial time are called NP problems. Clearly a problem in P is in NP. However, there seem to be many decision problems, like the traveling salesman problem, that are much easier to check than they are to solve. In fact, there are many NP problems for which no polynomial–time solution has ever been developed. Many NP problems have been proven to belong to a special class for which a polynomial algorithm found to solve one such problem could be used to solve all such problems. This is the class of NP-complete problems. There are other problems at least as difficult, for which a polynomial algorithm—if found—would be known to provide a solution to all NP-complete problems, even though the problem itself is not proven to be NP-complete. These are NP-hard problems. There are also many combinatorial decision problems that are difficult and probably NP-complete or NP-hard although they haven't been proven to be in these classes. Finally, optimization problems are no easier than their decision counterparts, and we may classify optimization problems using the same categories listed above.

It has been shown that if there is a polynomial algorithm for any NP-complete problem, then there are polynomial algorithms for all NP-complete problems. The utter failure of scientists to develop a polynomial algorithm for any NP-complete problem motivates the popular conjecture that there cannot be any polynomial algorithm for any NP-complete problem. Proof (or counterexample) of this conjecture is one of the great unsolved problems in mathematics.

This leads us to the realization that there are optimization problems that are inherently too difficult to solve exactly by traditional means. Many problems in bioinformatics, experimental design, and nonparametric statistical modeling, for example, require combinatorial optimization.

### 3.1.1 Examples

Statisticians have been slow to realize how frequently combinatorial optimization problems are encountered in mainstream statistical model-fitting efforts. Below we give two examples. In general, when fitting a model requires optimal decisions about the inclusion, exclusion, or arrangement of a number of parameters in a set of possible parameters, combinatorial optimization problems arise frequently.

**Example 3.1 (Genetic Mapping)** Genetic data for individuals and groups of related individuals are often analyzed in ways that present highly complex combinatorial optimization problems. For example, consider the problem of locating genes on a chromosome, known as the genetic mapping problem.

The genes, or more generally genetic markers, of interest in a chromosome can be represented as a sequence of symbols. The position of each symbol along the chromosome is called its *locus*. The symbols indicate genes or genetic markers, and the particular content stored at a locus is an *allele*.

Diploid species like humans have pairs of chromosomes and hence two alleles at any locus. An individual is *homozygous* at a locus if the two alleles are identical at this locus; otherwise the individual is *heterozygous*. In either case, each parent contributes one allele at each locus of an offspring's chromosome pair. There are

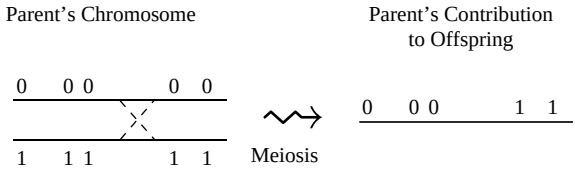


**FIGURE 3.1** During meiosis, a crossover occurs between the third and fourth loci. The zeros and ones indicate the origin of each allele in the contributed chromosome. Only one parental contribution is shown, for simplicity.

two possible contributions from any parent, because the parent has two alleles at the corresponding locus in his/her chromosome pair. Although each parent allele has a 50% chance of being contributed to the offspring, the contributions from a particular parent are not made independently at random. Instead, the contribution by a parent consists of a chromosome built during *meiosis* from segments of each chromosome in the parent's pair of chromosomes. These segments will contain several loci. When the source of the alleles on the contributed chromosome changes from one chromosome of the parent's pair to the other one, a *crossover* is said to have occurred. Figure 3.1 illustrates a crossover occurring during meiosis, forming the chromosome contributed to the offspring by one parent. This method of contribution means that alleles whose loci are closer together on one of the parent's chromosomes are more likely to appear together on the chromosome contributed by that parent.

When the alleles at two loci of a parent's chromosome appear jointly on the contributed chromosome more frequently than would be expected by chance alone, they are said to be *linked*. When the alleles at two different loci of a parent's chromosome do not both appear in the contributed chromosome, a *recombination* has occurred between the loci. The frequency of recombinations determines the degree of linkage between two loci: Infrequent recombination corresponds to strong linkage. The degree of linkage, or *map distance*, between two loci corresponds to the expected number of crossovers between the two loci.

A genetic map of $p$ markers consists of an ordering of their loci and a list of distances or probabilities of recombination between adjacent loci. Assign to each locus a label, $\ell$, for $\ell = 1, \ldots, p$. The ordering component of the map, denoted $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$, describes the arrangement of the $p$ locus labels in order of their positions along the chromosome, with $\theta_j = \ell$ if the locus labeled $\ell$ lies at the $j$th position along the chromosome. Thus, $\boldsymbol{\theta}$ is a permutation of the integers $1, \ldots, p$. The other component of a genetic map is a list of distances between adjacent loci. Denote the probability of recombination between adjacent loci $\theta_j$ and $\theta_{j+1}$ as $d(\theta_j, \theta_{j+1})$. This amounts to the map distance between these loci. Figure 3.2 illustrates this notation.

Such a map can be estimated by observing the alleles at the $p$ loci for a sample of $n$ chromosomes generated during the meiosis from a parent that is heterozygous at all $p$ loci. Each such chromosome can be represented by a sequence of zeros and ones, indicating the origin of each allele in the contributed parent. For example, the chromosome depicted on the right side of Figure 3.1 can be denoted 00011, because

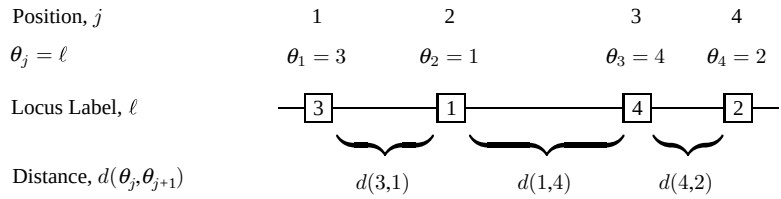


**FIGURE 3.2** Notation for gene mapping example with $p = 4$ loci. The loci are labeled in boxes at their positions along the chromosome. The correct sequential ordering of loci is defined by the $\theta_j$ values. Distances between loci are given by $d(\theta_j, \theta_{j+1})$ for $j = 1, \ldots, 3$.

the first three alleles originate from the first chromosome of the parent and the final two alleles originate from the second chromosome of the parent.

Let the random variable $X_{i,\theta_j}$ denote the origin of the allele in the locus labeled $\theta_j$ for the $i$th chromosome generated during meiosis. The dataset consists of observations, $x_{i,\theta_j}$, of these random variables. Thus, a recombination for two adjacent markers has been observed in the $i$th case if $\left|x_{i,\theta_j} - x_{i,\theta_{j+1}}\right| = 1$, and no recombination has been observed if $\left|x_{i,\theta_j} - x_{i,\theta_{j+1}}\right| = 0$. If recombination events are assumed to occur independently in each interval, the probability of a given map is

$$\prod_{j=1}^{p-1}\prod_{i=1}^{n}\Big\{\big(1 - d(\theta_j, \theta_{j+1})\big)\big(1 - \left|x_{i,\theta_j} - x_{i,\theta_{j+1}}\right|\big) + d(\theta_j, \theta_{j+1})\left|x_{i,\theta_j} - x_{i,\theta_{j+1}}\right|\Big\}. \tag{3.1}$$

Given an ordering $\boldsymbol{\theta}$, the MLEs for the recombination probabilities are easily found to be

$$\hat{d}(\theta_j, \theta_{j+1}) = \frac{1}{n}\sum_{i=1}^{n}\left|x_{i,\theta_j} - x_{i,\theta_{j+1}}\right|. \tag{3.2}$$

Given $d(\theta_j, \theta_{j+1})$, the number of recombinations between the loci in positions $j$ and $j+1$ is $\sum_{i=1}^{n}|X_{i,\theta_j} - X_{i,\theta_{j+1}}|$, which has a Bin$(n, d(\theta_j, \theta_{j+1}))$ distribution. We can compute the profile likelihood for $\boldsymbol{\theta}$ by adding the log likelihoods of the $p-1$ sets of adjacent loci and replacing each $d(\theta_j, \theta_{j+1})$ by its conditional maximum likelihood estimate $\hat{d}(\theta_j, \theta_{j+1})$. Let $\hat{\mathbf{d}}(\boldsymbol{\theta})$ compute these maximum likelihood estimates for any $\boldsymbol{\theta}$. Then the profile likelihood for $\boldsymbol{\theta}$ is

$$\begin{aligned} l(\boldsymbol{\theta}|\hat{\mathbf{d}}(\boldsymbol{\theta})) &= \sum_{j=1}^{p-1} n\left\{\hat{d}(\theta_j, \theta_{j+1})\log\{\hat{d}(\theta_j, \theta_{j+1})\}\right. \\ &\quad \left. + (1 - \hat{d}(\theta_j, \theta_{j+1}))\log\{1 - \hat{d}(\theta_j, \theta_{j+1})\}\right\} \\ &= \sum_{j=1}^{p-1} T(\theta_j, \theta_{j+1}), \end{aligned} \tag{3.3}$$

where $T(\theta_j, \theta_{j+1})$ is defined to be zero if $\hat{d}(\theta_j, \theta_{j+1})$ is zero or one. Then the maximum likelihood genetic map is obtained by maximizing (3.3) over all permutations $\boldsymbol{\theta}$. Note that (3.3) constitutes a sum of terms $T(\theta_j, \theta_{j+1})$ whose values depend on only two loci. Suppose that all possible pairs of loci are enumerated, and the value $T(i, j)$ is computed for every $i$ and $j$ where $1 \leq i < j \leq p$. There are $p(p-1)/2$ such values of $T(i, j)$. The profile log likelihood can then be computed rapidly for any permutation $\boldsymbol{\theta}$ by summing the necessary values of $T(i, j)$.

However, finding the maximum likelihood genetic map requires maximizing the profile likelihood by searching over all $p!/2$ possible permutations. This is a variant of the traveling salesman problem, where each genetic marker corresponds to a city and the distance between cities $i$ and $j$ is $T(i, j)$. The salesman's tour may start at any city and terminates in the last city visited. A tour and its reverse are equivalent. There are no known algorithms for solving general traveling salesman problems in polynomial time.

Further details and extensions of this example are considered in [215, 572]. □

**Example 3.2 (Variable Selection in Regression)** Consider a multiple linear regression problem with $p$ potential predictor variables. A fundamental step in regression is selection of a suitable model. Given a dependent variable $Y$ and a set of candidate predictors $x_1, x_2, \ldots, x_p$, we must find the best model of the form $Y = \beta_0 + \sum_{j=1}^{s} \beta_{i_j} x_{i_j} + \epsilon$, where $\{i_1, \ldots, i_s\}$ is a subset of $\{1, \ldots, p\}$ and $\epsilon$ denotes a random error. The notion of what model is best may have any of several meanings.

Suppose that the goal is to use the Akaike information criterion (AIC) to select the best model [7, 86]. We seek to find the subset of predictors that minimizes the fitted model AIC,

$$\text{AIC} = N \log\{\text{RSS}/N\} + 2(s + 2), \tag{3.4}$$

where $N$ is the sample size, $s$ is the number of predictors in the model, and RSS is the sum of squared residuals. Alternatively, suppose that Bayesian regression is performed, say with the normal-gamma conjugate class of priors $\boldsymbol{\beta} \sim N(\boldsymbol{\mu}, \sigma^2 \mathbf{V})$ and $\nu\lambda/\sigma^2 \sim \chi^2_\nu$. In this case, one might seek to find the subset of predictors corresponding to the model that maximizes the posterior model probability [527].

In either case, the variable selection problem requires an optimization over a space of $2^{p+1}$ possible models, since each variable and the intercept may be included or omitted. It also requires estimating the best $\beta_{i_j}$ for each of the $2^{p+1}$ possible models, but this step is easy for any given model. Although a search algorithm that is more efficient than exhaustive search has been developed to optimize some classical regression model selection criteria, it is practical only for fairly small $p$ [213, 465]. We know of no efficient general algorithm to find the global optimum (i.e., the single best model) for either the AIC or the Bayesian goals. □

### 3.1.2 Need for Heuristics

The existence of such challenging problems requires a new perspective on optimization. It is necessary to abandon algorithms that are guaranteed to find the global

maximum (under suitable conditions) but will never succeed within a practical time limit. Instead we turn to algorithms that can find a good local maximum within tolerable time.

Such algorithms are sometimes called heuristics. They are intended to find a *globally competitive* candidate solution (i.e., a nearly optimal one), with an explicit trade of global optimality for speed. The two primary features of such heuristics are

1. iterative improvement of a current candidate solution, and
2. limitation of the search to a local neighborhood at any particular iteration.

These two characteristics embody the heuristic strategy of *local search*, which we address first.

No single heuristic will work well in all problems. In fact, there is no search algorithm whose performance is better than another when performance is averaged over the set of all possible discrete functions [576, 672]. There is clearly a motivation to adopt different heuristics for different problems. Thus we continue beyond local search to examine *simulated annealing*, *genetic algorithms*, and *tabu algorithms*.

## 3.2 LOCAL SEARCH

Local search is a very broad optimization paradigm that arguably encompasses all of the techniques described in this chapter. In this section, we introduce some of the its simplest, most generic variations such as *k-optimization* and *random starts local search*.

Basic local search is an iterative procedure that updates a current candidate solution $\boldsymbol{\theta}^{(t)}$ at iteration $t$ to $\boldsymbol{\theta}^{(t+1)}$. The update is termed a *move* or a *step*. One or more possible moves are identified from a neighborhood of $\boldsymbol{\theta}^{(t)}$, say $\mathcal{N}(\boldsymbol{\theta}^{(t)})$. The advantage of local search over global (i.e., exhaustive) search is that only a tiny portion of $\boldsymbol{\Theta}$ need be searched at any iteration, and large portions of $\boldsymbol{\Theta}$ may never be examined. The disadvantage is that the search is likely to terminate at an uncompetitive local maximum.

A neighborhood of the current candidate solution, $\mathcal{N}(\boldsymbol{\theta}^{(t)})$, contains candidate solutions that are near $\boldsymbol{\theta}^{(t)}$. Often, proximity is enforced by limiting the number of changes to the current candidate solution used to generate an alternative. In practice, simple changes to the current candidate solution are usually best, resulting in small neighborhoods that are easily searched or sampled. Complex alterations are often difficult to conceptualize, complicated to code, and slow to execute. Moreover, they rarely improve search performance, despite the intuition that larger neighborhoods would be less likely to lead to entrapment at a poor local maximum. If the neighborhood is defined by allowing as many as $k$ changes to the current candidate solution in order to produce the next candidate, then it is a *k-neighborhood*, and the alteration of those features is called a *k-change*.

The definition a neighborhood is intentionally vague to allow flexible usage of the term in a wide variety of problems. For the gene mapping problem introduced in Example 3.1, suppose $\boldsymbol{\theta}^{(t)}$ is a current ordering of genetic markers. A simple

neighborhood might be the set of all orderings that can be obtained by swapping the locations of only two markers on the chromosome whose order is $\boldsymbol{\theta}^{(t)}$. In the regression model selection problem introduced in Example 3.2, a simple neighborhood is the set of models that either add or omit one predictor from $\boldsymbol{\theta}^{(t)}$.

A local neighborhood will usually contain several candidate solutions. An obvious strategy at each iteration is to choose the best among all candidates in the current neighborhood. This is the method of *steepest ascent*. To speed performance, one might instead select the first randomly chosen neighbor for which the objective function exceeds its previous value; this is *random ascent* or *next ascent*.

If $k$-neighborhoods are used for a steepest ascent algorithm, the solution is said to be *$k$-optimal*. Alternatively, any local search algorithm that chooses $\boldsymbol{\theta}^{(t+1)}$ uphill from $\boldsymbol{\theta}^{(t)}$ is an *ascent algorithm*, even if the ascent is not the steepest possible within $\mathcal{N}(\boldsymbol{\theta}^{(t)})$.

The sequential selection of steps that are optimal in small neighborhoods, disregarding the global problem, is reminiscent of a *greedy algorithm*. A chess player using a greedy algorithm might look for the best immediate move with total disregard to its future consequences: perhaps moving a knight to capture a pawn without recognizing that the knight will be captured on the opponent's next move. Wise selection of a new candidate solution from a neighborhood of the current candidate must balance the need for a narrow focus enabling quick moves against the need to find a globally competitive solution. To avoid entrapment in poor local maxima, it might be reasonable—every once in a while—to eschew some of the best neighbors of $\boldsymbol{\theta}^{(t)}$ in favor of a direction whose rewards are only later realized. For example, when $\boldsymbol{\theta}^{(t)}$ is a local maximum, the approach of *steepest ascent/mildest descent* [306] allows a move to the least unfavorable $\boldsymbol{\theta}^{(t+1)} \in \mathcal{N}(\boldsymbol{\theta}^{(t)})$ (see Section 3.5). There are also a variety of techniques in which a candidate neighbor is selected from $\mathcal{N}(\boldsymbol{\theta}^{(t)})$ and a random decision rule is used to decide whether to adopt it or retain $\boldsymbol{\theta}^{(t)}$. These algorithms generate Markov chains $\{\boldsymbol{\theta}^{(t)}\}$ $(t = 0, 1, \ldots)$ that are closely related to simulated annealing (Section 3.3) and the methods of Chapter 7.

Searching within the current neighborhood for a $k$-change steepest ascent move can be difficult when $k$ is greater than 1 or 2 because the size of the neighborhood increases rapidly with $k$. For larger $k$, it can be useful to break the $k$-change up into smaller parts, sequentially selecting the best candidate solutions in smaller neighborhoods. To promote search diversity, breaking a $k$-change step into several smaller sequential changes can be coupled with the strategy of allowing one or more of the smaller steps to be suboptimal (e.g., random). Such *variable-depth local search* approaches permit a potentially better step away from the current candidate solution, even though it will not likely be optimal within the $k$-neighborhood.

Ascent algorithms frequently converge to local maxima that are not globally competitive. One approach to overcoming this problem is the technique of *random starts local search*. Here, a simple ascent algorithm is repeatedly run to termination from a large number of starting points. The starting points are chosen randomly. The simplest approach is to select starting points independently and uniformly at random over $\boldsymbol{\Theta}$. More sophisticated approaches may employ some type of stratified sampling where the strata are identified from some pilot runs in an effort to partition $\boldsymbol{\Theta}$ into regions of qualitatively different convergence behavior.

**TABLE 3.1** Potential predictors of baseball players' salaries.

| | | |
|---|---|---|
| 1. Batting average | 10. Strikeouts (SOs) | 19. Walks per SO |
| 2. On base pct. (OBP) | 11. Stolen bases (SBs) | 20. OBP / errors |
| 3. Runs scored | 12. Errors | 21. Runs per error |
| 4. Hits | 13. Free agency[a] | 22. Hits per error |
| 5. Doubles | 14. Arbitration[b] | 23. HRs per error |
| 6. Triples | 15. Runs per SO | 24. SOs × errors |
| 7. Home runs (HRs) | 16. Hits per SO | 25. SBs × OBP |
| 8. Runs batted in (RBIs) | 17. HRs per SO | 26. SBs × runs |
| 9. Walks | 18. RBIs per SO | 27. SBs × hits |

[a] Free agent, or eligible.

[b] Arbitration, or eligible.

It may seem unsatisfying to rely solely on random starts to avoid being fooled by a local maximum. In later sections we introduce methods that modify local search in ways that provide a reasonable chance of finding a globally competitive candidate solution—possibly the global maximum—on any single run. Of course, the strategy of using multiple random starts can be overlaid on any of these approaches to provide additional confidence in the best solution found. Indeed, we recommend that this is always done when feasible.

**Example 3.3 (Baseball Salaries)** Random starts local search can be very effective in practice because it is simple to code and fast to execute, allowing time for a large number of random starts. Here, we consider its application to a regression model selection problem.

Table 3.1 lists 27 baseball performance statistics, such as batting percentages and numbers of home runs, which were collected for 337 players (no pitchers) in 1991. Players' 1992 salaries, in thousands of dollars, may be related to these variables computed from the previous season. These data, derived from the data in [654], may be downloaded from the website for this book. We use the log of the salary variable as the response variable. The goal is to find the best subset of predictors to predict log salary using a linear regression model. Assuming that the intercept will be included in any model, there are $2^{27} = 134{,}217{,}728$ possible models in the search space.

Figure 3.3 illustrates the application of a random starts local search algorithm to minimize the AIC with respect to regression variable selection. The problem can be posed as maximizing the negative of the AIC, thus preserving our preference for uphill search. Neighborhoods were limited to 1-changes generated from the current model by either adding or deleting one predictor. Search was started from 5 randomly selected subsets of predictors (i.e., five starting points), and 14 additional steps were allocated to each start. Each move was made by steepest ascent. Since each steepest ascent step requires searching 27 neighbors, this small example requires 1890 evaluations of the objective function. A comparable limit to objective function evaluations was imposed on examples of other heuristic techniques that follow in the remainder of this chapter.

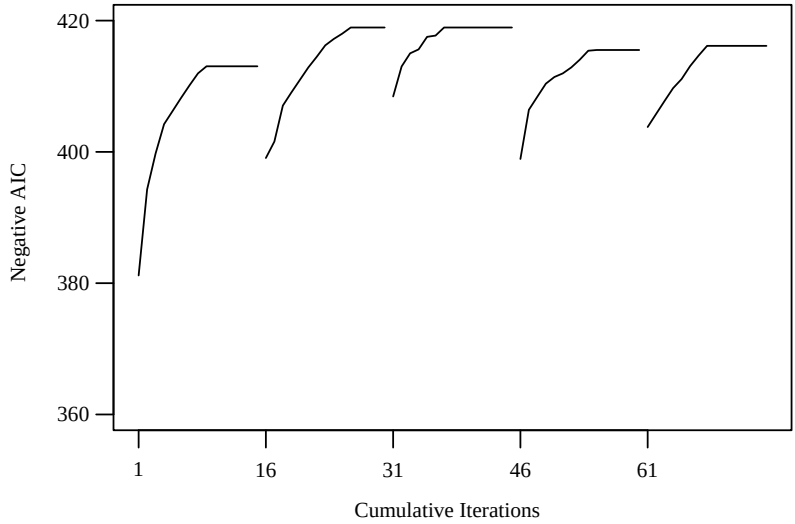


**FIGURE 3.3** Results of random starts local search by steepest ascent for Example 3.3, for 15 iterations from each of five random starts. Only AIC values between −360 and −420 are shown.

Figure 3.3 shows the value of the AIC for the best model at each step. Table 3.2 summarizes the results of the search. The second and third random starts (labeled LS (2,3)) led to an optimal AIC of −418.95, derived from the model using predictors 2, 3, 6, 8, 10, 13, 14, 15, 16, 24, 25, and 26. The worst random start was the first, which led to an AIC of −413.04 for a model with 10 predictors. For the sake of comparison, a greedy stepwise method (the `step()` procedure in S-Plus [642]) chose a model with 12 predictors, yielding an AIC of −418.94. The greedy stepwise method of Efroymson [465] chose a model with 9 predictors, yielding an AIC of −402.16; however, this method is designed to find a good parsimonious model using a criterion that differs slightly from the AIC. With default settings, neither of these off-the-shelf algorithms found a model quite as good as the one found with a simple random starts local search. □

## 3.3 SIMULATED ANNEALING

Simulated annealing is a popular technique for combinatorial optimization because it is generic and easily implemented in its simplest form. Also, its limiting behavior is well studied. On the other hand, this limiting behavior is not easily realized in practice, the speed of convergence can be maddeningly slow, and complex esoteric tinkering may be needed to substantially improve the performance. Useful reviews of simulated annealing include [75, 641].

Annealing is the process of heating up a solid and then cooling it slowly. When a stressed solid is heated, its internal energy increases and its molecules move randomly. If the solid is then cooled slowly, the thermal energy generally decreases slowly, but

**TABLE 3.2** Results of random starts local search model selection for Example 3.3. The bullets indicate inclusion of the corresponding predictor in each model selected, with model labels explained in the text. In addition, all models in this table included predictors 3, 8, 13 and 14.

| | Predictors selected | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | 1 | 2 | 6 | 7 | 9 | 10 | 12 | 15 | 16 | 18 | 19 | 20 | 21 | 22 | 24 | 25 | 26 | 27 | AIC |
| LS (2,3) | | • | • | | | • | | • | • | | | | | | • | • | • | | −418.95 |
| S-Plus | • | | • | | | • | | • | • | | | | | | • | • | • | | −418.94 |
| LS (5) | | | | | | • | • | • | • | | | • | | • | | • | | • | −416.15 |
| LS (4) | | | | | • | • | | | | | • | • | • | • | | | | | −415.52 |
| LS (1) | | | • | | | | • | | | | | | • | • | • | • | | | −413.04 |
| Efroy. | | | | • | | • | | | | • | | | | | • | | • | | −402.16 |

there are also random increases governed by Boltzmann's probability. Namely, at temperature $\tau$, the probability density of an increase in energy of magnitude $\Delta E$ is $\exp\{-\Delta E/k\tau\}$ where $k$ is Boltzmann's constant. If the cooling is slow enough and deep enough, the final state is unstressed, where all the molecules are arranged to have minimal potential energy.

For consistency with the motivating physical process, we pose optimization as minimization in this section, so the minimum of $f(\boldsymbol{\theta})$ is sought over $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. Then it is possible to draw an analogy between the physical cooling process and the process of solving a combinatorial minimization problem [130, 378]. For simulated annealing algorithms, $\boldsymbol{\theta}$ corresponds to the state of the material, $f(\boldsymbol{\theta})$ corresponds to its energy level, and the optimal solution corresponds to the $\boldsymbol{\theta}$ that has minimum energy. Random changes to the current state (i.e., moves from $\boldsymbol{\theta}^{(t)}$ to $\boldsymbol{\theta}^{(t+1)}$) are governed by the Boltzmann distribution given above, which depends on a parameter called temperature. When the temperature is high, acceptance of uphill moves (i.e., moves to a higher energy state) are more likely to be tolerated. This discourages convergence to the first local minimum that happens to be found, which might be premature if the space of candidate solutions has not yet been adequately explored. As search continues, the temperature is lowered. This forces increasingly concentrated search effort near the current local minimum, because few uphill moves will be allowed. If the cooling schedule is determined appropriately, the algorithm will hopefully converge to the global minimum.

The simulated annealing algorithm is an iterative procedure started at time $t = 0$ with an initial point $\boldsymbol{\theta}^{(0)}$ and a temperature $\tau_0$. Iterations are indexed by $t$. The algorithm is run in stages, which we index by $j = 0, 1, 2, \ldots$, and each stage consists of several iterations. The length of the $j$th stage is $m_j$. Each iteration proceeds as follows:

1. Select a candidate solution $\boldsymbol{\theta}^*$ within the neighborhood of $\boldsymbol{\theta}^{(t)}$, say $\mathcal{N}(\boldsymbol{\theta}^{(t)})$, according to a proposal density $g^{(t)}(\cdot \mid \boldsymbol{\theta}^{(t)})$.
2. Randomly decide whether to adopt $\boldsymbol{\theta}^*$ as the next candidate solution or to keep another copy of the current solution. Specifically, let $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^*$ with probability equal to $\min\left(1, \exp\left\{[f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*)]/\tau_j\right\}\right)$. Otherwise, let $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}$.

3. Repeat steps 1 and 2 a total of $m_j$ times.
4. Increment $j$. Update $\tau_j = \alpha(\tau_{j-1})$ and $m_j = \beta(m_{j-1})$. Go to step 1.

If the algorithm is not stopped according to a limit on the total number of iterations or a predetermined schedule of $\tau_j$ and $m_j$, one can monitor an absolute or relative convergence criterion (see Chapter 2). Often, however, the stopping rule is expressed as a minimum temperature. After stopping, the best candidate solution found is the estimated minimum.

The function $\alpha$ should slowly decrease the temperature to zero. The number of iterations at each temperature ($m_j$) should be large and increasing in $j$. Ideally, the function $\beta$ should scale the $m_j$ exponentially in $p$, but in practice some compromises will be required in order to obtain tolerable computing speed.

Although the new candidate solution is always adopted when it is superior to the current solution, note that it has some probability of being adopted even when it is inferior. In this sense, simulated annealing is a stochastic descent algorithm. Its randomness allows simulated annealing sometimes to escape uncompetitive local minima.

## 3.3.1 Practical Issues

***3.3.1.1 Neighborhoods and Proposals*** Strategies for choosing neighborhoods can be very problem specific, but the best neighborhoods are usually small and easily computed.

Consider the traveling salesman problem. Numbering the cities $1, 2, \ldots, p$, any tour $\boldsymbol{\theta}$ can be written as a permutation of these integers. The cities are linked in this order, with an additional link between the final city visited and the original city where the tour began. A neighbor of $\boldsymbol{\theta}$ can be generated by removing two nonadjacent links and reconnecting the tour. In this case, there is only one way to obtain a valid tour through reconnection: One of the tour segments must be reversed. For example, the tour 143256 is a neighbor of the tour 123456. Since two links are altered, the process of generating such neighbors is a 2-change, and it yields a 2-neighborhood. Any tour has $p(p-3)/2$ unique 2-change neighbors distinct from $\boldsymbol{\theta}$ itself. This neighborhood is considerably smaller than the $(p-1)!/2$ tours in the complete solution space.

It is critical that the chosen neighborhood structure allows all solutions in $\boldsymbol{\Theta}$ to *communicate*. For $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ to communicate, it must be possible to find a finite sequence of solutions $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k$ such that $\boldsymbol{\theta}_1 \in \mathcal{N}(\boldsymbol{\theta}_i)$, $\boldsymbol{\theta}_2 \in \mathcal{N}(\boldsymbol{\theta}_1), \ldots, \boldsymbol{\theta}_k \in \mathcal{N}(\boldsymbol{\theta}_{k-1})$, and $\boldsymbol{\theta}_j \in \mathcal{N}(\boldsymbol{\theta}_k)$. The 2-neighborhoods mentioned above for the traveling salesman problem allow communication between any $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$.

The most common proposal density, $g^{(t)}(\cdot \mid \boldsymbol{\theta}^{(t)})$, is discrete uniform—a candidate is sampled completely at random from $\mathcal{N}(\boldsymbol{\theta}^{(t)})$. This has the advantage of speed and simplicity. Other, more strategic methods have also been suggested [281, 282, 659].

Rapid updating of the objective function is an important strategy for speeding simulated annealing runs. In the traveling salesman problem, sampling a 2-neighbor at random amounts to selecting two integers from which is derived a permutation of the current tour. Note also for the traveling salesman problem that $f(\boldsymbol{\theta}^*)$ can be efficiently calculated for any $\boldsymbol{\theta}^*$ in the 2-neighborhood of $\boldsymbol{\theta}^{(t)}$ when $f(\boldsymbol{\theta}^{(t)})$ has

already been found. In this case, the new tour length equals the old tour length minus the distance for traveling the two broken links, plus the distance for traveling the two new links. The time to compute this does not depend on problem size $p$.

#### 3.3.1.2 Cooling Schedule and Convergence

The sequence of stage lengths and temperatures is called the *cooling schedule*. Ideally, the cooling schedule should be slow.

The limiting behavior of simulated annealing follows from Markov chain theory, briefly reviewed in Chapter 1. Simulated annealing can be viewed as producing a sequence of homogeneous Markov chains (one at each temperature) or a single inhomogeneous Markov chain (with temperature decreasing between transitions). Although these views lead to different approaches to defining limiting behavior, both lead to the conclusion that the limiting distribution of draws has support only on the set of global minima.

To understand why cooling should lead to the desired convergence of the algorithm at a global minimum, first consider the temperature to be fixed at $\tau$. Suppose further that proposing $\boldsymbol{\theta}_i$ from $\mathcal{N}(\boldsymbol{\theta}_j)$ has the same probability as proposing $\boldsymbol{\theta}_j$ from $\mathcal{N}(\boldsymbol{\theta}_i)$ for any pair of solutions $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ in $\boldsymbol{\Theta}$. In this case, the sequence of $\boldsymbol{\theta}^{(t)}$ generated by simulated annealing is a Markov chain with stationary distribution $\pi_\tau(\boldsymbol{\theta}) \propto \exp\{-f(\boldsymbol{\theta})/\tau\}$. This means that $\lim_{t\to\infty} P[\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}] = \pi_\tau(\boldsymbol{\theta})$. This approach to generating a sequence of random values is called the *Metropolis algorithm* and is discussed in Section 7.1.

In principle, we would like to run the chain at this fixed temperature long enough that the Markov chain is approximately in its stationary distribution before the temperature is reduced.

Suppose there are $M$ global minima and the set of these solutions is $\mathcal{M}$. Denote the minimal value of $f$ on $\boldsymbol{\Theta}$ as $f_{\min}$. Then the stationary distribution of the chain for a fixed $\tau$ is given by

$$\pi_\tau(\boldsymbol{\theta}_i) = \frac{\exp\{-\left[f(\boldsymbol{\theta}_i) - f_{\min}\right]/\tau\}}{M + \sum_{j \notin \mathcal{M}} \exp\{-\left[f(\boldsymbol{\theta}_j) - f_{\min}\right]/\tau\}} \tag{3.5}$$

for each $\boldsymbol{\theta}_i \in \boldsymbol{\Theta}$.

Now, as $\tau \to 0$ from above, the limit of $\exp\{-\left[f(\boldsymbol{\theta}_i) - f_{\min}\right]/\tau\}$ is 0 if $i \notin \mathcal{M}$ and 1 if $i \in \mathcal{M}$. Thus

$$\lim_{\tau \downarrow 0} \pi_\tau(\boldsymbol{\theta}_i) = \begin{cases} 1/M & \text{if } i \in \mathcal{M}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

The mathematics to make these arguments precise can be found in [67, 641].

It is also possible to relate the cooling schedule to a bound on the quality of the final solution. If one wishes any iterate to have not more than probability $\delta$ in equilibrium of being worse than the global minimum by no more than $\epsilon$, this can be achieved if one cools until $\tau_j \leq \epsilon/\log\{(N-1)/\delta\}$, where $N$ is the number of points in $\boldsymbol{\Theta}$ [426]. In other words, this $\tau_j$ ensures that the final Markov chain configuration will in equilibrium have $P\left[f(\boldsymbol{\theta}^{(t)}) > f_{\min} + \epsilon\right] < \delta$.

If neighborhoods communicate and the depth of the deepest local (and nonglobal) minimum is $c$, then the cooling schedule given by $\tau = c/\log\{1+i\}$ guarantees asymptotic convergence, where $i$ indexes iterations [292]. The depth of a local minimum is defined to be the smallest increase in the objective function needed to escape from that local minimum into the valley of any other minimum. However, mathematical bounds on the number of iterations required to achieve a high probability of having discovered at least one element of $\mathcal{M}$ often exceed the size of $\boldsymbol{\Theta}$ itself. In this case, one cannot establish that simulated annealing will find the global minimum more quickly than an exhaustive search [33].

If one wishes the Markov chain generated by simulated annealing to be approximately in its stationary distribution at each temperature before reducing temperature, then the length of the run ideally should be at least quadratic in the size of the solution space [1], which itself is usually exponential in problem size. Clearly, much shorter stage lengths must be chosen if simulated annealing is to require fewer iterations than exhaustive search.

In practice, many cooling schedules have been tried [641]. Recall that the temperature at stage $j$ is $\tau_j = \alpha(\tau_{j-1})$ and the number of iterations in stage $j$ is $m_j = \beta(m_{j-1})$. One popular approach is to set $m_j = 1$ for all $j$ and reduce the temperature very slowly according to $\alpha(\tau_{j-1}) = \tau_{j-1}/(1 + a\tau_{j-1})$ for a small value of $a$. A second option is to set $\alpha(\tau_{j-1}) = a\tau_{j-1}$ for $a < 1$ (usually $a \geq 0.9$). In this case, one might increase stage lengths as temperatures decrease. For example, consider $\beta(m_{j-1}) = bm_{j-1}$ for $b > 1$, or $\beta(m_{j-1}) = b + m_{j-1}$ for $b > 0$. A third schedule uses

$$\alpha(\tau_{j-1}) = \frac{\tau_{j-1}}{1 + \tau_{j-1}\log\{1+r\}/(3s_{\tau_{j-1}})}$$

where $s^2_{\tau_{j-1}}$ is the square of the mean objective function cost at the current temperature minus the mean squared cost at the current temperature, and $r$ is some small real number [1]. Using the temperature schedule $\tau = c/\log\{1+i\}$ mentioned above based on theory is rarely practical because it is too slow and the determination of $c$ is difficult, with excessively large guesses for $c$ further slowing the algorithm.

Most practitioners require lengthy experimentation to find suitable initial parameter values (e.g., $\tau_0$ and $m_0$) and values of the proposed schedules (e.g., $a$, $b$, and $r$). While selection of the initial temperature $\tau_0$ is usually problem dependent, some general guidelines may be given. A useful strategy is to choose a positive $\tau_0$ value so that $\exp\left\{[f(\boldsymbol{\theta}_i) - f(\boldsymbol{\theta}_j)]/\tau_0\right\}$ is close to 1 for any pair of solutions $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ in $\boldsymbol{\Theta}$. The rationale for this choice is that it provides any point in the parameter space with a reasonable chance of being visited in early iterations of the algorithm. Similarly, choosing $m_j$ to be large can produce a more accurate solution, but can result in long computing times. As a general rule of thumb, larger decreases in temperature require longer runs after the decrease. Finally, a good deal of evidence suggests that running simulated annealing long at high temperatures is not very useful. In many problems, the barriers between local minima are sufficiently modest that jumps between them are possible even at fairly low temperatures. Good cooling schedules therefore decrease the temperature rapidly at first.

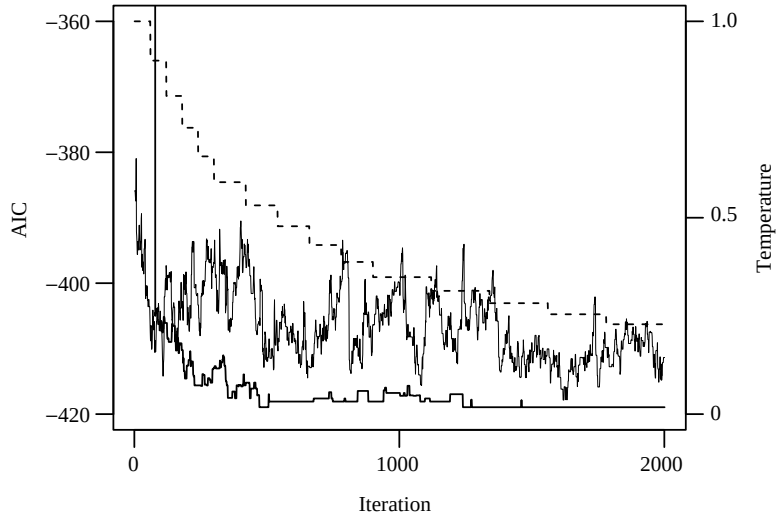


**FIGURE 3.4** Results of two simulated annealing minimizations of the regression model AIC for Example 3.4. The temperature for the bottom curve is shown by the dotted line and the right axis. Only AIC values between −360 and −420 are shown.

**Example 3.4 (Baseball Salaries, Continued)** To implement simulated annealing for variable selection via the AIC in the baseball salary regression problem introduced in Example 3.3, we must establish a neighborhood structure, a proposal distribution, and a temperature schedule. The simplest neighborhoods contain 1-change neighbors generated from the current model by either adding or deleting one predictor. We assigned equal probabilities to all candidates in a neighborhood. The cooling schedule had 15 stages, with stage lengths of 60 for the first 5 stages, 120 for the next 5, and 220 for the final 5. Temperatures were decreased according to $\alpha(\tau_{j-1}) = 0.9\tau_{j-1}$ after each stage.

Figure 3.4 shows the values of the AIC for the sequence of candidate solutions generated by simulated annealing, for two different choices of $\tau_0$. The bottom curve corresponds to $\tau_0 = 1$. In this case, simulated annealing became stuck at particular candidate solutions for distinct periods because the low temperatures allowed little tolerance for uphill moves. In the particular realization shown, the algorithm quickly found good candidate solutions with low AIC values, where it became stuck frequently. However, in other cases (e.g., with a very multimodal objective function), such stickiness may result in the algorithm becoming trapped in a region far from the global minimum. A second run with $\tau_0 = 6$ (top solid line) yielded considerable mixing, with many uphill proposals accepted as moves. The temperature schedule for $\tau_0 = 1$ is shown by the dotted line and the right axis. Both runs exhibited greater mixing at higher temperatures. When $\tau_0 = 1$, the best model found was first identified in the 1274th step and dominated the simulation after that point. This model achieved an AIC of −418.95, and matched the best model found using random starts local search in Table 3.2. When $\tau_0 = 6$, the best model found had an AIC of −417.85.

This run was clearly unsuccessful, requiring more iterations, cooler temperatures, or both. □

### 3.3.2 Enhancements

There are many variations on simulated annealing that purport to improve performance. Here we list a few ideas in an order roughly corresponding to the steps in the basic algorithm.

The simplest way to start simulated annealing is to start once, anywhere. A strategy employing multiple random starts would have the dual advantages of potentially finding a better candidate solution and allowing confirmation of convergence to the particular optimum found. Purely random starts could be replaced by a stratified set of starting points chosen by strategic preprocessing to be more likely to lead to minima than simple random starts. Such strategies must have high payoffs if they are to be useful, given simulated annealing's generally slow convergence. In some cases, the extra iterations dedicated to various random starts may be better spent on a single long run with longer stage sizes and a slower cooling schedule.

The solution space, $\boldsymbol{\Theta}$, may include constraints on $\boldsymbol{\theta}$. For example, in the genetic mapping problem introduced in Example 3.1, $\boldsymbol{\theta}$ must be a permutation of the integers $1, \ldots, p$ when there are $p$ markers. When the process for generating neighbors creates solutions that violate these constraints, substantial time may be wasted fixing candidates or repeatedly sampling from $\mathcal{N}(\boldsymbol{\theta}^{(t)})$ until a valid candidate is found. An alternative is to relax the constraints and introduce a penalty into $f$ that penalizes invalid solutions. In this manner, the algorithm can be discouraged from visiting invalid solutions without dedicating much time to enforcing constraints.

In the basic algorithm, the neighborhood definition is static and the proposal distribution is the same at each iteration. Sometimes improvements can be obtained by adaptively restricting neighborhoods at each iteration. For example, it can be useful to shrink the size of the neighborhood as time increases to avoid many wasteful generations of distant candidates that are very likely to be rejected at such low temperatures. In other cases, when a penalty function is used in place of constraints, it may be useful to allow only neighborhoods composed of solutions that reduce or eliminate constraint violations embodied in the current $\boldsymbol{\theta}$.

It is handy if $f$ can be evaluated quickly for new candidates. We noted previously that neighborhood definitions can sometimes enable this, as in the traveling salesman problem where a 2-neighborhood strategy led to a simple updating formula for $f$. Simple approximation of $f$ is sometimes made, often in a problem-specific manner. At least one author suggests monitoring recent iterates and introducing a penalty term in $f$ that discourages revisiting states like those recently visited [201].

Next consider the acceptance probability given in step 2 of the canonical simulated annealing algorithm in Section 3.3. The expression $\exp\{[f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*)]/\tau_j\}$ is motivated by the Boltzmann distribution from statistical thermodynamics. Other acceptance probabilities can be used, however. The linear Taylor series expansion of the Boltzmann distribution motivates $\min\left\{1, 1 + \left(\left[f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*)\right]/\tau_j\right)\right\}$ as a possible acceptance probability [352]. To encourage moderate moves away from

local minima while preventing excessive small moves, the acceptance probability $\min\left\{1, \exp\left\{\left[c + f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*)\right] \,/\, \tau_j\right\}\right\}$, where $c > 0$, has been suggested for certain problems [169].

In general, there is little evidence that the shape of the cooling schedule (linear, polynomial, exponential) matters much, as long as the useful range of temperatures is covered, the range is traversed at roughly the same rate, and sufficient time is spent at each temperature (especially the low temperatures) [169]. Reheating strategies that allow sporadic, systematic, or interactive temperature increases to prevent getting stuck in a local minimum at low temperatures can be effective [169, 256, 378].

After simulated annealing is complete, one might take the final result of one or more runs and polish these with a descent algorithm. In fact, one could refine occasional accepted steps in the same way, instead of waiting until simulated annealing has terminated.

## 3.4 GENETIC ALGORITHMS

Annealing is not the only natural process successfully exploited as a metaphor to solve optimization problems. *Genetic algorithms* mimic the process of Darwinian natural selection. Candidate solutions to a maximization problem are envisioned as biological organisms represented by their genetic code. The fitness of an organism is analogous to the quality of a candidate solution. Breeding among highly fit organisms provides the best opportunity to pass along desirable attributes to future generations, while breeding among less fit organisms (and rare genetic mutations) ensures population diversity. Over time, the organisms in the population should evolve to become increasingly fit, thereby providing a set of increasingly good candidate solutions to the optimization problem. The pioneering development of genetic algorithms was done by Holland [333]. Other useful references include [17, 138, 200, 262, 464, 531, 533, 661].

We revert now to our standard description of optimization as maximization, where we seek the maximum of $f(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. In statistical applications of genetic algorithms, $f$ is often a joint log profile likelihood function.

### 3.4.1 Definitions and the Canonical Algorithm

***3.4.1.1 Basic Definitions*** In Example 3.1 above, some genetics terminology was introduced. Here we discuss additional terminology needed to study genetic algorithms.

In a genetic algorithm, every candidate solution corresponds to an *individual*, or *organism*, and every organism is completely described by its genetic code. Individuals are assumed to have one *chromosome*. A chromosome is a sequence of $C$ symbols, each of which consists of a single choice from a predetermined alphabet. The most basic alphabet is the binary alphabet, $\{0, 1\}$, in which case a chromosome of length $C = 9$ might look like 100110001. The $C$ elements of the chromosome are the *genes*. The values that might be stored in a gene (i.e., the elements of the alphabet) are *alleles*. The position of a gene in the chromosome is its *locus*.

The information encoded in an individual's chromosome is its *genotype*. We will represent a chromosome or its genotype as $\boldsymbol{\vartheta}$. The expression of the genotype in the organism itself is its *phenotype*. For optimization problems, phenotypes are candidate solutions and genotypes are encodings: Each genotype, $\boldsymbol{\vartheta}$, encodes a phenotype, $\boldsymbol{\theta}$, using the chosen allele alphabet.

Genetic algorithms are iterative, with iterations indexed by $t$. Unlike the methods previously discussed in this chapter, genetic algorithms track more than one candidate solution simultaneously. Let the $t$th *generation* consist of a collection of $P$ organisms, $\boldsymbol{\vartheta}_1^{(t)}, \ldots, \boldsymbol{\vartheta}_P^{(t)}$. This population of size $P$ at generation $t$ corresponds to a collection of candidate solutions, $\boldsymbol{\theta}_1^{(t)}, \ldots, \boldsymbol{\theta}_P^{(t)}$.

Darwinian natural selection favors organisms with high *fitness*. The fitness of an organism $\boldsymbol{\vartheta}_i^{(t)}$ depends on the corresponding $f(\boldsymbol{\theta}_i^{(t)})$. A high-quality candidate solution has a high value of the objective function and a high fitness. As generations progress, organisms inherit from their parents bits of genetic code that are associated with high fitness if fit parents are predominantly selected for breeding. An *offspring* is a new organism inserted in the $(t+1)$th generation to replace a member of the $t$th generation; the offspring's chromosome is determined from those of two *parent* chromosomes belonging to the $t$th generation.

To illustrate some of these ideas, consider a regression model selection problem with 9 predictors. Assume that an intercept will be included in any model. The genotype of any model can then be written as a chromosome of length 9. For example, the chromosome $\boldsymbol{\vartheta}_i^{(t)} = 100110001$ is a genotype corresponding the phenotype of a model containing only the fitted parameters for the intercept and predictors 1, 4, 5, and 9.

Another genotype is $\boldsymbol{\vartheta}_j^{(t)} = 110100110$. Notice that $\boldsymbol{\vartheta}_i^{(t)}$ and $\boldsymbol{\vartheta}_j^{(t)}$ share some common genes. A *schema* is any subcollection of genes. In this example, the two chromosomes share the schema $1{*}01{*****}$, where $*$ represents a wildcard: The allele in that locus is ignored. (These two chromosomes also share the schemata $**01{*****}$, $1{*}01{*}0{***}$, and others.) The significance of schemata is that they encode modest bits of genetic information that may be transferred as a unit from parent to offspring. If a schema is associated with a phenotypic feature that induces high values of the objective function, then the inheritance of this schema by individuals in future generations promotes optimization.

***3.4.1.2 Selection Mechanisms and Genetic Operators*** Breeding drives most genetic change. The process by which parents are chosen to produce offspring is called the *selection mechanism*. One simple approach is to select one parent with probability proportional to fitness and to select the other parent completely at random. Another approach is to select each parent independently with probability proportional to fitness. Section 3.4.2.2 describes some of the most frequently used selection mechanisms.

After two parents from the $t$th generation have been selected for breeding, their chromosomes are combined in some way that allows schemata from each parent to be inherited by their offspring, who become members of generation $t+1$. The methods for producing offspring chromosomes from chosen parent chromosomes are *genetic operators*.

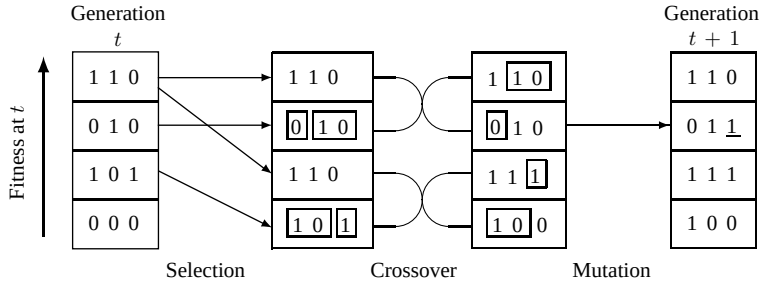


**FIGURE 3.5** An example of generation production in a genetic algorithm for a population of size $P = 4$ with chromosomes of length $C = 3$. Crossovers are illustrated by boxing portions of some chromosomes. Mutation is indicated by an underlined gene in the final column.

A fundamental genetic operator is *crossover*. One of the simplest crossover methods is to select a random position between two adjacent loci and split both parent chromosomes at this position. Glue the left chromosome segment from one parent to the right segment from the other parent to form an offspring chromosome. The remaining segments can be combined to form a second offspring or discarded. For example, suppose the two parents are 100110001 and 110100110. If the random split point is between the third and fourth loci, then the potential offspring are 100100110 and 110110001. Note that in this example, both offspring inherit the schema $1{*}01{*}{*}{*}{*}{*}$. Crossover is the key to a genetic algorithm—it allows good features of two candidate solutions to be combined. Some more complicated crossover operators are discussed in Section 3.4.2.3.

*Mutation* is another important genetic operator. Mutation changes an offspring chromosome by randomly introducing one or more alleles in loci where those alleles are not seen in the corresponding loci of either parent chromosome. For example, if crossover produced 100100110 from the parents mentioned above, subsequent mutation might yield 101100110. Note that the third gene was 0 in both parents and therefore crossover alone was guaranteed to retain the schema $**0{*}{*}{*}{*}{*}{*}$. Mutation, however, provides a way to escape this constraint, thereby promoting search diversification and providing a way to escape from local maxima.

Mutation is usually applied after breeding. In the simplest implementation, each gene has an independent probability, $\mu$, of mutating, and the new allele is chosen completely at random from the genetic alphabet. If $\mu$ is too low, many potentially good innovations will be missed; if $\mu$ is too high, the algorithm's ability to learn over time will be degraded, because excessive random variation will disturb the fitness selectivity of parents and the inheritance of desirable schemata.

To summarize, genetic algorithms proceed by producing generations of individuals. The $(t + 1)$th generation is produced as follows. First the individuals in generation $t$ are ranked and selected according to fitness. Then crossover and mutation are applied to these selected individuals to produce generation $t + 1$. Figure 3.5 is a small example of the production of a generation of four individuals with three chromosomes per individual and binary chromosome encoding. In generation $t$, individual 110 has the highest fitness among its generation and is chosen twice in the

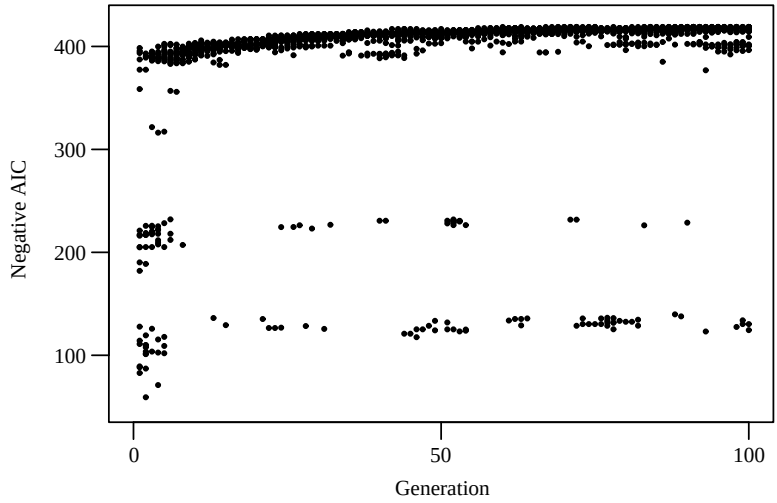


**FIGURE 3.6** Results of a genetic algorithm for Example 3.5.

selection stage. In the crossover stage, the selected individuals are paired off so that each pair recombines to generate two new individuals. In the mutation stage, a low mutation rate is applied. In this example, only one mutation occurs. The completion of these steps yields the new generation.

**Example 3.5 (Baseball Salaries, Continued)** The results of applying a simple genetic algorithm to the variable selection problem for the baseball data introduced in Example 3.3 are shown in Figure 3.6. One hundred generations of size $P = 20$ were used. Binary inclusion–exclusion alleles were used for each possible predictor, yielding chromosomes of length $C = 27$. The starting generation consisted of purely random individuals. A rank-based fitness function was used; see Equation (3.9). One parent was selected with probability proportional to this fitness; the other parent was selected independently, purely at random. Breeding employed simple crossover. A 1% mutation rate was randomly applied independently to each locus.

The horizontal axis in Figure 3.6 corresponds to generation. The AIC values for all 20 individuals in each generation are plotted. The best model found included predictors 2, 3, 6, 8, 10, 13, 14, 15, 16, 24, 25, and 26, yielding an AIC of $-418.95$. This matches the best model found using random starts local search (Table 3.2). Darwinian survival of the fittest is clearly illustrated in this figure: The 20 random starting individuals rapidly coalesce into 3 effective subspecies, with the best of these quickly overwhelming the rest and slowly improving thereafter. The best model was first found in generation 60. □

#### *3.4.1.3 Allele Alphabets and Genotypic Representation*

The binary alphabet for alleles was introduced in the pioneering work of Holland [333] and continues to be very prevalent in recent research. The theoretical behavior of the algorithm and the relative performance of various genetic operators and other algorithmic variations are better understood for binary chromosomes than for other choices.

For many optimization problems, it is possible to construct a binary encoding of solutions. For example, consider the univariate optimization of $f(\theta) = 100 - (\theta - 4)^2$ on the range $\theta \in [1, 12.999] = [a_1, a_2]$. Suppose that we represent a number in $[a_1, a_2]$ as

$$a_1 + \left( \frac{a_2 - a_1}{2^d - 1} \right) \text{decimal}(b), \tag{3.7}$$

where $b$ is a binary number of $d$ digits and the decimal() function converts from base 2 to base 10. If $c$ decimal places of accuracy are required, then $d$ must be chosen to satisfy

$$(a_2 - a_1)10^c \leq 2^d - 1. \tag{3.8}$$

In our example, 14 binary digits are required for accuracy to 3 decimal places, and $b = 01000000000000$ maps to $\theta = 4.000$ using Equation (3.7).

In some cases, such as the regression model selection problem, a binary-encoded chromosome may be very natural. In others, however, the encoding seems forced, as it does above. For $f(\theta) = 100 - (\theta - 4)^2$, the chromosome $\vartheta = 01000000000000$ ($\theta = 4.000$) is optimal. However, chromosomes that are genetically close to this, such as 10000000000000 ($\theta = 7.000$) and 00000000000000 ($\theta = 1.000$), have phenotypes that are not close to $\theta = 4.000$. On the other hand, the genotype 00111111111111 has phenotype very close to 4.000 even though the genotype is very different than 01000000000000. Chromosomes that are similar in genotype may have very different phenotypes. Thus, a small mutation may move to a drastically different region of solution space, and a crossover may produce offspring whose phenotypes bear little resemblance to either parent. To resolve such difficulties, a different encoding scheme or modified genetic operators may be required (see Section 3.4.2.3).

An important alternative to binary representation arises in permutation problems of size $p$, like the traveling salesman problem. In such cases, a natural chromosome is a permutation of the integers $1, \ldots, p$, for example, $\vartheta = 752631948$ when $p = 9$. Since such chromosomes must obey the requirement that each integer appear in exactly one locus, some changes to standard genetic operators will be required. Strategies for dealing with permutation chromosomes are discussed in Section 3.4.2.3.

***3.4.1.4 Initialization, Termination, and Parameter Values*** Genetic algorithms are usually initialized with a first generation of purely random individuals.

The size of the generation, $P$, affects the speed, convergence behavior, and solution quality of the algorithm. Large values of $P$ are to be preferred, if feasible, because they provide a more diverse genetic pool from which to generate offspring, thereby diversifying the search and discouraging premature convergence. For binary encoding of chromosomes, one suggestion is to choose $P$ to satisfy $C \leq P \leq 2C$, where $C$ is the chromosome length [8]. For permutation chromosomes, the range $2C \leq P \leq 20C$ has been suggested [335]. In most real applications, population sizes have ranged between 10 and 200 [566], although a review of empirical studies suggests that $P$ can often be as small as 30 [531].

Mutation rates are typically very low, in the neighborhood of 1%. Theoretical work and empirical studies have supported a rate of $1/C$ [464], and another investigation suggested that the rate should be nearly proportional to $1/(P\sqrt{C})$ [571]. Nevertheless, a fixed rate independent of $P$ and $C$ is a common choice.

The termination criterion for a genetic algorithm is frequently just a maximum number of iterations chosen to limit computing time. One might instead consider stopping when the genetic diversity within chromosomes in the current generation is sufficiently low [17].

### 3.4.2 Variations

In this section we survey a number of methodological variations that may offer improved performance. These include alterations to the fitness function, selection mechanism, genetic operators, and other aspects of the basic algorithm.

***3.4.2.1 Fitness*** In a canonical genetic algorithm, the fitness of an organism is often taken to be the objective function value of its phenotype, perhaps scaled by the mean objective function value in its generation. It is tempting to simply equate the objective function value $f(\boldsymbol{\theta})$ to the fitness because the fittest individual then corresponds to the maximum likelihood solution. However, directly equating an organism's fitness to the objective function value for its corresponding phenotype is usually naive in that other choices yield superior optimization performance. Instead, let $\phi(\boldsymbol{\vartheta})$ denote the value of a *fitness function* that describes the fitness of a chromosome. The fitness function will depend on the objective function $f$, but will not equal it. This increased flexibility can be exploited to enhance search effectiveness.

A problem seen in some applications of genetic algorithms is excessively fast convergence to a poor local optimum. This can occur when a few of the very best individuals dominate the breeding and their offspring saturate subsequent generations. In this case, each subsequent generation consists of genetically similar individuals that lack the genetic diversity needed to produce offspring that might typify other, more profitable regions of solution space. This problem is especially troublesome if it occurs directly after initialization, when nearly all individuals have very low fitness. A few chromosomes that are more fit than the rest can then pull the algorithm to an unfavorable local maximum. This problem is analogous to entrapment near an uncompetitive local maximum, which is also a concern for the other search methods discussed earlier in this chapter.

Selective pressure must be balanced carefully, however, because genetic algorithms can be slow to find a very good optimum. It is therefore important to maintain firm selective pressure without allowing a few individuals to cause premature convergence. To do this, the fitness function can be designed to reduce the impact of large variations in $f$.

A common approach is to ignore the values of $f(\boldsymbol{\theta}_i^{(t)})$ and use only their ranks [18, 532, 660]. For example, one could set

$$\phi(\boldsymbol{\vartheta}_i^{(t)}) = \frac{2r_i}{P(P+1)}, \tag{3.9}$$

where $r_i$ is the rank of $f(\boldsymbol{\theta}_i^{(t)})$ among generation $t$. This strategy gives the chromosome corresponding to the median quality candidate a selection probability of $1/P$, and the best chromosome has probability $2/(P+1)$, roughly double that for the median. Rank-based methods are attractive in that they retain a key feature of any successful genetic algorithm—selectivity based on relative fitness—while discouraging premature convergence and other difficulties caused by the actual form of $f$, which can be somewhat arbitrary [660]. Some less common fitness function formulations involving scaling and transforming $f$ are mentioned in [262].

***3.4.2.2 Selection Mechanisms and Updating Generations*** Previously, in Section 3.4.1.2, we mentioned only simple approaches to selecting parents on the basis of fitness. Selecting parents on the basis of fitness ranks (Section 3.4.2.1) is far more common than using selection probabilities proportional to fitness.

Another common approach is *tournament selection* [204, 263, 264]. In this approach, the set of chromosomes in generation $t$ is randomly partitioned into $k$ disjoint subsets of equal size (perhaps with a few remaining chromosomes temporarily ignored). The best individual in each group is chosen as a parent. Additional random partitionings are carried out until sufficient parents have been generated. Parents are then paired randomly for breeding. This approach ensures that the best individual will breed $P$ times, the median individual will breed once on average, and the worst individual will not breed at all. The approaches of proportional selection, ranking, and tournament selection apply increasing selective pressure, in that order. Higher selective pressure is generally associated with superior performance, as long as premature entrapment in local optima can be avoided [17].

Populations can be partially updated. The *generation gap*, $G$, is a proportion of the generation to be replaced by generated offspring [146]. Thus, $G = 1$ corresponds to a canonical genetic algorithm with distinct, nonoverlapping generations. At the other extreme, $G = 1/P$ corresponds to incremental updating of the population one offspring at a time. In this case, a *steady-state* genetic algorithm produces one offspring at a time to replace the least fit (or some random relatively unfit) individual [661]. Such a process typically exhibits more variance and higher selective pressure than a standard approach.

When $G < 1$, performance can sometimes be enhanced with a selection mechanism that departs somewhat from the Darwinian analogy. For example, an *elitist* strategy would place an exact copy of the current fittest individual in the next generation, thereby ensuring the survival of the best current solution [146]. When $G = 1/P$, each offspring could replace a chromosome randomly selected from those with below-average fitness [5].

Deterministic selection strategies have been proposed to eliminate sampling variability [19, 464]. We see no compelling need to eliminate the randomness inherent in the selection mechanism.

One important consideration when generating or updating a population is whether to allow duplicate individuals in the population. Dealing with duplicate individuals wastes computing resources, and it potentially distorts the parent selection criterion by giving duplicated chromosomes more chances to produce offspring [138].

***3.4.2.3 Genetic Operators and Permutation Chromosomes*** To increase genetic mixing, it is possible to choose more than one crossover point. If two crossover points are chosen, the gene sequence between them can be swapped between parents to create offspring. Such multipoint crossover can improve performance [54, 187].

Many other approaches for transferring genes from parents to offspring have been suggested. For example, each offspring gene could be filled with an allele randomly selected from the alleles expressed in that position in the parents. In this case, the parental origins of adjacent genes could be independent [4, 622] or correlated [602], with strength of correlation controlling the degree to which offspring resemble a single parent.

In some problems, a different allele alphabet may be more reasonable. Allele alphabets with many more than two elements have been investigated [13, 138, 524, 534]. For some problems, genetic algorithms using a floating-point alphabet have outperformed algorithms using the binary alphabet [138, 346, 463]. Methods known as messy genetic algorithms employ variable-length encoding with genetic operators that adapt to changing length [265–267]. Gray coding is another alternative encoding that is particularly useful for real-valued objective functions that have a bounded number of optima [662].

When a nonbinary allele alphabet is adopted, modifications to other aspects of the genetic algorithm, particularly to the genetic operators, is often necessary and even fruitful. Nowhere is this more evident than when permutation chromosomes are used. Recall that Section 3.4.1.3 introduced a special chromosome encoding for permutation optimization problems. For such problems (like the traveling salesman problem), it is natural to write a chromosome as a permutation of the integers $1, \ldots, n$. New genetic operators are needed then to ensure that each generation contains only valid permutation chromosomes.

For example, let $p = 9$, and consider the crossover operator. From two parent chromosomes 752631948 and 912386754 and a crossover point between the second and third loci, standard crossover would produce offspring 752386754 and 912631948. Both of these are invalid permutation chromosomes, because both contain some duplicate alleles.

A remedy is *order crossover* [623]. A random collection of loci is chosen, and the order in which the alleles in these loci appear in one parent is imposed on the same alleles in the other parent to produce one offspring. The roles of the parents can be switched to produce a second offspring. This operator attempts to respect relative positions of alleles. For example, consider the parents 752631948 and 912386754, and suppose that the fourth, sixth, and seventh loci are randomly chosen. In the first parent, the alleles in these loci are 6, 1, and 9. We must rearrange the 6, 1, and 9 alleles in the second parent to impose this order. The remaining alleles in the second parent are $**238*754$. Inserting 6, 1, and 9 in this order yields 612389754 as the offspring. Reversing the roles of the parents yields a second offspring 352671948.

Many other crossover operators for permutation chromosomes have been proposed [135, 136, 138, 268, 464, 492, 587]. Most are focused on the positions of individual genes. However, for problems like the traveling salesman problem, such operators have the undesirable tendency to destroy links between cities in the parent tours. The

**TABLE 3.3** Edge tables showing the cities linked to or from each allele in either parent for each of the first three steps of edge recombination crossover. Beneath each column is the offspring chromosome resulting from each step.

| Step 1 | | Step 2 | | Step 3 | |
|---|---|---|---|---|---|
| City | Links | City | Links | City | Links |
| 1 | 3, 9, 2 | 1 | 3, 2 | 1 | 3, 2 |
| 2 | 5, 6, 1, 3 | 2 | 5, 6, 1, 3 | 2 | 5, 6, 1, 3 |
| 3 | 6, 1, 2, 8 | 3 | 6, 1, 2, 8 | 3 | 6, 1, 2, 8 |
| 4 | 9, 8, 5 | 4 | 8, 5 | 4 | Used |
| 5 | 7, 2, 4 | 5 | 7, 2, 4 | 5 | 7, 2 |
| 6 | 2, 3, 8, 7 | 6 | 2, 3, 8, 7 | 6 | 2, 3, 8, 7 |
| 7 | 8, 5, 6 | 7 | 8, 5, 6 | 7 | 8, 5, 6 |
| 8 | 4, 7, 3, 6 | 8 | 4, 7, 3, 6 | 8 | 7, 3, 6 |
| 9 | 1, 4 | 9 | Used | 9 | Used |
| **9********** | | **94********** | | **945********* | |

desirability of a candidate solution is a direct function of these links. Breaking links is effectively an unintentional source of mutation. *Edge-recombination crossover* has been proposed to produce offspring that contain only links present in at least one parent [663, 664].

We use the traveling salesman problem to explain edge-recombination crossover. The operator proceeds through the following steps.

1. We first construct an edge table that stores all the links that lead into and out of each city in either parent. For our two parents, 752631948 and 912386754, the result is shown in the leftmost portion of Table 3.3. Note that the number of links into and out of each city in either parent will always be at least two and no more than four. Also, recall that a tour returns to its starting city, so, for example, the first parent justifies listing 7 as a link from 8.
2. To begin creating an offspring, we choose between the initial cities of the two parents. In our example, the choices are cities 7 and 9. If the parents' initial cities have the same number of links, then the choice is made randomly. Otherwise, choose the initial city from the parent whose initial city has fewer links. In our example, this yields 9********.
3. We must now link onward from allele 9. From the leftmost column of the edge table, we find that allele 9 has two links: 1 and 4. We want to chose between these by selecting the city with the fewest links. To do this, we first update the edge table by deleting all references to allele 9, yielding the center portion of Table 3.3. Since cities 1 and 4 both have two remaining links, we choose randomly between 1 and 4. If 4 is the choice, then the offspring is updated to 94********.
4. There are two possible links onward from city 4: cities 5 and 8. Updating the edge table to produce the rightmost portion of Table 3.3, we find that city 5 has

the fewest remaining links. Therefore, we choose city 5. The partial offspring is now 945******.

Continuing this process might yield the offspring 945786312 by the following steps: select 7; select 8; select 6; randomly select 3 from the choices of 2 and 3; randomly select 1 from the choices of 1 and 2; select 2.

Note that in each step a city is chosen among those with the fewest links. If, instead, links were chosen uniformly at random, cities would be more likely to be left without a continuing edge. Since tours are circuital, the preference for a city with few links does not introduce any sort of bias in offspring generation.

An alternative *edge assembly* strategy has been found to be extremely effective in some problems [477].

Mutation of permutation chromosomes is not as difficult as crossover. A simple mutation operator is to randomly exchange two genes in the chromosome [531]. Alternatively, the elements in a short random segment of the chromosome can be randomly permuted [138].

### 3.4.3 Initialization and Parameter Values

Although traditionally a genetic algorithm is initiated with a generation of purely random individuals, heuristic approaches to constructing individuals with good or diverse fitness have been suggested as an improvement on random starts [138, 531].

Equal sizes for subsequent generations are not required. Population fitness usually improves very rapidly during the early generations of a genetic algorithm. In order to discourage premature convergence and promote search diversity, it may be desirable to use a somewhat large generation size $P$ for early generations. If $P$ is fixed at too large a value, however, the entire algorithm may be too slow for practical use. Once the algorithm has made significant progress toward the optimum, important improving moves most often come from high-quality individuals; low-quality individuals are increasingly marginalized. Therefore, it has been suggested that $P$ may be decreased progressively as iterations continue [677]. However, rank-based selection mechanisms are more commonly employed as an effective way to slow convergence.

It can be also useful to allow a variable mutation rate that is inversely proportional to the population diversity [531]. This provides a stimulus to promote search diversity as generations become less diverse. Several authors suggest other methods for allowing the probabilities of mutation and crossover and other parameters of the genetic algorithm to vary adaptively over time in manners that may encourage search diversity [54, 137, 138, 464].

### 3.4.4 Convergence

The convergence properties of genetic algorithms are beyond the scope of this chapter, but several important ideas are worth mentioning.

Much of the early analysis about why genetic algorithms work was based on the notion of schemata [262, 333]. Such work is based on a canonical genetic algorithm with binary chromosome encoding, selection of each parent with probability proportional to fitness, simple crossover applied every time parents are paired, and mutation

randomly applied to each gene independently with probability $\mu$. For this setting, the *schema theorem* provides a lower bound on the expected number of instances of a schema in generation $t + 1$, given that it was present in generation $t$.

The schema theorem shows that a short, low-order schema (i.e., one specifying only a few nearby alleles) will enjoy increased expected representation in the next generation if the average fitness of chromosomes containing that schema in the generation at time $t$ exceeds the average fitness of all chromosomes in the generation. A longer and/or more complex schema will require higher relative fitness to have the same expectation. Proponents of schema theory argue that convergence to globally competitive candidate solutions can be explained by how genetic algorithms simultaneously juxtapose many short low-order schemata of potentially high fitness, thereby promoting propagation of advantageous schemata.

More recently, the schema theorem and convergence arguments based upon it have become more controversial. Traditional emphasis on the number of instances of a schema that propagate to the next generation and on the average fitness of chromosomes containing that schema is somewhat misguided. What matters far more is which particular chromosomes containing that schema are propagated. Further, the schema theorem overemphasizes the importance of schemata: in fact it applies equally well to any arbitrary subsets of $\mathbf{\Theta}$. Finally, the notion that genetic algorithms succeed because they implicitly simultaneously allocate search effort to many schemata-defined regions of $\mathbf{\Theta}$ has been substantially discredited [647]. An authoritative exposition of the mathematical theory of genetic algorithms is given by Vose [646]. Other helpful treatments include [200, 533].

Genetic algorithms are not the only optimization strategy that can be motivated by analogy to a complex biological system. For example, *particle swarm* optimization also creates and updates a population of candidate solutions [372, 373, 594]. The locations of these solutions within the search space evolve through simple rules that can be viewed as reflecting cooperation and competition between individuals analogous to the movement of birds in a flock. Over a sequence of iterations, each individual adjusts its location (i.e., candidate solution) based on its own flying experience and those of its companions.

## 3.5 TABU ALGORITHMS

A tabu algorithm is a local search algorithm with a set of additional rules that guide the selection of moves in ways that are believed to promote the discovery of a global maximum. The approach employs variable neighborhoods: The rules for identifying acceptable steps change at each iteration. Detailed studies of tabu methods include [254, 255, 257–259].

In a standard ascent algorithm, entrapment in a globally uncompetitive local maximum is likely, because no downhill moves are allowed. Tabu search allows downhill moves when no uphill move can be found in the current neighborhood (and possibly in other situations too), thereby potentially escaping entrapment. An early form of tabu search, called steepest ascent/mildest descent, moved to the least unfavorable neighbor when there was no uphill move [306].

**TABLE 3.4** Examples of attributes. The left column gives examples in a generic context. The right column gives corresponding attributes in the specific context of 2-change neighborhoods in a regression model selection problem.

| Attribute | Model Selection Example |
|---|---|
| A change in the value of $\theta_i^{(t)}$. The attribute may be the value from which the move began, or the value at which it arrived. | $A_1$: Whether the $i$th predictor is added (or deleted) from the model. |
| A swap in the values of $\theta_i^{(t)}$ and $\theta_j^{(t)}$ when $\theta_i^{(t)} \neq \theta_j^{(t)}$. | $A_2$: Whether the absent variable is exchanged for the variable present in the model. |
| A change in the value of $f$ resulting from the step, $f(\boldsymbol{\theta}^{(t+1)}) - f(\boldsymbol{\theta}^{(t)})$. | $A_3$: The reduction in AIC achieved by the move. |
| The value $g(\boldsymbol{\theta}^{(t+1)})$ of some other strategically chosen function $g$. | $A_4$: The number of predictors in the new model. |
| A change in the value of $g$ resulting from the step, $g(\boldsymbol{\theta}^{(t+1)}) - g(\boldsymbol{\theta}^{(t)})$. | $A_5$: A change to a different variable selection criterion such as Mallows's $C_p$ [435] or the adjusted $R^2$ [483]. |

If a downhill step is chosen, care must be taken to ensure that the next step (or a future one) does not simply reverse the downhill move. Such cycling would eliminate the potential long-term benefit of the downhill move. To prevent such cycling, certain moves are temporarily forbidden, or made *tabu*, based on the recent history of the algorithm.

There are four general types of rules added to local search by tabu search methods. The first is to make certain potential moves temporarily tabu. The others involve *aspiration* to better solutions, *intensification* of search in promising areas of solution space, and *diversification* of search candidates to promote broader exploration of the solution space. These terms will be defined after we discuss tabus.

### 3.5.1 Basic Definitions

Tabu search is an iterative algorithm initiated at time $t = 0$ with a candidate solution $\boldsymbol{\theta}^{(0)}$. At the $t$th iteration, a new candidate solution is selected from a neighborhood of $\boldsymbol{\theta}^{(t)}$. This candidate becomes $\boldsymbol{\theta}^{(t+1)}$. Let $H^{(t)}$ denote the history of the algorithm through time $t$. It suffices for $H^{(t)}$ to be a selective history, remembering only certain matters necessary for the future operation of the algorithm.

Unlike simple local search, a tabu algorithm generates a neighborhood of the current candidate solution that depends on the search history; denote this by $\mathcal{N}(\boldsymbol{\theta}^{(t)}, H^{(t)})$. Furthermore, the identification of the preferred $\boldsymbol{\theta}^{(t+1)}$ in $\mathcal{N}(\boldsymbol{\theta}^{(t)}, H^{(t)})$ may depend not only on $f$ but also on the search history. Thus, we may assess neighbors using an augmented objective function, $f_{H^{(t)}}$.

A single step from $\boldsymbol{\theta}^{(t)}$ to $\boldsymbol{\theta}^{(t+1)}$ can be characterized by many *attributes*. Attributes will be used to describe moves or types of moves that will be forbidden, encouraged, or discouraged in future iterations of the algorithm. Examples of attributes are given in the left column of Table 3.4. Such attributes are not unique to tabu search; indeed, they can be used to characterize moves from any local search.

However, tabu search explicitly adapts the current neighborhood according to the attributes of recent moves.

The attributes in Table 3.4 can be illustrated by considering a regression model selection problem. Suppose $\theta_i^{(t)} = 1$ if the $i$th predictor is included in the model at time $t$, and 0 otherwise. Suppose that 2-change neighborhoods consist of all models to which two variables separately have each been added or deleted from the current model. The right column of Table 3.4 gives one example of each generic attribute listed, in the context of these 2-change neighborhoods in the regression model selection problem from Example 3.2. These examples are labeled $A_1$ through $A_5$. Many other effective attributes can be identified from the context of specific optimization problems.

Denote the $a$th attribute as $A_a$. Note that the complement (i.e., negation) of an attribute is also an attribute, so if $A_a$ corresponds to swapping the values of $\theta_i^{(t)}$ and $\theta_j^{(t+1)}$, then $\overline{A}_a$ corresponds to not making that swap.

As the algorithm progresses, the attributes of the $t$th move will vary with $t$, and the quality of the candidate solution will also vary. Future moves can be guided by the history of past moves, their objective function values, and their attributes. The *recency* of an attribute is the number of steps that have passed since a move most recently had that attribute. Let $R\left(A_a, H^{(t)}\right) = 0$ if the $a$th attribute is expressed in the move yielding $\boldsymbol{\theta}^{(t)}$, let $R\left(A_a, H^{(t)}\right) = 1$ if it is most recently expressed in the move yielding $\boldsymbol{\theta}^{(t-1)}$, and so forth.

### 3.5.2 The Tabu List

When considering a move from $\boldsymbol{\theta}^{(t)}$, we compute the increase in the objective function achieved for each neighbor of $\boldsymbol{\theta}^{(t)}$. Ordinarily, the neighbor that provides the greatest increase would be adopted as $\boldsymbol{\theta}^{(t+1)}$. This corresponds to the steepest ascent.

Suppose, however, that no neighbor of $\boldsymbol{\theta}^{(t)}$ yields an increased objective function. Then $\boldsymbol{\theta}^{(t+1)}$ is ordinarily chosen to be the neighbor that provides the smallest decrease. This is the mildest descent.

If only these two rules were used for search, the algorithm would quickly become trapped and converge to a local maximum. After one move of mildest descent, the next move would return to the hilltop just departed. Cycling would ensue.

To avoid such cycling, a *tabu list* of temporarily forbidden moves is incorporated in the algorithm. Each time a move with attribute $A_a$ is taken, $\overline{A}_a$ is put on a tabu list for $\tau$ iterations. When $R\left(A_a, H^{(t)}\right)$ first equals $\tau$, the tabu expires and $\overline{A}_a$ is removed from the tabu list. Thus, moves with attributes on the tabu list are effectively excluded from the current neighborhood. The modified neighborhood is denoted

$$\mathcal{N}(\boldsymbol{\theta}^{(t)}, H^{(t)}) = \left\{\boldsymbol{\theta} : \ \boldsymbol{\theta} \in \mathcal{N}(\boldsymbol{\theta}^{(t)}) \quad \text{and no attribute of } \boldsymbol{\theta} \text{ is currently tabu}\right\}. \tag{3.10}$$

This prevents undoing the change for $\tau$ iterations, thereby discouraging cycling. By the time that the tabu has expired, enough other aspects of the candidate solution

should have changed that reversing the move may no longer be counterproductive. Note that the tabu list is a list of attributes, not moves, so a single tabu attribute may forbid entire classes of moves.

The *tabu tenure*, $\tau$, is the number of iterations over which an attribute is tabu. This can be a fixed number or it may vary, systematically or randomly, perhaps based on features of the attribute. For a given problem, a well-chosen tabu tenure will be long enough to prevent cycling and short enough to prevent the deterioration of candidate solution quality that occurs when too many moves are forbidden. Fixed tabu tenures between 7 and 20, or between $0.5\sqrt{p}$ and $2\sqrt{p}$, where $p$ is the size of the problem, have been suggested for various problem types [257]. Tabu tenures that vary dynamically seem more effective in many problems [259]. Also, it will often be important to use different tenures for different attributes. If an attribute contributes tabu restrictions for a wide variety of moves, the corresponding tabu tenure should be short to ensure that future choices are not limited.

**Example 3.6 (Genetic Mapping, Continued)** We illustrate some uses of tabus, using the gene mapping problem introduced in Example 3.1.

First, consider monitoring the swap attribute. Suppose that $A_a$ is the swap attribute corresponding to exchanging two particular loci along the chromosome. When a move $A_a$ is taken, it is counterproductive to immediately undo the swap, so $\overline{A}_a$ is placed on the tabu list. Search progresses only among moves that do not reverse recent swaps. Such a tabu promotes search diversity by avoiding quick returns to recently searched areas.

Second, consider the attribute identifying the locus label $\theta_j$ for which $\hat{d}(\theta_j, \theta_{j+1})$ is smallest in the new move. In other words, this attribute identifies the two loci in the new chromosome that are nearest each other. If the complement of this attribute is put on the tabu list, any move to a chromosome for which other loci are closer will be forbidden moves for $\tau$ iterations. Such a tabu promotes search intensity among genetic maps for which loci $\theta_j$ and $\theta_j$ are closest.

Sometimes, it may be reasonable to place the attribute itself, rather than its complement, on the tabu list. For example, let $h(\boldsymbol{\theta})$ compute the mean $\hat{d}(\theta_j, \theta_{j+1})$ between adjacent loci in a chromosome ordered by $\boldsymbol{\theta}$. Let $A_a$ be the attribute indicating excessive change of the mean conditional MLE map distance, so $A_a$ equals 1 if $\left|h(\boldsymbol{\theta}^{(t+1)}) - h(\boldsymbol{\theta}^{(t)})\right| > c$ and 0 otherwise, for some fixed threshold $c$. If a move with mean change greater than $c$ is taken, we may place $A_a$ itself on the tabu list for $\tau$ iterations. This prevents any other drastic mean changes for a period of time, allowing better exploration of the newly entered region of solution space before moving far away. □

### 3.5.3 Aspiration Criteria

Sometimes, choosing not to move to a nearby candidate solution because the move is currently tabu can be a poor decision. In these cases, we need a mechanism to override the tabu list. Such a mechanism is called an *aspiration criterion*.

A simple and popular aspiration criterion is to permit a tabu move if it provides a higher value of the objective function than has been found in any iteration so far. Clearly it makes no sense to overlook the best solution found so far, even if it is currently tabu. One can easily envision scenarios where this aspiration criterion is useful. For example, suppose that a swap of two components of $\boldsymbol{\theta}$ is on the tabu list and the candidate solutions at each iteration recently have drifted away from the region of solution space being explored when the tabu began. The search will now be in a new region of solution space where it is quite possible that reversing the tabu swap would lead to a drastic increase in the objective function.

Another interesting option is *aspiration by influence*. A move or attribute is influential if it is associated with a large change in the value of the objective function. There are many ways to make this idea concrete [257]. To avoid unnecessary detail about numerous specific possibilities, let us simply denote the influence of the $a$th attribute as $I\left(A_a, H^{(t)}\right)$ for a move yielding $\boldsymbol{\theta}^{(t)}$. In many combinatorial problems, there are a lot of neighboring moves that cause only small incremental changes to the value of the objective function, while there are a few moves that cause major shifts. Knowing the attributes of such moves can help guide search. Aspiration by influence overrides the tabu on reversing a low-influence move if a high-influence move is made prior to the reversal. The rationale for this is that the recent high-influence step may have moved the search to a new region of the solution space where further local exploration is useful. The reversal of the low-influence move will probably not induce cycling, since the intervening high-influence move likely shifted scrutiny to a portion of solution space more distant than what could be reached by the low-influence reversal.

Aspiration criteria can also be used to encourage moves that are not tabu. For example, when low-influence moves appear to provide only negligible improvement in the objective function, they can be downweighted and high-influence moves can be given preference. There are several ways to do this; one approach is to incorporate in $f_{H^{(t)}}$ either a penalty or an incentive term that depends on the relative influence of candidate moves.

### 3.5.4 Diversification

An important component of any search is to ensure that the search is broad enough. Rules based on how often attributes are observed during search can be used to increase the diversity of candidate solutions examined during tabu search.

The *frequency* of an attribute records the number of moves that manifested that attribute since the search began. Let $C(A_a, H^{(t)})$ represent the count of occurrences of the $a$th attribute thus far. Then $F(A_a, H^{(t)})$ represents a frequency function that can be used to penalize moves that are repeated too frequently. The most direct definition is $F(A_a, H^{(t)}) = C(A_a, H^{(t)})/t$, but the denominator may be replaced by the sum, the maximum, or the average of the counts of occurrences of various attributes.

Suppose the frequency of each attribute is recorded, either over the entire history or over the most recent $\psi$ moves. Note that this frequency may be one of two types, depending on the attribute considered. If the attribute corresponds to some feature

of $\boldsymbol{\theta}^{(t)}$, then the frequency measures how often that feature is seen in candidate solutions considered during search. Such frequencies are termed *residence frequencies*. If, alternatively, the attribute corresponds to some change induced by moving from one candidate solution to another, then the frequency is a *transition frequency*. For example, in the regression model selection problem introduced in Example 3.2, the attribute noting the inclusion of the predictor $x_i$ in the model would have a residence frequency. The attribute that signaled when a move reduced the AIC would have a transition frequency.

If attribute $A_a$ has a high residence frequency and the history of the most recent $\psi$ moves covers nearly optimal regions of solution space, this may suggest that $A_a$ is associated with high-quality solutions. On the other hand, if the recent history reflects the search getting stuck in a low-quality region of solution space, then a high residence frequency may suggest that the attribute is associated with bad solutions. Usually, $\psi > \tau$ is an intermediate or long-term memory parameter that allows the accumulation of additional historical information to diversify future search.

If attribute $A_a$ has a high transition frequency, this attribute may be what has been termed a crack filler. Such an attribute may be frequently visited during the search in order to fine-tune good solutions but rarely offers fundamental improvement or change [257]. In this case, the attribute has low influence.

A direct approach employing frequency to increase search diversification is to incorporate a penalty or incentive function in $f_{H^{(t)}}$. The choice

$$f_{H^{(t)}}(\boldsymbol{\theta}^{(t+1)}) = \begin{cases} f(\boldsymbol{\theta}^{(t+1)}) & \text{if } f(\boldsymbol{\theta}^{(t+1)}) \geq f(\boldsymbol{\theta}^{(t)}), \\ f(\boldsymbol{\theta}^{(t+1)}) - cF\left(A_a, H^{(t)}\right) & \text{if } f(\boldsymbol{\theta}^{(t+1)}) < f(\boldsymbol{\theta}^{(t)}) \end{cases} \tag{3.11}$$

with $c > 0$ has been suggested [566]. If all nontabu moves are downhill, then this approach discourages moves that have the high-frequency attribute $A_a$. An analogous strategy can be crafted to diversify the selection of uphill moves.

Instead of incorporating a penalty or incentive in the objective function, it is possible to employ a notion of graduated tabu status, where an attribute may be only partially tabu. One way to create a tabu status that varies by degrees is to invoke probabilistic tabu decisions: An attribute can be assigned a probability of being tabu, where the probability is adjusted according to various factors, including the tabu tenure [257].

### 3.5.5 Intensification

In some searches it may be useful to intensify the search effort in particular areas of solution space. Frequencies can also be used to guide such intensification. Suppose that the frequencies of attributes are tabulated over the most recent $\upsilon$ moves, and a corresponding record of objective function values is kept. By examining these data, key attributes shared by good candidate solutions can be identified. Then moves that retain such features can be rewarded and moves that remove such features can be penalized through $f_{H^{(t)}}$. The time span $\upsilon > \tau$ parameterizes the length of a long-term memory to enable search intensification in promising areas of solution space.

### 3.5.6 Comprehensive Tabu Algorithm

Below we summarize a fairly general tabu algorithm that incorporates many of the features described above. After initialization and identification of a list of problem-specific attributes, the algorithm proceeds as follows:

1. Determine an augmented objective function $f_{H^{(t)}}$ that depends on $f$ and perhaps on
   a. frequency-based penalties or incentives to promote diversification, and/or
   b. frequency-based penalties or incentives to promote intensification.
2. Identify neighbors of $\boldsymbol{\theta}^{(t)}$, namely the members of $\mathcal{N}(\boldsymbol{\theta}^{(t)})$.
3. Rank the neighbors in decreasing order of improvement, as evaluated by $f_{H^{(t)}}$.
4. Select the highest ranking neighbor.
5. Is this neighbor currently on the tabu list? If not, go to step 8.
6. Does this neighbor pass an aspiration criterion? If so, go to step 8.
7. If all neighbors of $\boldsymbol{\theta}^{(t)}$ have been considered and none have been adopted as $\boldsymbol{\theta}^{(t+1)}$, then stop. Otherwise, select the next most high-ranking neighbor and go to step 5.
8. Adopt this solution as $\boldsymbol{\theta}^{(t+1)}$.
9. Update the tabu list by creating new tabus based on the current move and by deleting tabus whose tenures have expired.
10. Has a stopping criterion been met? If so, stop. Otherwise, increment $t$ and go to step 1.

It is sensible to stop when a maximum number of iterations has been reached, and then to take the best candidate solution yet found as the final result. Search effort can be split among a collection of random starts rather than devoting all resources to one run from a single start. By casting tabu search in a Markov chain framework, it is possible to obtain results on the limiting convergence of the approach [191].

**Example 3.7 (Baseball Salaries, Continued)** A simple tabu search was applied to the variable selection problem for regression modeling of the baseball data introduced in Example 3.3. Only attributes signaling the presence or absence of each predictor were monitored. Moves that would reverse the inclusion or removal of a predictor were made tabu for $\tau = 5$ moves, and the algorithm was run for 75 moves from a random start. The aspiration criterion permitted an otherwise tabu move if it yielded an objective function value above the best previously seen.

Figure 3.7 shows the values of the AIC for the sequence of candidate solutions generated by this tabu search. The AIC was quickly improved, and an optimum value of $-418.95$, derived from the model using predictors 2, 3, 6, 8, 10, 13, 14, 15, 16, 24, 25, and 26, was found on two occasions: iterations 29 and 43. This solution matches the best model found using random starts local search (Table 3.2). □

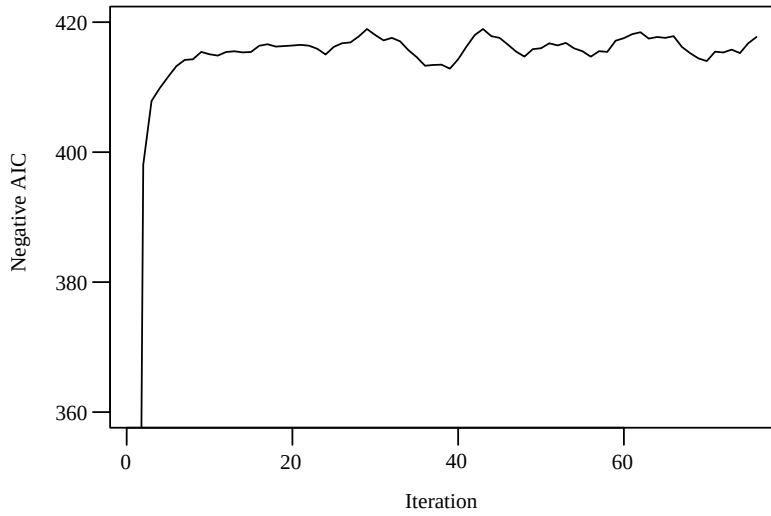


**FIGURE 3.7** Results of tabu search for Example 3.7. Only AIC values between −360 and −420 are shown.

## PROBLEMS

The baseball data introduced in Section 3.3 are available from the website for this book. Problems 3.1–3.4 explore the implications of various algorithm configurations. Treat these problems in the spirit of experiments, trying to identify settings where interesting differences can be observed. Increase the run lengths from those used above to suit the speed of your computer, and limit the total number of objective function evaluations in every run (effectively the search effort) to a fixed number so that different algorithms and configurations can be compared fairly. Summarize your comparisons and conclusions. Supplement your comments with graphs to illustrate key points.

**3.1.** Implement a random starts local search algorithm for minimizing the AIC for the baseball salary regression problem. Model your algorithm after Example 3.3.

- **a.** Change the move strategy from steepest descent to immediate adoption of the first randomly selected downhill neighbor.
- **b.** Change the algorithm to employ 2-neighborhoods, and compare the results with those of previous runs.

**3.2.** Implement a tabu algorithm for minimizing the AIC for the baseball salary regression problem. Model your algorithm after Example 3.7.

- **a.** Compare the effect of using different tabu tenures.
- **b.** Monitor changes in AIC from one move to the next. Define a new attribute that signals when the AIC change exceeds some value. Allow this attribute to be included on the tabu list, to promote search diversity.
- **c.** Implement aspiration by influence, overriding the tabu of reversing a low-influence move if a high-influence move is made prior to the reversal. Measure influence with changes in $R^2$.

**3.3.** Implement simulated annealing for minimizing the AIC for the baseball salary regression problem. Model your algorithm on Example 3.4.

**a.** Compare the effects of different cooling schedules (different temperatures and different durations at each temperature).

**b.** Compare the effect of a proposal distribution that is discrete uniform over 2-neighborhoods versus one that is discrete uniform over 3-neighborhoods.

**3.4.** Implement a genetic algorithm for minimizing the AIC for the baseball salary regression problem. Model your algorithm on Example 3.5.

**a.** Compare the effects of using different mutation rates.

**b.** Compare the effects of using different generation sizes.

**c.** Instead of the selection mechanism used in Example 3.5, try the following three mechanisms:

**i.** Independent selection of one parent with probability proportional to fitness and the other completely at random

**ii.** Independent selection of each parent with probability proportional to fitness

**iii.** Tournament selection with $P/5$ strata, and/or another number of strata that you prefer

To implement some of these approaches, you may need to scale the fitness function. For example, consider the scaled fitness functions $\pi$ given by

$$\phi(\boldsymbol{\vartheta}_i^{(t)}) = af(\boldsymbol{\theta}_i^{(t)}) + b, \tag{3.12}$$

$$\phi(\boldsymbol{\vartheta}_i^{(t)}) = f(\boldsymbol{\theta}_i^{(t)}) - (\overline{f} - zs), \tag{3.13}$$

or

$$\phi(\boldsymbol{\vartheta}_i^{(t)}) = f(\boldsymbol{\theta}_i^{(t)})^v, \tag{3.14}$$

where $a$ and $b$ are chosen so that the mean fitness equals the mean objective function value and the maximum fitness is a user-chosen $c$ times greater than the mean fitness, $\overline{f}$ is the mean and $s$ is the standard deviation of the unscaled objective function values in the current generation, $z$ is a number generally chosen between 1 and 3, and $v$ is a number slightly larger than 1. Some scalings can sometimes produce negative values for $\boldsymbol{\vartheta}_i^{(t)}$. In such situations, we may apply the transformation

$$\phi_{\text{new}}(\boldsymbol{\vartheta}_i^{(t)}) = \begin{cases} \phi(\boldsymbol{\vartheta}_i^{(t)}) + d^{(t)} & \text{if } \phi(\boldsymbol{\vartheta}_i^{(t)}) + d^{(t)} > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{3.15}$$

where $d^{(t)}$ is the absolute value of the fitness of the worst chromosome in generation $t$, in the last $k$ generations for some $k$, or in all preceding generations. Each of these scaling approaches has the capacity to dampen the variation in $f$, thereby retaining within-generation diversity and increasing the potential to find the global optimum.

Compare and comment on the results for your chosen methods.

**d.** Apply a steady-state genetic algorithm, with the generation gap $G = 1/P$. Compare with the canonical option of distinct, nonoverlapping generations.

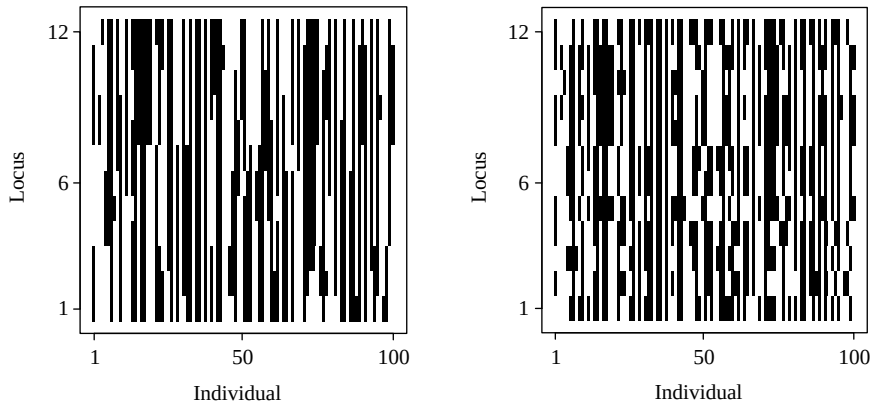


**FIGURE 3.8** Chromosomes for Problem 3.5. Simulated data on 12 loci are available for 100 individuals. For each locus, the source chromosome from the heterozygous parent is encoded in black or white, analogously to Figure 3.1 in Example 3.1. The left panel shows the data arranged according to the true locus ordering, whereas the right panel shows the data arranged by locus label as they would be recorded during data collection.

**e.** Implement the following crossover approach, termed *uniform crossover* [622]: Each locus in the offspring is filled with an allele independently selected at random from the alleles expressed in that position in the parents.

**3.5.** Consider the genetic mapping example introduced in Example 3.1. Figure 3.8 shows some data for 100 simulated data sequences for a chromosome of length 12. The left panel of this figure shows the data under the true genetic map ordering, and the right panel shows the actual data, with the ordering unknown to the analyst. The data are available from the website for this book.

**a.** Apply a random starts local search approach to estimate the genetic map (i.e., the ordering and the genetic distances). Let neighborhoods consist of 20 orderings that differ from the current ordering by randomly swapping the placement of two alleles. Move to the best candidate in the neighborhood, thereby taking a random descent step. Begin with a small number of starts of limited length, to gauge the computational difficulty of the problem; then report the best results you obtained within reasonable limits on the computational burden. Comment on your results, the performance of the algorithm, and ideas for improved search. [Hint: Note that the orderings $(\theta_{j_1}, \theta_{j_2}, \ldots, \theta_{j_{12}})$ and $(\theta_{j_{12}}, \theta_{j_{11}}, \ldots, \theta_{j_1})$ represent identical chromosomes read from either end.]

**b.** Apply an algorithm for random starts local search via steepest descent to estimate the genetic map. Comment on your results and the performance of the algorithm. This problem is computationally demanding and may require a fast computer.

**3.6.** Consider the genetic mapping data described in Problem 3.5.

**a.** Apply a genetic algorithm to estimate the genetic map (i.e., the ordering and the genetic distances). Use the order crossover method. Begin with a small run to gauge the computational difficulty of the problem, then report your results for a run

using reasonable limits on the computational burden. Comment on your results, the performance of the algorithm, and ideas for improved search.

**b.** Compare the speed of fitness improvements achieved with the order crossover and the edge-recombination crossover strategies.

**c.** Attempt any other heuristic search method for these data. Describe your implementation, its speed, and the results.

**3.7.** The website for this book also includes a second synthetic dataset for a genetic mapping problem. For these data, there are 30 chromosomes. Attempt one or more heuristic search methods for these data. Describe your implementation, the results, and the nature of any problems you encounter. The true ordering used to simulate the data is also given for this dataset. Although the true ordering may not be the MLE, how close is your best ordering to the true ordering? How much larger is this problem than the one examined in the previous problem?

**3.8.** Thirteen chemical measurements were carried out on each of 178 wines from three regions of Italy [53]. These data are available from the website for this book. Using one or more heuristic search methods from this chapter, partition the wines into three groups for which the total of the within-group sum of squares is minimal. Comment on your work and the results. This is a search problem of size $3^p$ where $p = 178$. If you have access to standard cluster analysis routines, check your results using a standard method like that of Hartigan and Wong [317].