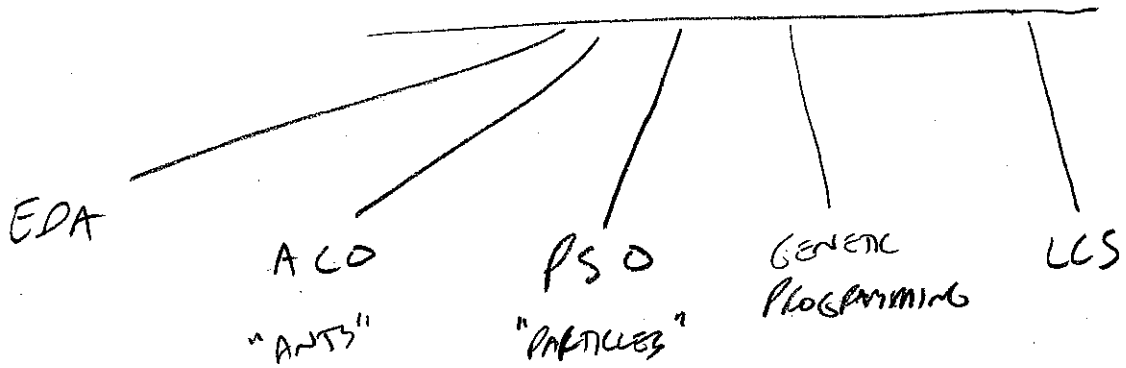
NEAL'S VIEW OF EVOLUTIONARY FAMILY OF ~~ALGORITHMS~~ ALGORITHMS

- ANT SYSTEMS ARE PART OF THE FAMILY
- ALL THESE ALGORITHMS BUILD "MODELS"
  - PROB. DISTRIBUTION OF POPULATION OF SOLUTIONS
  - SELECT FROM MODEL
    - ALTER INDIVIDUAL
    - TEST FITNESS
    - UPDATE THE MODEL

- FAMILY TREE "FROM THEORY PERSPECTIVE"
  - GAs HOLLAND, [illegible], OTHERS
  - EVOLUTIONARY STRATEGIES RECHENBERG SCHWEFEL
  - EVOLUTIONARY PROGRAMMING FOGEL



ALL OF THESE ALGORITHMS CAN BE MODELED WITH THE "VOSE" INFINITE POPULATION MODEL

- MICHAEL VOSE, ALDEN WRIGHT @ U OF MONTANA MAJOR CONTRIBUTORS

1. Are there any absorbing states?
2. If so, how long does it take to reach them?
3. If not, is the transition matrix primitive (and if so, what is its limiting distribution)?

We shall now apply these ideas to the study of genetic algorithms.

5.2 Limiting Distribution of the Simple GA

The first step in analysing a genetic algorithm as a stochastic process is to define the set of possible states that the algorithm can be in. We then need to confirm that the random sequence of states generated by a run of the algorithm does indeed form a Markov chain. That is, we need to check that the probability that the GA is in a particular state at a particular time depends at most on the state at the previous time step.

The natural candidate for describing the state of a GA is the population. At each generation we have a particular population, and the contents of that population depend (stochastically) on the previous generation. With this choice of state, we do indeed have a Markov chain. If the size of the search space is $|\mathcal{X}| = n$, and the population contains N individuals from the search space (possibly with duplications—remember that in general it is a multiset), then the number of possible states is:

$$\binom{n+N-1}{N}.$$

(This can be proven by induction, or by a combinatorial argument [182]). To keep things simple, we identify the search space $\mathcal{X}$ with the set of integers $\{0, 1, 2, \ldots, n-1\}$. Then we can represent populations as vectors:

$$v = (v_0, v_1, \ldots v_{n-1})$$

in which v_k is the number of copies of individual $k \in \mathcal{X}$ in the population. Clearly

$$\sum_{k=0}^{n-1} v_k = N.$$

The states are all the possible non-negative integer vectors with this property. As we have remarked in Chapter 2, the fitness function can be thought

5.2.1 Effect of selection

Having defined the states of the GA Markov chain, we can calculate the transition matrix. It is easiest if we break this down into three stages—selection, mutation and crossover—so first we consider a GA that only employs selection. Such an algorithm introduces no new search space elements into the population. What would we expect to happen in this situation? Assuming that selection always tends to preserve the better elements of the population, and throw out the worse, we would expect that eventually the population would contain only copies of the best individual in the initial population. We can easily formalise this using the Markov chain framework.

Suppose our current population is v and we apply selection. We need to calculate the probability distribution over all possible next populations, as a function of v to give us our transition matrix. If we use fitness-proportional selection, the probability of any individual $i \in \mathcal{X}$ being selected is

$$\mathbf{Pr}[i|v]_1 = \frac{v_i f(i)}{\sum_{j \in \mathcal{X}} v_j f(j)}$$

where the subscript 1 indicates that we have a one-operator GA. To obtain the next generation, we take N samples of this distribution over $\mathcal{X}$. This is called a *multinomial* distribution. The probability of generating a population u in this way is

$$\mathbf{Pr}[u|v] = N! \prod_{i \in \mathcal{X}} \frac{\mathbf{Pr}[i|v]_1^{u_i}}{u_i!}$$

and this formula gives us our selection-only transition matrix. We want to know if this Markov chain has any absorbing states. Our guess was that *uniform* populations (those containing N copies of a single individual) would be absorbing. To check this, suppose that v is a population containing only copies of $k \in \mathcal{X}$. That means $v_k = N$ and $v_i = 0$ for all $i \neq k$. Then with fitness-proportional selection, we have

$$\mathbf{Pr}[i|v]_1 = [i = k]$$

and therefore

$$\mathbf{Pr}[v|v] = 1.$$

(Note again the use of the indicator function notation $[expr]$ as introduced

Now, recall that having a 1 on the diagonal of the transition matrix corresponds to an absorbing state. This proves that all uniform populations are absorbing states of the selection-only genetic algorithm. Notice that the same is true for *any* selection method for which $\mathbf{Pr}[i|v] = [i = k]$ (i.e., where population v contains only copies of individual k). Since no selection method introduces new individuals into the population, then the conclusion holds.

5.2.2 Effect of mutation

Mutation involves changing one element of the search space to another, with a certain probability. Denote the probability that $j \in \mathcal{X}$ mutates to i by $U_{i,j}$. This gives us a $n \times n$ matrix U. The probability of generating an individual $i \in \mathcal{X}$ after selection, then mutation is

$$\mathbf{Pr}[i|v]_2 = \sum_{j \in \mathcal{X}} U_{i,j} \mathbf{Pr}[j|v]_1$$

and therefore the two-operator transition matrix is given by

$$\mathbf{Pr}[u|v] = N! \prod_{i \in \mathcal{X}} \frac{\mathbf{Pr}[i|v]_2^{u_i}}{u_i!}$$

Now suppose mutation is defined in such a way that $U_{i,j} > 0$ for all $i, j \in \mathcal{X}$. This would be the case, for example, with bitwise mutation acting on fixed-length binary strings. Then it is easy to see that $\mathbf{Pr}[i|v]_2 > 0$ for all $i \in \mathcal{X}$, and for any population v. That is, given any population v, there is a non-zero probability that the next generation will contain $i \in \mathcal{X}$, for any individual i. From this observation, we can conclude that the two-operator transition matrix contains no zero terms—and therefore contains no absorbing states. The matrix is primitive, so that (applying Theorem 5.1) there is a limiting distribution over the set of possible populations and, in the long-term, there is a non-zero probability that any given population might arise. The GA with selection and mutation does not 'converge'. It will wander through every possible population. However, some populations may be much more likely to occur than others, and we shall consider this possibility later.

5.2.3 Effect of crossover

The addition of a crossover operator will not change this conclusion. As long

matrix will be primitive. Suppose we denote the probability that i and j cross to form k by $r(i, j, k)$. Then the probability of generating individual k from population v in the three-operator GA is:

$$\mathbf{Pr}[k|v]_3 = \sum_{i,j \in \mathcal{X}} r(i, j, k) \mathbf{Pr}[i|v]_2 \mathbf{Pr}[j|v]_2$$

This probability will be non-zero provided that, for any $k \in \mathcal{X}$ there exists at least one pair i, j such that $r(i, j, k) > 0$. For most crossovers this will be true, since at least $r(k, k, k) > 0$ for each k. The transition matrix is now given by:

$$\mathbf{Pr}[u|v] = N! \prod_{i \in \mathcal{X}} \frac{\mathbf{Pr}[i|v]_3^{u_i}}{u_i!}$$

which is therefore primitive. Note that, if we assume crossover takes place before mutation instead of afterwards, it is easy to verify that we still obtain a primitive transition matrix.

When we have mutation in a GA (with or without crossover), we know that the long-term behaviour of the algorithm is described by a limiting distribution over the possible states. This distribution is a vector q, in which q_v is the probability associated with population v. The following theorem gives us a formula for q.

Theorem 5.4 (Davis-Principe [50]) *Suppose Q is the primitive transition matrix for a genetic algorithm with non-zero mutation. Then the limiting distribution is given by*

$$q_v = \frac{|Q_v - I|}{\sum_u |Q_u - I|}$$

where the matrix Q_u is derived from Q by replacing the uth column of Q with zeros.

5.2.4 An example

We can illustrate this result by a simple example. Suppose our search space is $\{00, 01, 10, 11\}$ which we identify with $\mathcal{X} = \{0, 1, 2, 3\}$ by considering each string to be a binary number. The fitness function is:

$$\begin{aligned} f(00) &= 1 \\ f(01) &= 2 \\ f(10) &= 3 \end{aligned}$$

ulation, and they can represent a probability distribution over individuals in the search space. We define the *generational operator*

$$\mathcal{G} : \Lambda \longrightarrow \Lambda$$

by $\mathcal{G}(\boldsymbol{p}) = \boldsymbol{p}'$ where $\boldsymbol{p}'$ is the probability distribution that is sampled to give the next population after $\boldsymbol{p}$. As described previously, $\mathcal{G}(\boldsymbol{p})$ is sampled *multinomially* so that, if $\boldsymbol{q}$ is any population of size N, the probability that it is the next generation after $\boldsymbol{p}$ is

$$N! \prod \frac{(\mathcal{G}(\boldsymbol{p})_j)^{Nq_j}}{(Nq_j)!}$$

Thus $\mathcal{G}(\boldsymbol{p})$ describes the probability distribution over all populations for the next generation. It also has further interpretations, as in the following theorem.

Theorem 6.1 (Vose [295]) *If the current population is given by the population vector $\boldsymbol{p}$, then the expected next population is $\mathcal{G}(\boldsymbol{p})$.*

One interpretation therefore, is that the vector $\mathcal{G}(\boldsymbol{p})$ describes the average over all possible next generations. Clearly, it is unlikely actually to *be* the next generation—it is only the *expected* value of the next population vector. There will be some variance about the average, and it can be shown that this variance is inversely proportional to the population size. That is, as N increases, the variance becomes smaller, and the probability distribution for the next generation is focused on $\mathcal{G}(\boldsymbol{p})$. Indeed, in the limit as $N \to \infty$ the variance shrinks to zero, and the next generation will actually be $\mathcal{G}(\boldsymbol{p})$. In the infinite population limit, the process becomes deterministic, and follows the trajectory in the simplex given by $\boldsymbol{p}, \mathcal{G}(\boldsymbol{p}), \mathcal{G}^2(\boldsymbol{p}), \ldots$. The operator $\mathcal{G}$ can therefore also be interpreted as an *infinite population* model of a GA. For finite populations, of course, there will be a stochastic deviation from this trajectory, which can be viewed as arising from sampling errors.

A study of the dynamics of the genetic algorithm thus comprises an analysis of the operator $\mathcal{G}$, the trajectory it describes in Λ, and the effects of having a finite population creating deviations from this trajectory. This general theory has been worked out by Michael Vose and co-workers, and we shall follow their approach [293]. Some important issues will be:

1. Deriving equations for $\mathcal{G}$.
2. Relating the structural properties of the genetic operators to these equations.
3. Studying the dynamics of $\mathcal{G}$. We shall be interested in fixed points, and the trajectories of populations.
4. Relating the dynamics to the stochastic behaviour of finite populations.

We shall now look at the equations for $\mathcal{G}$ and their associated dynamics for selection, mutation, and crossover.

6.2 Selection

We have previously seen that if we use proportional selection, the probability of any individual $i \in \mathcal{X}$ being selected is

$$\mathbf{Pr}[i|\boldsymbol{v}] = \frac{v_i f(i)}{\sum_{j \in \mathcal{X}} v_j f(j)}$$

where $\boldsymbol{v}$ is the incidence vector for the population. Dividing the numerator and denominator of the fraction by the population size N enables us to re-write this in terms of the population vector $\boldsymbol{p}$:

$$\mathbf{Pr}[i|\boldsymbol{p}] = \frac{p_i f(i)}{\sum_{j \in \mathcal{X}} p_j f(j)}$$

We can write this more compactly if we view the fitness function f as a vector $\boldsymbol{f} \in \mathbb{R}^n$ given by $f_k = f(k)$. Viewing selection as an operator $\mathcal{F} : \Lambda \to \Lambda$ that acts on the simplex then

$$\mathcal{F}(\boldsymbol{p}) = \frac{\mathrm{diag}(\boldsymbol{f})\boldsymbol{p}}{\boldsymbol{f}^T\boldsymbol{p}}$$

where $\mathrm{diag}(\boldsymbol{f})$ is the diagonal matrix with entries from vector $\boldsymbol{f}$ along the diagonal, and $\boldsymbol{f}^T\boldsymbol{p}$ is the inner product of vectors $\boldsymbol{f}$ and $\boldsymbol{p}$.

For example, suppose our search space is $\mathcal{X} = \{0, 1, 2, 3\}$ and the fitness function is

$$\begin{aligned} f(0) &= 2 \\ f(1) &= 1 \\ f(2) &= 3 \\ f(3) &= 0 \end{aligned}$$

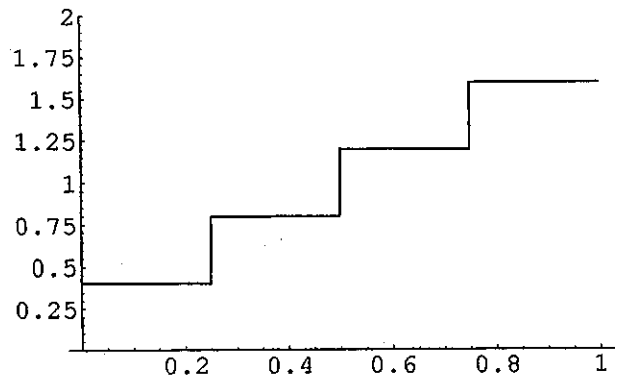


Figure 6.5: Probability density function $\rho(x)$ corresponding to a particular set of ranking probabilities $R = (0.1, 0.2, 0.3, 0.4)$ with population size $N = 4$. The density function is defined to be $\rho(x) = 4R(\lfloor 4x \rfloor)$.

For example, suppose that $N = 4$, and we assign the following probabilities to ranks

i	0	1	2	3
$R(i)$	0.1	0.2	0.3	0.4

then the step function $\rho(x)$ is as shown in Figure 6.5.

Given a population vector $\boldsymbol{p}$, suppose that $k \in \mathcal{X}$ is the worst individual in $\boldsymbol{p}$, i.e., p_k is non-zero and $p_j = 0$ for all $j \in \mathcal{X}$ such that $f(j) < f(k)$. The probability that k is selected is given by collecting together the first Np_k steps of the function ρ, this being the number of copies of k that are in the population. The probability of selecting the next best individual is found in a similar way, by collecting together the appropriate number of steps of ρ. Notice that this works even if there are no copies of an individual in the population, as then we simply collect zero steps. In general, then, we can find the probability of selecting any individual $k \in \mathcal{X}$ by integrating over ρ to collect together the appropriate number of steps:

$$\mathcal{F}(\boldsymbol{p})_k = \int_{\sum[f(j)<f(k)]p_j}^{\sum[f(j)\leq f(k)]p_j} \rho(y)dy$$

Continuing with the example, suppose we have a search space with four indi-

table shows how the rank probabilities are calculated:

k	$\sum[f(j) < f(k)]p_j$	$\sum[f(j) \leq f(k)]p_j$	$\mathcal{F}(\boldsymbol{p})_k$
0	0	$1/4$	$\int_0^{1/4} \rho(y)dy = 0.1$
1	$1/4$	$1/4 + 1/2$	$\int_{1/4}^{3/4} \rho(y)dy = 0.2 + 0.3 = 0.5$
2	$1/4 + 1/2$	$1/4 + 1/2 + 0$	$\int_{3/4}^{3/4} \rho(y)dy = 0$
3	$1/4 + 1/2 + 0$	$1/4 + 1/2 + 0 + 1/4$	$\int_{3/4}^{1} \rho(y)dy = 0.4$

This method of defining rank-based selection can be generalised by using any non-decreasing probability density function $\rho(x)$ in the integral.

6.3 Mutation

In the previous chapter, the contribution of mutation to the transition matrix was described in terms of a $n \times n$ matrix $\boldsymbol{U}$ in which $U_{i,j}$ is the probability that individual $j \in \mathcal{X}$ mutates to individual $i \in \mathcal{X}$. The same matrix directly gives the effect of mutation on a population vector. We seek an operator $\mathcal{U} : \Lambda \to \Lambda$ such that $\mathcal{U}(\boldsymbol{p})$ is a vector describing the probability distribution over $\mathcal{X}$, from which the next population is chosen multinomially—reflecting the effect of applying mutation to the population vector $\boldsymbol{p}$. It is simple to show that

$$\mathcal{U}(\boldsymbol{p}) = \boldsymbol{U}\boldsymbol{p}$$

is the required operator.

The combined effects of applying selection and mutation to a population vector are given by $\mathcal{U} \circ \mathcal{F}$. For the case of proportional selection this gives

$$\mathcal{U} \circ \mathcal{F}(\boldsymbol{p}) = \frac{\boldsymbol{U}\mathrm{diag}(\boldsymbol{f})\boldsymbol{p}}{\boldsymbol{f}^T\boldsymbol{p}}.$$

If our GA consists only of proportional selection and mutation (i.e. no crossover) then $\mathcal{G} = \mathcal{U} \circ \mathcal{F}$. The fixed points of $\mathcal{G}$ in this case are given by

$$\frac{\boldsymbol{U}\mathrm{diag}(\boldsymbol{f})\boldsymbol{p}}{\boldsymbol{f}^T\boldsymbol{p}} = \boldsymbol{p},$$

so that

$$\boldsymbol{U}\mathrm{diag}(\boldsymbol{f})\boldsymbol{p} = (\boldsymbol{f}^T\boldsymbol{p})\boldsymbol{p}.$$

associate the search space $\{00, 01, 10, 11\}$ with the integers $\{0, 1, 2, 3\}$ using the standard binary representation. $r(i, j, \mathbf{0})$ therefore is the probability that i and j cross to form $\mathbf{0} = (00)$. The following table gives the different values of $r(i, j, \mathbf{0})$:

	00	01	10	11
00	1	1/3	2/3	1/3
01	2/3	0	1/3	0
10	1/3	0	0	0
11	1/3	0	0	0

It can be seen that one-point crossover is not symmetric since, for example,

$$r(00, 01, 00) \neq r(01, 00, 00)$$

However, we can take the matrix $\boldsymbol{M}_0$ to be symmetric and still obtain the correct quadratic form for $\mathcal{C}$.

$$\boldsymbol{M}_0 = \begin{bmatrix} 1 & 1/2 & 1/2 & 1/3 \\ 1/2 & 0 & 1/6 & 0 \\ 1/2 & 1/6 & 0 & 0 \\ 1/3 & 0 & 0 & 0 \end{bmatrix}$$

Doing this for each $k \in \mathcal{X}$ gives us

$$\mathcal{C}(\boldsymbol{p}) = (\boldsymbol{p}^T \boldsymbol{M}_0 \boldsymbol{p}, \boldsymbol{p}^T \boldsymbol{M}_1 \boldsymbol{p}, \ldots, \boldsymbol{p}^T \boldsymbol{M}_{n-1} \boldsymbol{p})$$

where each $\boldsymbol{M}_k$ is symmetric.

It should be noticed that this means that different crossovers might lead to the same operator $\mathcal{C}$, indicating that they have identical effects on a population. For example, suppose we have two different forms of one-point crossover. The first one generates a single offspring from the left-hand end of one parent and the right-hand end of the other parent. The second one generates both offspring and selects one of them at random. It can be checked that both of these crossovers have the same corresponding set of symmetric matrices $\boldsymbol{M}_k$ and that therefore their effect on populations is identical. (A similar effect in the case of permutations is demonstrated by Whitley and Yoo [309].)

6.4.1 Mixing

If we combine crossover with mutation we obtain the *mixing* operator

$$\mathcal{M} = \mathcal{C} \circ \mathcal{U}$$

where mutation is applied to parents before crossover. For a large class of *independent* mutation operators (including all the standard ones on bit-strings), it doesn't matter whether mutation is applied before crossover or afterwards [293]. The mixing operator also turns out to be a quadratic operator [243]:

$$\mathcal{M}(\boldsymbol{p})_k = \boldsymbol{p}^T (\boldsymbol{U}^T \boldsymbol{M}_k \boldsymbol{U}) \boldsymbol{p},$$

where $\boldsymbol{U}$ is the mutation matrix. Some of the properties of $\mathcal{M}$ that relate to the structure of the search space $\mathcal{X}$ will be described in the following section.

The full sequence of selection, mutation and crossover gives us the complete operator for the genetic algorithm:

$$\mathcal{G} = \mathcal{M} \circ \mathcal{F}$$

If selection is proportional to fitness then, as we have seen,

$$\mathcal{F}(\boldsymbol{p}) = \frac{\operatorname{diag}(\boldsymbol{f})\boldsymbol{p}}{\boldsymbol{f}^T \boldsymbol{p}}.$$

One of the properties of quadratic operators such as $\mathcal{M}$ is that, for any vector $\boldsymbol{x}$ and scalar α

$$\mathcal{M}(\alpha \boldsymbol{x}) = \alpha^2 \mathcal{M}(\boldsymbol{x}).$$

From this it follows that, for proportional selection

$$\mathcal{M}\left(\frac{\operatorname{diag}(\boldsymbol{f})\boldsymbol{p}}{\boldsymbol{f}^T \boldsymbol{p}}\right) = \frac{\mathcal{M}(\operatorname{diag}(\boldsymbol{f})\boldsymbol{p})}{(\boldsymbol{f}^T \boldsymbol{p})^2}$$

and so we obtain

$$\mathcal{G}(\boldsymbol{p})_k = \frac{\boldsymbol{p}^T (\operatorname{diag}(\boldsymbol{f}) \boldsymbol{U}^T \boldsymbol{M}_k \boldsymbol{U} \operatorname{diag}(\boldsymbol{f})) \boldsymbol{p}}{(\boldsymbol{f}^T \boldsymbol{p})^2}$$

for each $k \in \mathcal{X}$, which gives us the complete equation for the genetic algorithm. Iterating this equation gives us the infinite population trajectory for the genetic algorithm with proportional selection.

As in the case of selection and mutation, the dynamics of a finite population genetic algorithm with crossover are strongly influenced by the fixed points of the infinite population system [290]. It can be shown empirically that in many cases the finite population spends most of its time in the vicinity of a fixed point. It eventually escapes and quickly becomes captive to

with the performance of algorithms that search for the optimum of a cost function.

Summarizing briefly, what Wolpert and Macready show is that, averaged over all possible functions, the performance of all search algorithms that satisfy certain conditions is the same. For example, if these conditions hold, random search is on average as good as any more sophisticated algorithm such as a GA. However, the idealized search algorithm they describe is different in some important respects from 'real' search algorithms such as simulated annealing, tabu search, or genetic algorithms. Nonetheless, it is frequently claimed that these results are relevant to such algorithms—as for example,

> *..., if simulated annealing outperforms genetic algorithms on some [sub]set ϕ [of the set of all functions $\mathcal{F}$], genetic algorithms must outperform simulated annealing on $\mathcal{F} \setminus \phi$.* [314]

This is true in terms of what Wolpert and Macready mean by 'outperform', but as we shall argue below, their interpretation is somewhat idiosyncratic, and makes certain assumptions that may not be easily verified in practice. Firstly, we present the theorem, largely in the notation of [314], which is slightly different from that used in the rest of this book.

4.2 The Theorem

Assume we have a discrete search space $\mathcal{X}$, and a function

$$f : \mathcal{X} \mapsto \mathcal{Y} \subset \mathbb{R}.$$

For convenience, the value of $f(\mathbf{x})$ is called the cost of $\mathbf{x}$. The general problem is to find an optimal solution, i.e., a point $\mathbf{x}^* \in \mathcal{X}$ that minimizes or maximizes f.

Wolpert and Macready assume a search algorithm A that has generated a set of m *distinct* points $\{d_m^{\mathbf{x}}(i)\}$, and associated cost values $\{d_m^{y}(i)\}$ where $y = f(\mathbf{x})$ and the index $i = 1, \ldots, m$ implies some ordering (the most obvious one being with respect to the search chronology). For convenience, the whole ensemble of points and cost values can be denoted by d_m.

Thus initially, starting from a point $\mathbf{x}_1$ with cost y_1, we have $d_1^{\mathbf{x}}(1) = \mathbf{x}_1$ and $d_1^{y}(1) = y_1$. Subsequent points are generated by A based on d_m; that is, A is a function

$$A : d_m \mapsto \mathbf{x} \in \mathcal{X} \setminus \{d_m^{\mathbf{x}}(i)\}$$

The information generated by this sequence of points can be encapsulated in a frequency table or 'histogram', $\vec{c}$, of the cost values $\{d_m^{y}(i)\}$; the quality of the algorithm's performance can be measured in terms of some characteristic of $\vec{c}$ such as its minimum, mean, or median. For a given f, the quantity of interest is thus the conditional probability $P[\vec{c}\,|f, m, A]$. For this histogram, Wolpert and Macready prove the following:

Theorem 4.1 (No Free Lunch) *For any pair of algorithms A_1 and A_2,*

$$\sum_f P[\vec{c}\,|f, m, A_1] = \sum_f P[\vec{c}\,|f, m, A_2].$$

where the sum is over all possible functions f.

As originally stated, the theorem also assumes that A is deterministic—which search methods such as GAs and simulated annealing are definitely not. However, the definition of A can be extended so that the NFL theorem also encompasses stochastic algorithms.

More generally, attention can be shifted to some *performance measure* $\Phi(\vec{c})$. As a corollary, it is clear that the average of $P[\Phi(\vec{c})\,|f, m, A]$ over all functions f is independent of A, which provides the justification for such interpretations of the theorem as those quoted above.

4.2.1 Representations

Although [314] was the first rigorous statement of the 'No Free Lunch' concept, it is not the only one, and its proof is certainly not the easiest to follow. Radcliffe and Surry [206] approached this topic by considering the related issue of the choice of representation. Earlier (Chapter 2) we described the way in which an encoding function c is often used to represent the search space $\mathcal{V}$ indirectly in a representation space $\mathcal{X}$. It is intuitively obvious that some representations will work better than others in any particular case—for example, consider the experimental evidence regarding binary or Gray coding.

While early work in GAs tended to assume by default that the conventional integer-to-binary code was always appropriate, it does have problems. For example, suppose that the optimum of a function occurs at a point whose value is the integer 32, and that we are using a 6-bit chromosome, which means the binary mapped value is (100000). If the function is reasonably smooth, a good practical approximation to this value is 31, which maps to the maximally different string (011111). Conversely, the (very close) bi-