Studies in Evolutionary Algorithms

Michael Stillwell

B.A., B.Sc. (Hons) Monash University

A major thesis submitted in fulfillment
of the requirements for the
Degree of Master of Computing

School of Computer Science and Software Engineering
Faculty of Information Technology
Monash University

November 13, 2001

Contents

Declaration

I, Michael Stillwell, affirm that the thesis contains no material which has been accepted or submitted for the award of degree at any university.

To the best of my knowledge and belief, the thesis contains no material previously published or written by any other person, except where due reference is made in the text.

Acknowledgements

So many people helped me with this work. I must first thank my family: my mother and father, and my brother. They have been supportive when I needed it, inspirational when I lacked it, and reasonable when I was not.

I thank my father, John Stillwell, for help with some of the mathematical details. (If only math were as easy for me as it is him.)

I am most grateful to the School of Computer Science and Software Engineering for providing me with a postgraduate scholarship.

Thanks also go to my two office-mates: Brian May (who kept me up-to-date with what was going on in the computer world as I struggled with evolutionary biology) and Tao Chen (who kept me company late—and sometimes very late—at night).

Finally I wish to thank Kevin Korb and Lloyd Allison: two of the finest supervisors one could ever hope to know. Their wit and insight informed every single page of this work.

Chapter 1

Introduction

1.1 About Evolutionary Algorithms

Evolutionary Algorithms (EAs) simulate the process of evolution on a digital computer: for each aspect of evolution in nature—the genome, reproduction, "survival of the fittest," etc.—EAs have their artificial analogue. There are several different types of EAs, the most prominent of which are genetic algorithms and genetic programming. In each, we have a collection of binary strings[1] representing "individuals," we have a fitness function that determines the effectiveness of each individual, and we have functions that simulate the effects of reproduction and death.

Computer scientists (and others) are interested in EAs due to what one might say is the flip side to the "argument from design." In the 19th Century, many people found evolution hard to accept, in part because it required one to believe that a process as directionless, haphazard and random as evolution could produce structures as complex and subtle as eyes, ears and wings. Every part of every animal seemed not only perfectly designed, but perfectly designed to work with each other. These facts seemed to lead to the conclusion that God, not evolution, was behind it all.

Needless to say, the agency of evolution has now been established to

[1] Or data structures ultimately reducible to binary strings.

(almost) everyone's satisfaction. But this doesn't take anything away from the beauty and sophistication of the natural world that Darwin's doubters so strongly felt. If evolution, and not an omniscient and omnipotent God, created the birds and the bees, then evolution is clearly a problem-solving mechanism of extraordinary power.

Though evolution is both undirected and unthinking, it is able—if given sufficient time—to devise solutions to remarkably difficult engineering problems. The sensory organs of many animals, for example, exceed by far the capabilities of the best human designs. And the human brain—perhaps the most impressive product of evolution's factory—is an organ not likely to be surpassed anytime soon by anything humans can devise, at least in terms of space requirements, power consumption, noise production, durability, and price.

These are all reasons to find evolution interesting, but computer scientists have an additional one: it seems that all there is to it is the mindless execution of a few simple instructions very many times, which is exactly what computers are good for. For the simulation of evolution, computers seem perfect; if we are indeed able to simulate the process of evolution *in silico*, then perhaps we can use computers to bring the problem-solving power of evolution to bear on problems of our own.

1.2 Thesis Overview

There are two main sections to this thesis. In the first, we investigate the dynamics of EAs, with a particular focus on Genetic Algorithms (GAs). After describing how GAs work, we present and critique two prominent dynamical descriptions, the Schema Theorem and the Building Block Hypothesis, both of which attempt to describe and predict how GAs behave. We conclude the section with some remarks on the potential of EAs.

The second section deals with the process of selection—that is, which individuals should be selected to reproduce and when. We explain and describe

some important selection schemes, then detail the results of our experiments on "selection profiles."

Chapter 2

The Dynamics of Evolutionary Algorithms

2.1 What are Genetic Algorithms?

Of all the Evolutionary Algorithms, the best understood are Genetic Algorithms (GAs). They are also the variety closest to evolution as it occurs in nature. We briefly describe the operation of a canonical GA here.

GAs work with a population of individuals (usually a few hundred), where each individual is a bit string (usually less than one hundred bits). This population is initialised to random bit strings. Each individual represents a potential solution to a problem. For example, if we are trying to find the value of x that maximises the function $x \cdot \sin(10\pi \cdot \pi) + 1.0$ over the range $[-1, 2]$, different individuals will encode different values of x. In the first instance, the GA aims to turn a population of good solutions into a population of better ones. It does this by "breeding" from the better individuals using two procedures copied from nature: crossover and mutation. Crossover is analogous to sexual reproduction—it takes two parent individuals (two bit strings) and creates a child from them by taking some genetic material from one parent (say the first 30 bits), and some from the other (the last 70). Mutation is simply the flipping of some small number of randomly chosen

bits. After the GA has created a new population of individuals in this way, it repeats the process until the optimum is found, or some stopping point is reached.

2.2 Genetic Algorithms as a Search Procedure

GAs are inspired by a biological process, but in reality they are nothing more than a search procedure. Suppose we have a problem whose solution is to be found at one of the 2^{100} points in the search space. A GA represents just one way of sifting through this search space; there are of course many others. We briefly list some here (after Langdon, 1998, p. 15–6):

Exhaustive Enumeration Try each of the 2^{100} bit strings in turn until a solution is found.

Newton's Method Calculate (or estimate) the derivative of the function representing the surface to be maximised and use this information to step toward the solution.

Simulated Annealing Pick a point at random then take ever-smaller steps in the general direction (chosen stochastically) of better points.

It is relatively easy to determine what search trajectories each of these three search techniques will take through the search space. Consequently, it is easy to see which sorts of problems they are likely to find hard, and which they are likely to find easy. (Problems that give rise to a smooth search space will (in general) be easy, problems with a rough search space will (in general) be hard—exhaustive enumeration finds every problem equally difficult.) The search trajectory of evolutionary algorithms, however, is considerably more difficult to determine; the action of crossover, in particular, seems to lead to large, difficult-to-track jumps from one point to another.

Nevertheless, it is vital that this process be understood. Evolutionary algorithms have successfully solved many non-trivial problems[1]. However, for EAs to enjoy even greater success, their workings must be better understood. Knowing that they do work is not enough: we also need to know exactly why they work. Why *does* the process of repeated selection and reproduction frequently yield good solutions to problems? And especially: which problems do they find easy, and which do they find hard? Answering these questions requires a full account of the dynamic behaviour of evolutionary algorithms.

A suitable theory is not likely to be developed any time soon. However, over the past 25 years, our understanding of evolutionary algorithms has gradually improved. Two important contributions are the Schema Theorem and the Building Block Hypothesis. These are discussed below.

2.3 The Schema Theorem

In *Adaptation in Natural and Artificial Systems* (Holland, 1975), John Holland presented the Schema Theorem, which says that substrings of above-average fitness will increase in the population at an exponential rate. Though this was hardly a complete account of the dynamic behaviour of GAs, it was useful in that it provided a hint as to why they were so successful: even though we did not know exactly what was going on, we did know that useful substrings ("building blocks") were accumulating within the population. We describe the Schema Theorem below. Later on we consider some objections.

[1]We are aware of Wolpert and Macready (1995) (following Hume, 1739), which proves that no search procedure can outperform any other (including blind search) across the set of all search problems. However, we hold that the world is such that some problems are more likely to be encountered than others, making some search procedures more useful than others. (In the same way, some data compression routines happen to be more useful than others, despite the fact that no data compression routine can outperform any other across the set of all data.)

2.3.1 What is a Schema?

A *schema* (plural schemata) is a string of letters taken from the alphabet $\{0, 1, *\}$. Schemata are templates, or patterns, where the * character represents the "don't care" symbol (it is a wildcard). For example, the pattern `101*****` matches all binary strings of length 8 that begin with `101`; strings that match schemata are said to be *instances* of them. The number of characters that must match (i.e., the number of non-wildcards) for a schema to be satisfied is known as the schema's *order*; the distance between the first non-wildcard and the last non-wildcard is the schema's *defining length*. For example, the schema `**1*10**` is of order 3 and has a defining length of 4. We will sometimes use the word "schema" to mean the template itself, and sometimes to mean the set of strings[2] the template matches. Note that the set of all schemata is much smaller than the set of all subsets; there are 3^l schemata of length l but there are 2^{2^l} possible bit string subsets.

How do schemata help us understand the dynamics of genetic algorithms? To answer this question, we first need to understand the dynamics of the individual bit strings. A genetic algorithm explicitly calculates the fitness of every bit string in the population. (So for some problem, the bit string `11101010` might have a fitness of 0.67 and the string `11111111` might have a fitness of 1.0.) In Holland's original, canonical, GA, individual strings are selected for reproduction on the basis of their relative fitness, that is, the fitness of the string, divided by the mean fitness of all the bit strings in the population.

This means that if $f(s)$ is the objective fitness of s, f_μ is the mean fitness, and $m(s)$ is s's number of offspring, then the expected number of offspring is

$$E(m(s)) = f(s)/f_\mu \tag{2.1}$$

[2]At least one author has taken this idea even further, and considers schemata to be *predicates*—"`*11`(x) = true iff x matches `*11` at every position not containing `*`" (Vose, 1991, p. 386). It is sometimes useful to think of schemata as limited predicates in this way.

We can also calculate the variance. If n is the population size, then

$$\begin{aligned} Var(m(s)) &= n\left(\frac{f(s)}{nf_\mu}\right)\left(1-\frac{f(s)}{nf_\mu}\right) \\ &= \frac{f(s)}{f_\mu} - \frac{f(s)^2}{nf_\mu^2} \end{aligned} \tag{2.2}$$

(After Feller, 1968, p. 228.)

Holland realised that although it is individual strings that are directly manipulated—the fittest ones increase in number, the less-fit ones decrease and so on—the schemata contained within the population are subject to exactly the same pressures. (This is proved in Section 2.3.2.) That is to say, if the mean fitness of schema `11******` (the mean of the fitnesses of all matching bit strings) is twice that of `00*****`, and there are equal numbers of both schema, then we can expect that in the next generation there will be twice as many instances of `11******` as there are of `00******`, *even though it is individual bit strings, and not schemata, that are explicitly processed.* For each schema present (i.e., each schema that matches at least one member of the population), the change in frequency from one generation to the next is proportional to the schema's relative mean fitness. (That is, the mean fitness of all matching bit strings, divided by the mean fitness of the population.) Because the number of schema "processed" is vastly greater than the number of individual bit strings processed (see Appendix A for exactly how much more), GAs are said to exhibit "implicit parallelism." This, it is claimed, is one source of their power.

2.3.2 The Fate of Individuals is not Pertinent

As we have observed, a population contains many more schemata than individuals. (Michalewicz (1996, p. 54) wryly notes, "This constitutes possibly the only known example of a combinatorial explosion working to our advantage instead of our disadvantage.") There are 3^l possible schemata in a given population, where l is the length of each bit string because each position can take on the value `0`, `1` or `*`. A possible arrangement is shown in Table 2.1.

Schema	(Mean) Fitness
`*****`	0.70
`****0`	0.60
`****1`	1.20
⋮	⋮
`*101*`	Not present
`*110*`	0.50
`*111*`	0.80
⋮	⋮
`11111`	1.50

Table 2.1: Possible arrangement of schemata.

It is claimed that from one generation to the next, the frequency of each of the 3^l schemata change as dictated by their relative fitness. This is to be achieved by processing individual bit strings.

At first blush, this appears to be impossible. But possible it is: though there are a multitude of constraints, as it turns out, they can all be satisfied. If $s_1, s_2, \ldots, s_k$ are (not necessarily unique) instances of schema H, then the fitness of H is[3]

$$f(H) = \frac{\sum_{i=1}^{k} f(s_i)}{k}$$

The relative fitness of H is then

$$f_{\mathcal{R}}(H) = f(H)/f_\mu = \frac{\sum_{i=1}^{k} f(s_i)}{k f_\mu}$$

We want the number of matching individuals, k, to increase by the relative fitness, or $f_{\mathcal{R}}(H)$. Therefore, we expect that the number of instances of schema H in the next generation will be $k \times f_{\mathcal{R}}(H)$. But the GA cannot directly control the incidence of individual schemata—the fate of H depends on the fate of the individual bit strings that represent it (if we ignore crossover

[3]This notation is due to Goldberg (1989); H stands for "hyperplane."

and mutation). What is the GA's effect on *them*? According to equation (2.1) the expected frequency of bit string s_i is $f(s_i)/f_\mu$. Thus

$$\begin{aligned}\sum_{i=1}^{k} f(s_i)/f_\mu &= k \times f_{\mathcal{R}}(H) \\ &= \frac{k\sum_{i=1}^{k} f(s_i)}{kf_\mu}\end{aligned}$$

where the LHS and RHS are readily seen to be equal. Thus, the expected size of the schema as calculated by the change in frequency of its members (left side) equals the expected size of the schemata as calculated by schema's fitness (right side), as required. Hence, we can re-write equation (2.1) in terms of schemata:

$$E(m'(H)) = m(H) \times f_{\mathcal{R}}(H) = m(H)\frac{f(H)}{f_\mu} \tag{2.3}$$

which demonstrates that a GA behaves as if it were processing schemata directly, even though it does no such thing. The variance is

$$\begin{aligned}Var(m'(H)) &= \sum_{i=1}^{k}\left(\frac{f(s_i)}{f_\mu} - \frac{f(s_i)^2}{nf_\mu^2}\right) \\ &= \frac{1}{f_\mu}\sum_{i=1}^{k} f(s_i) - \frac{1}{nf_\mu^2}\sum_{i=1}^{k} f(s_i)^2\end{aligned} \tag{2.4}$$

Note that these relations hold for every one of the 3^l schemata. This is trivially so in the case of the schemata not represented (not matched), since their observed fitness is (in effect) zero, and thus their expected frequency is also zero. But every schema, of every order and defining length that *is* represented will change in frequency according to equation (2.3), with a variance equal to equation (2.4).

2.3.3 Change in Frequency over Time

From equation (2.3) it can be seen that the change in schema frequency is dependent on the value of the expression $f(H)/f_\mu$. Consequently, the only

possible rate of growth is exponential: if the fitness of the schema stays above the mean, it will experience an exponential increase; if the fitness is below the mean, it will experience an exponential decrease. It is not possible to give figures any more precise than this without knowledge of the specific problem being investigated, the objective function used, and the resulting populations.

2.3.4 Change in Fitness over Time

We can also use equation (2.3) to determine the change in (mean) schema fitness over time. At each generation after the first, the schema frequency is equal to the sum over $s \in H$ of however many instances there were in the previous generation, multiplied by the reproduction rate of each s. The mean fitness is the sum over $s \in H$ of the expected number of instances multiplied by the fitness of s, divided by the expected number of matches for the entire schema. That is,

Gen.	Schema Frequency	Schema Fitness
0	k	$\sum_{i=1}^{k} f(s_i)/k$
1	$\sum_{i=1}^{k} f(s_i)/f_\mu(0)$	$\frac{\sum_{i=1}^{k} f(s_i) \times f(s_i)/f_\mu(0)}{\sum_{i=1}^{k} f(s_i)/f_\mu(0)} = \frac{\sum_{i=1}^{k} f(s_i)^2}{\sum_{i=1}^{k} f(s_i)}$
2	$\sum_{i=1}^{k} f(s_i)/f_\mu(0) \times f(s_i)/f_\mu(1) = \frac{\sum_{i=1}^{k} f(s_i)^2}{f_\mu(0) \times f_\mu(1)}$	$\frac{\sum_{i=1}^{k} f(s_i) \times f(s_i)/f_\mu(0) \times f(s_i)/f_\mu(1)}{\frac{\sum_{i=1}^{k} f(s_i)^2}{f_\mu(0) \times f_\mu(1)}} = \frac{\sum_{i=1}^{k} f(s_i)^3}{\sum_{i=1}^{k} f(s_i)^2}$
$\vdots$	$\vdots$	$\vdots$
g	$\frac{\sum_{i=1}^{k} f(s_i)^k}{f_\mu(0) \times f_\mu(1) \times \cdots \times f_\mu(k-1)}$	$\frac{\sum_{i=1}^{k} f(s_i)^{g+1}}{\sum_{i=1}^{k} f(s_i)^g}$

where $s_1, s_2, \ldots, s_k$ are the (not necessarily unique) bit strings that match H and $f_\mu(g)$ is the mean fitness of the population at generation g.

Hence, if $\Delta(H, g)$ is the expected fitness of schema H at generation g, then

$$\Delta(H, g) = \frac{\sum_{i=1}^{k} f(s_i)^{g+1}}{\sum_{i=1}^{k} f(s_i)^{g}} \tag{2.5}$$

This result has a number of surprising implications: (a) a schema's fitness depends only on the fitnesses of the bit strings that match it—the fitness of non-matching strings, the fitness of the population as a whole, and the size of population is irrelevant; (b) over time, the mean fitness of each and every schema can be expected to increase (even schemata of below average fitness); and (c) the mean fitness approaches that of the fittest member.[4]

The proof is as follows. Suppose $T = \{t_1, t_2, \ldots, t_m\}$ (where $t_i > 0$) and

$$f(n) = \frac{\sum_{i=1}^{m} t_i^{n+1}}{\sum_{i=1}^{m} t_i^{n}}$$

Without loss of generality, assume that t_m is $\max(T)$. Then

$$f(n) \quad = \quad \frac{\sum_{i=1}^{m} t_i^{n+1}}{\sum_{i=1}^{m} t_i^{n}}$$

[4]The conclusions are possibly unexpected, but they do have a satisfactory explanation. The paradoxes arise not because the effects of crossover and mutation have been ignored but because we have effectively assumed that individual bit strings can exist in fractional quantities, which is of course impossible.

Initially, a schema is represented by a number of distinct, but not necessarily unique bit strings, each present exactly once. However, after one generation, their expected quantities change, most likely to a non-integral number. This means that the expected number of instances of a schema of below average fitness approaches zero, but never equals it. So whilst the mean fitness of such schemata do increase according to (2.5), the proportion of the population that they occupy dwindles to zero. But, since a GA cannot deal with fractions of bit strings, what actually happens is that such schemata die out.

Nevertheless, if a schema is relatively common, and of near-average fitness or better, then over a small number of generations equation (2.5) will accurately track the change in mean fitness.

$$
\begin{aligned}
&= \frac{t_m^{n+1} \sum_{i=1}^{m} \left(\frac{t_i}{t_m}\right)^{n+1}}{t_m^n \sum_{i=1}^{m} \left(\frac{t_i}{t_m}\right)^n} \\
&= \frac{t_m^{n+1} \left(\left(\frac{t_m}{t_m}\right)^{n+1} + \sum_{i=1}^{m-1} \left(\frac{t_i}{t_m}\right)^{n+1}\right)}{t_m^n \left(\left(\frac{t_m}{t_m}\right)^{n} + \sum_{i=1}^{m-1} \left(\frac{t_i}{t_m}\right)^{n}\right)} \\
&= t_m \frac{1 + \sum_{i=1}^{m-1} \left(\frac{t_i}{t_m}\right)^{n+1}}{1 + \sum_{i=1}^{m-1} \left(\frac{t_i}{t_m}\right)^{n}} \\
\therefore \lim_{n\to\infty} f(n) &= t_m
\end{aligned}
$$

(As n tends to ∞, the sums tend to zero since each of the summed terms is less than 1.) This proves that over time, any set of (positive) fitness values will approach the value of the most-fit member.

2.3.5 Incorporating the Effects of Crossover and Mutation

So far we have examined what happens to schemata in the absence of crossover and mutation. Needless to say, any useful GA does have these things. In this section, we incorporate the effects of both operations into our analysis in order to derive Holland's Schema theorem in full.

We shall only consider the negative effects of crossover and mutation—that is, the ways in which their actions eliminate schemata—for the constructive effects are difficult to quantify. In doing so, we will produce a lower bound on the chance that a schema will survive into the next generation. (If a schema is said to "survive," then at least one instance exists in the next generation.)

We consider only single-point crossover, which refers to a form of crossover involving two parent strings. Under single-point crossover, we pick a point within the string a random, and form offspring by copying all the bits up till that point from one parent and the remainder from the other. If single-point crossover occurs with probability p_c, then the chance that a schema

embedded within a string survives into the next generation is dependent on its defining length, $\mathcal{L}(H)$. Specifically, the chance is $1 - p_c \frac{\mathcal{L}(H)}{l-1}$, where l is the length of the bit string.

The chance that mutation will destroy a schema is dependent on the mutation rate, p_m. A schema H of order $\mathcal{O}(H)$ will survive if none of the positions at which it is defined (i.e., the non-wildcarded positions) are subjected to mutation. The chance that a bit is not disrupted is $1 - p_m$ and hence the chance that a schema survives mutation is $(1 - p_m)^{\mathcal{O}(H)}$.

We can now add this information to equation (2.3) in order to produce Holland's Schema Theorem:

$$E(m'(H)) \geq \left(m(H) \frac{f(H)}{f_\mu} \right) \left(1 - p_c \frac{\mathcal{L}(H)}{l-1} \right) \left((1 - p_m)^{\mathcal{O}(H)} \right) \tag{2.6}$$

Because it only takes into account the destructive effects of crossover and mutation, the Schema Theorem provides a *lower bound* on the expected number instances of schema H that will survive into the next generation.

Note that the two right-most terms, which represent the probability that a schema will be disrupted due to crossover and mutation respectively, are inversely proportional to the length and order of the schema. This means that schemata of long defining length and high order are more likely to be disrupted than schemata of short defining length and low order. We shall see later on that they are also more likely to be eliminated by stochastic effects and noise.

2.3.6 The Implications of the Schema Theorem

The Schema Theorem captures a good deal of the dynamic behaviour exhibited by GAs. (Missing are the *constructive* effects of crossover and mutation. These are considered in Section 2.4.) Importantly, the Schema Theorem makes it possible to analyse a GA's behaviour at one level above that of individual bit strings, showing that schema of short defining length, low order, and above average fitness will tend to increase in number.

But, Holland argues, there is more to the Schema Theorem than this. By analogy to the Two-Armed Bandit problem, he claims that the Schema Theorem constitutes a proof that under some conditions, the way in which GAs "sample" schemata is optimal.

The Two-Armed Bandit Problem

The Two-Armed Bandit problem can be stated quite simply. Suppose we have a two-armed slot machine where one arm pays either μ_1 dollars (with variance σ_1) or μ_2 dollars (with variance σ_2), and the other pays μ_1 at σ_1 or μ_2 at σ_2 (whatever's left). We are then given some number of pulls on the arms of the machine. (On every pull we have a choice of arm.) Given that our object is to maximise the payoff, how should we allocate pulls?

This problem clearly involves making a choice between exploration and exploitation. An optimal strategy might be to "explore" for some small number of pulls, then allocate all the pulls that remain to the observed best arm. But when should we stop "exploring"—searching for the best arm, or making sure that the observed best really is the best—and start "exploiting"—pulling exclusively on the observed best arm? The nature of the problem is especially clear if we generalise the problem to a k-armed bandit, where k is so large that trialling every arm is not feasible.

Holland found that for this problem, the optimal approach is to allocate an *exponentially increasing number of trials to the observed best arm.* Here lies the connection to the Schema Theorem, and the proof of the optimality of GAs. As Holland (1975, p. 181) puts it: "schemata can be looked upon as random variables ... Under a genetic algorithm, a schema with an above-average fitness in the population increases its proportion exponentially ... If we think of the genetic algorithm as generating samples of n random variables (n-armed bandit), in a search for the best, then this exponential increase is just what Theorem 5.1 [which states the optimal strategy for the Two-Armed Bandit] suggests it should be."

A running GA can thus be considered to be playing a 3^l-Armed Bandit,

where each schema represents an arm. The fitness of each schema (arm) represents a sample from a random variable. According to the Schema Theorem, each schema receives an exponentially increasing (or decreasing) number of pulls, an arrangement which, by analogy to Holland's solution of the Two-Armed Bandit problem, is optimal.

Objections

Holland relates the Two-Armed Bandit problem to GAs like this: (a) the optimal solution to the Two-Armed Bandit Problem involves allocating an exponentially-increasing number of trials to the observed-best arm; (b) a GA allocates an exponentially-increasing number of trials to the observed-best schema; therefore (c) there is a sense in which GAs are optimal. There are two types of objection to this line of reasoning. First, complaints that his analysis of the Two-Armed Bandit problem is not correct. Second, objections to the analogy drawn between the Two-Armed Bandit problem and GAs.

As Wolpert and Macready (1996) point out, Holland only considers a severely limited class of strategies in his analysis. He allocates an equal number of trials to both arms; after a certain number of trials only the arm with the better observed payoff is pulled. Under this regime, the only decision to make is when to halt the "exploration" phase. (Holland's answer to this question leads, somewhat obscurely, to the conclusion that it is best to allocate an exponentially-increasing number of trials to the observed best arm.) If a more comprehensive class of strategies were considered, Wolpert and Macready argue, a better solution might be found. Holland's solution also exhibits a number of irregularities. For example, the point at which exploration is halted depends on the variance of one arm, but not the other.

But even if the optimal procedure is to allocate an exponentially increasing number of trials to the observed best arm (without saying exactly how many), the implications this has for GAs is not clear. Whilst GAs and the Two-Armed Bandit problem are both in some way concerned with the problem of "how to allocate resources in the face of uncertainty" (Mitchell, 1996,

p. 122)—that is, the trade-off between exploration and exploitation—this question is also an issue for just about any search procedure one cares to name. To be similar in this one respect in not reason enough to think that the solution to one is the solution to the other.

Also questionable is the claim that the generation-by-generation schema fitnesses can be thought of as samples from the same random variable. At generation zero, we can expect that for every schema, the set of bit strings that match a particular schema will be uniformly distributed. But as Grefenstette and Baker (1989) showed, after just a few generations, this set can exhibit a significant bias, which amounts to a change in the random variable represented by the schema. In other words, they showed that it is entirely possible for the fitness distribution governing a schema at generation zero to be completely different to the distribution that governs it at generation fifty. So even if schema fitnesses can be considered samples from *some* distribution, it seems that these distributions change over time, which amounts to changing the payoffs of the arms.

The argument-from-analogy has other problems too. In the case of the k-Armed Bandit, pulling an arm gives us information about the mean and variance of that arm only. But a GA, by sampling one schema, say `1111*···*`, can acquire information about the expected value of another, say `11111*···*`. (They are not independent.) So there seems to be some aspect to GAs that is not part of the Two-Armed Bandit problem. Another element that has no obvious parallel is the way in which a GA transforms solutions via crossover and mutation.

Finally, the argument only applies to fitness-proportional selection since this is the basis of the Schema Theorem itself. Very early on other selection schemes—schemes that do not allocate an exponentially-increasing number of trial to the observed best schama—were found to provide superior results. This again casts doubt on the supposed optimality of this approach.

Whilst the Schema Theorem goes a significant way toward capturing the dynamics of a GA in motion, it seems premature to conclude that GAs are in

some way optimal. Nevertheless, the basic Schema Theorem stands, and continues to be cited with approval in the literature. For example, whilst noting that the Schema Theorem is controversial, Banzhaf et al. (1998, p. 146) conclude that "the schema theorem remains the best starting point for a mathematically based analysis of the mechanisms at work in genetic algorithms using crossover." Recently, O'Reilly (1995), among others, has attempted to adapt the Schema Theorem to systems of Genetic Programming.

Genetic Algorithms and Function Optimisation

When playing the Two-Armed Bandit, the aim is to maximise the money won—there is no prize for doing such things as identifying the arm with the highest payoff. Problems of this sort—where the way in which the search is conducted is factored into the score—are said to have an "on-line" performance metric.

Some have argued that because GAs are not explicitly trying to find the optimal solution to a given problem, they don't make good function optimisers. For example, Mitchell (1996, p. 124) argues that the "goal" of a GA is to "*satisfice* (find a solution that is good enough for one's purposes) rather than to optimize (find the best possible solution)."

It is certainly true that in some cases a GA will not be able to find the optimal solution to a particular problem. Sometimes this will be because insufficient selection pressure was applied, in which case it might be said that the GA wasn't "trying" to find the optimal solution. But it is also possible that the GA is simply unable to find the solution. A GA can fail for reasons other than a lack of trying. (And in any case, what if nothing but the best is good enough?)

If the payoff is high enough, and the number of pulls large enough, the optimal solution to the k-Armed Bandit will involve searching through the arms until the optimum is found. Similar pressure can be exerted on a GA, but this provides no guarantee that the optimum will be found.

2.4 The Building Block Hypothesis

The Schema Theorem describes (in lower bound), how the frequencies of schemata *already present in the population* change over time. So if schema H exists, then the Schema Theorem can be used to estimate how many instances of H will exist in the next generation. However, it says absolutely nothing about what schemata might come to exist. In other words, the Schema Theorem cannot be used to determine what points in the search space will be considered in the next generation (or even how many new points will be considered), much less the chance that a solution will be reached within a certain number of generations. This is a significant shortcoming.

How schemata are created is an important topic to understand. For any non-trivial problem (and indeed, most trivial ones), the optimal bit string will not be present in the initial population. It must somehow be cobbled together from bits of other strings in the population. To put it another way, how does it come to pass that the short, low-order, and highly fit schemata favoured by the Schema Theorem come to be formed into the long, high-order, and ever-fitter schemata that comprise an optimal solution? A loose answer is provided by the Building Block Hypothesis.

2.4.1 What is the Building Block Hypothesis?

The Building Block Hypothesis is made up of two complementary parts. The first is the claim that (a) solutions to problems are decomposable into discrete parts, and (b) each of the parts will record a higher fitness than the parts of non-solutions. The second is the claim that GAs are (a) capable of identifying these parts and (b) able to assemble them into a solution. In short, the Building Block Hypothesis says that GAs can both find the "building blocks" that make up solutions, and put them together.

When expressed in terms as vague as these—as it commonly is (see Grefenstette (1993, p. 79) for examples)—it is not very difficult to prove the Building Block Hypothesis true. If the optimal bit string is not in the

initial population, then it must have been produced by crossover and/or mutation. If crossover, then we can justifiably say that each parent contributes a "building block." If mutation, then in all probability only a few bits were flipped, again leading to the conclusion that what we had previously were "building blocks."

The Building Block Hypothesis as stated above is not much use as it is. Too much is left unspecified: How can building blocks be identified? How long are they? Can they be distinguished from non-building blocks by way of their fitness and/or frequency? If all schemata are treated equally under the Schema Theorem, why are building blocks favoured, at least initially? How are they put together? Are some combinations more likely that others? How could you prove the Building Block Hypothesis true, or prove it false?

Grefenstette (1993) noticed that although the BBH was often loosely defined in the papers dealing with it, it was later invoked in a way that suggested that a much stricter definition was meant. This led Grefenstette to define a Static Building Block Hypothesis, which he describes as "an assumption" that "appears to underlie much of the recent published work in GA theory":

> **Static Building Block Hypothesis (SBBH)** Given any short, low-order hyperplane partition, a GA is expected to converge to the hyperplane [schema] with the best static average fitness (the "expected winner"). (Grefenstette, 1993, p. 78)

(The "static average" of a schema is the mean fitness of every matching bit string, not just those in the population.) This definition is still not explicit, but it is more precise than the one we had. Importantly, and in contrast with the BBH, the SBBH makes clear predictions about what schemata are likely to form, and what schemata are not likely to form. In particular, it predicts that problems are likely to be hard if schemata with high static fitnesses do not match the optimal bit string.

2.4.2 The Static Building Block Hypothesis

The SBBH typically comes up in studies of deception—or, more generally, the study of which problems GAs find hard. To be true, the hypothesis requires that there be a correlation between the static and observed fitnesses, since it is the observed fitness that counts. Whilst it is likely that *some* degree of correlation exists, this is not guaranteed, and it is also possible that the correlation will be weak. (See Section 2.3.6.)

Grefenstette (1993) showed that the SBBH was wrong via two counter-examples. The first was a problem that could be solved even though the SBBH predicted that it couldn't, and the second was a problem that couldn't be solved even though the SBBH predicted that it could.

The fitness function of his first example is

$$f(x_1, x_2) = \begin{cases} x_1^2 + 10x_2^2 & \text{if } x_2 < 0.995 \\ 2(1 - x_1)^2 + 10x_2^2 & \text{if } x_2 \geq 0.995 \end{cases}$$

Here, x_1 and x_2 are encoded as bit string of length 10, where $0 \leq x_i \leq 1$. The optimal solution is $(x_1, x_2) = (0, 1)$, which corresponds the bit string `00000000001111111111`. Because the fitness of 99.5% of the search space is determined by the first equation, in every schema partition defined over the first 10 positions, the schema with the highest static fitness is made up entirely of 1s. Despite this, Grefenstette reports that a standard GA has no trouble finding the correct solution, because once the second 10 bits have converged to $x_2 = 1$, "the *observed* fitness of schemas representing values of x_1 near 0 will change from low to high, and the GA will converge toward the global optimum" (Grefenstette, 1993, p. 82). As Grefenstette notes, this change in behaviour is predicted by the Schema Theorem, but not by the SBBH. The SBBH is not an extension of the Schema Theorem; it acts in opposition to it.

Grefenstette's second example has a fitness function of

$$f(x) = \begin{cases} x^2 & \text{if } x \neq 0 \\ 2^{11} & \text{if } x = 0 \end{cases}$$

where x is represented by a bit string of length 10, as before. This fitness function has a maximum at $x = 0$. The fitness at this point is so much greater than the fitness at any other, that—across all schema partitions—the schema with the highest fitness always contains the optimum. Despite this, GAs find this problem impossible. Even though the mean fitness provides the right hint, at every other point the fitness is misleading, and the GA is led away from the global optimum.

Though something like "building blocks" must form within the population over the course of a run—the solutions that are often found cannot be conjured from nothing, and for crossover and mutation to have aided the process solutions must be decomposeable into constituent parts—the Building Block Hypothesis does not constitute an adequate explanation of this process.

2.5 The Two-Way Onemax Problem

Neither the Schema Theorem nor the Building Block Hypothesis provides a very good account of what goes on inside a GA. In this section, we study one problem GAs find difficult to solve; in doing so, we hope to provide some feeling for the dynamics involved (the moves a GA does and doesn't make, and why). The example also serves to raise some questions about the utility of EAs in general.

2.5.1 Description

The (one-way) ONEMAX problem is to maximise the number of 1s in a bit string. (For this reason, it is sometimes known as the "bit-counting" problem.) The fitness function is usually linear in the number of bits equal to 1 such that the fitness of a given string is equal to the proportion of bits that are equal to 1:

$$f_{\text{1MAX}}(s) = \frac{\sum_{i=1}^{l} bit(s, i)}{l}$$

where l is the length of the bit string, and $bit(s,i)$ is the ith bit of string s. GAs find it very easy to find the optimal solution to the ONEMAX problem.

The Two-Way ONEMAX problem is only slightly different. If we define

$$f'_{\text{1MAX}}(s,a,k) = \frac{\sum_{i=a}^{a+k-1} bit(s,i)}{k} \text{ for } k > 0$$

such that $f_{\text{1MAX}}(s,a,k)$ is the proportion of ones in s between the locations a and $a+k$, then the Two-Way ONEMAX fitness function is

$$f_{\text{2MAX}}(s,a,k) = \begin{cases} 0.9 \times f'_{\text{1MAX}}(s,2,k) & \text{if } bit(s,1) = 0 \\ f'_{\text{1MAX}}(s,2k+1,k)^2 & \text{otherwise} \end{cases} \tag{2.7}$$

The Two-Way ONEMAX function conceptually involves a bit string divided into two equal sections, plus a special "switch" bit. If the switch bit is 0, then the fitness is $0.9 \times f'_{\text{1MAX}}(s,2,k)$, and so is entirely dependent on the content of the first section. If the switch bit is 1, then the fitness is $f'_{\text{1MAX}}(s,2k+1,k)^2$ and so is entirely dependent on the content of the second section. The function has a maximum of 1.0, which is achieved by any instance of the schema $\texttt{1*}\cdots\texttt{*1}\cdots\texttt{1}$. (A single 1, followed by k *'s, followed by k 1's.) On the other hand, the schema $\texttt{0*}\cdots\texttt{*}$ has a maximum fitness of 0.9.

GAs find this problem very difficult to solve. Initially, a bit string beginning with 0 (with half the bits set to 1) will have a fitness of 0.45, whilst a bit string beginning with 1 will have a fitness of 0.25. This gives the schema $\texttt{0*}\cdots\texttt{*}$ has higher fitness than $\texttt{1*}\cdots\texttt{*}$, and so instances of $\texttt{1*}\cdots\texttt{*}$ are quickly driven out of the population, even though the optimum is only attained if the first bit is 1. If k is small, a GA may be able to find the optimal solution if a near-solution can be found in the initial population. But as the length of the bit string increases, this becomes less and less likely, and the optimal solution becomes harder and harder to find.

2.5.2 Significance

The Two-Way ONEMAX problem is one that cannot[5] be solved by standard GAs. This is despite the fact that the standard ONEMAX problem is trivial to solve: though there is only one optimal string (out of 2^l possible strings), the search space is smooth, and well-suited to the sort of transformations carried out by (in particular) crossover. However, the Two-Way ONEMAX problem—the product of an apparently minor addition—is extremely difficult. The GA is misled early on, and once the "switching bit" becomes fixed in the population, the GA becomes caught in a local maxima from which it is unable to escape.

It is interesting exercise to try to come up with mechanisms that would allow the GA to escape this fate. The usual remedy is some sort of diversity maintenance scheme, but in this case, it is difficult to see how such a scheme could help. The problem manifests itself at the level of a single bit; it is difficult to see how problems that occur at this level can be side-stepped. (For example, it is not possible to sub-divide the population into 2^l sub-populations, one for each of the two values each bit can take.)

As we have seen, the Schema Theorem (but not so much the Building Block Hypothesis) tells us something about the dynamic behaviour of GAs. The Two-Way ONEMAX problem adds another pebble to our store of information: it shows us that whatever route the GA takes through the search space, the route has a particular bias that causes it to miss certain opportunities, and ultimately, solutions.

[5] We say "cannot" even though any GA that includes mutation can, in principle, find the optimal solution to any problem. (Given enough time, every bit string will be sampled.) But this is a monkeys-typing-Shakespeare argument that is of no practical value. To all intents and purposes, this is a problem GAs cannot solve.

2.6 Obstacles

In the introduction we said that EAs appear to hold great promise. They are based on and inspired by a system whose power is self-evident, and the results obtained so far have been encouraging, or at least have lent themselves to such an interpretation. Banzhaf et al. (1998, p. 4), for example, hope that one day Genetic Programming (GP) will take the job of human programmers: "The ability to enable computers to learn to program themselves is of the utmost importance in freeing the computer industry and the computer user from code that is obsolete before it is released."

We do not believe that EAs will become as successful as is hoped. The previous section provided one reason for thinking this way: if problems like the Two-Way ONEMAX problem can be so difficult, harder problems can only be more so. In this section, we give two additional reasons for our lack of confidence: "fitness" does not measure the quantity we want to measure; and preadaptation, though probably essential if we are to solve non-trivial problems, is largely impossible to achieve.

2.6.1 Fitness and Hard Problems

An individual's fitness is a measure of how good a solution that particular individual is. (That is, a measure of how "fit" it is.) For example, a lawn mowing program that manages to cover half the lawn might score 0.5; one that covers 40% of the lawn might score 0.4. (Note that fitness values are not necessarily ratio values. In general all one can say is that an individual with a fitness of 0.5 is better than an individual with a fitness of 0.4, but not that it is 25% better.)

In general terms, a GA works by repeatedly transforming a population of individuals into populations of successively fitter and fitter individuals until an admissible solution is found. At each generation, high-fitness individuals are chosen to be parents, under the assumption that the children of high-fitness individuals will be of higher fitness than the children of low-fitness

parents.

It is probably true that there is a hereditary component to fitness. However, the approach makes another assumption that is not so likely to be true: that high-fitness individuals are "closest" (in terms of the number of transformations required—crossovers, mutations, etc.) to the admissible solution. In other words, GAs assume that if we are looking for an individual of fitness 1.0, an individual of fitness 0.5 is a better starting point than one of fitness 0.4. This reasoning will not always be correct. A local maxima, for example, is a point that appears to be more promising than any of its neighbours (it is more fit than they) without actually being so.

In other words, fitness is used as if it measured the *distance to solution* (i.e., as if it were a search *heuristic*) even though it actually measures the *goodness of solution*. (Despite general ignorance of the distinction, the substitution usually turns out to be a matter of necessity not choice: it is extremely difficult to calculate the distance between one point in the search space and another.) If this arrangement is to be effective, there must be a strong negative correlation between the two metrics. (As fitness increases, the distance must decrease.) For many problems there is such a correlation—in the case of the ONEMAX problem, for example, there is a perfect correlation between the two—but no such correlation is required to exist, nor is it required to be strong. If *goodness of solution* is not a good estimate of *distance to solution*, then the GA can easily find itself in the search space equivalent of a cul-de-sac.

We contend that as problems get harder, fitness provides less and less useful guidance, because the disparity between it and the distance to solution becomes greater. Performance is far easier to measure than potential performance. Performance can often be measured directly: how much dust the vacuum collected, how many cities were visited, how much silicon was used. The potential—how an individual's child's child's child will perform—is more difficult to measure.

On test functions where the global optimum is known, the Hamming dis-

tance between a point and the optimum can be used to estimate *distance to solution*. This approach was taken by Jones and Forrest (1995), who calculated the correlation between fitness and (Hamming) distance for a number of common test functions. They found that their Fitness Distance Correlation (FDC) was often able to predict how a GA would perform, classifying "easy deceptive problems as easy and difficult non-deceptive problems as difficult" (Jones and Forrest, 1995, p. 184). This result supports our argument that difficult problems are those where the correlation between the two metrics is weak.

(Note that fitness is one aspect of EAs that has no natural analogue.[6])

2.6.2 Preadaptation

A preadaptation is "a feature that fortuitously serves a new function" (Futuyma, 1998, p. 355). In nature, it often happens that a feature designed for one purpose turns out to be useful for another. For example, the kea, a bird indigenous to New Zealand, has a strong, sharp, beak that enables it to feed on fruits and seeds. Though this beak evolved to serve one specific purpose, it proved useful for another: when sheep were introduced to the country, the kea found that it was able to use its beak to pierce the skin on their backs, giving it direct access to the fat underneath. (From Futuyma, 1998, p. 355.)

Preadaptation occurs when a feature that originally arose to do job A comes to do a completely different job B. Jobs A and B can be quite different:

> Many of the proteins in the eye lens, for example, begin their careers doing something completely different and unrelated to vision. ... My current favorite example (of many available) is the discovery that a gene complex originally involved in specifying the pattern of segmentation in insects has now been found to assist in

[6]Biologists do refer to "fitness," but they mean a different thing by it. To a biologist, an individual's fitness is its *actual* contribution to the population, not the *expected* contribution.

> the proper development of the vertebrate hindbrain (a structure that has no counterpart in segmented insects). (Dorit, 1997)

If dimension-jumping preadaptions such as these are both indispensable and common, EAs are in for some trouble. Suppose that the only way to evolve a bird that eats fat from the backs of sheep (job A) is to first evolve a bird that can eat seeds (job B). Under this scenario, an EA employing a fitness function that scores individuals according to their skill at doing job A may never be able to evolve an individual to do this job. Only if the fitness function rewards the right intermediate step will an individual evolve to do job B, from where it can reach A. The problem is that the correct intermediate step is not known.

One hypothesised example of preadaptation is particularly relevant to the efforts being made to solve human-level problems with EAs: Calvin (1994, p. 79) argues that "language, foresight, musical skills and other hallmarks of intelligence are connected through an underlying facility that enhances rapid movements." In other words, it was initially the facility to plan ballistic movements (hammering, clubbing, throwing, etc.) that was selected for. Later, he says, the planning skills required to perform and execute these actions became the basis for language and intelligence. If Calvin is correct (and if this is the only way in which intelligence can evolve), it might be that in order to evolve a chess-playing program, we first need to evolve a program that can throw rocks at lions. Then, only once rock-throwing has been mastered, is the fitness function changed to measure chess playing prowess, with the result that a chess-playing program ultimately evolves. But it is difficult to imagine a fitness function that could anticipate and negotiate such a circuitous route.

The problem preadaptation poses for EAs is that: (a) it seems likely that many useful structures arose via this mechanism; (b) it follows that artificial systems would exhibit similar behaviour; and (c) there is no analogous process in EAs, nor can there be.

Note that a close relative of EAs—systems that implement "artificial life"

(A-life systems) via an evolutionary approach—*do* permit preadaptation. To describe them briefly: in basic operation, they operate much the same as EAs—there are individuals, there are generations, there are operations like crossover and mutation, etc. However, there is usually some sort of interaction between individuals, and the individuals live in—and have to interact with—a particular "environment." Moreover (and this is the big and crucial difference), there is no fitness function. Instead, individuals prosper or perish according to the success of their interactions with other individuals and their environment: survival is all that counts. Since there are usually many more ways to survive than there are ways to satisfy a fitness function, A-life systems would seem to support the sorts of evolutionary side-steps that result in preadaptations. However, without an explicit fitness function, the system can't readily be coerced into solving any one problem, with the result that A-life systems cannot easily be used in the situations EAs are intended for.

Chapter 3

Selection

3.1 Introduction

An EA repeatedly performs three distinct operations: fitness evaluation, selection, and reproduction. First, it calculates the fitness of every member of the population. Second (by looking at fitness only), it determines which individuals will reproduce, and how many times they will be allowed to reproduce. Third, it generates a new population by crossing over and mutating the selected individuals.

The second step—deciding which individuals reproduce—is performed by the selection scheme (or selection function). The aim is to identify those individuals that are most likely to yield individuals of ever-higher fitness. If there is a positive correlation between the fitness of parents and that of their offspring,[1] then (broadly speaking) the selection scheme needs to serve up individuals of high fitness.

Since it deals with fitness only—a single real value—selection schemes stand completely independent of the remainder of the EA. This makes selection a particularly rewarding topic of study, since any results obtained can immediately be put to use across the entire range of EAs.

[1] If there is not, then performing fitness-based selection makes no sense; random search would be just as good.

We will attempt to describe the desirable properties of selection schemes later; we next provide some examples of the many selections schemes that have been developed, followed by some descriptive measures.

3.2 Some Selection Schemes

3.2.1 Fitness-Proportional Selection

Under fitness-proportional selection, the number of times each individual reproduces is proportional to its fitness. If the population size is to remain the same, then

$$p_i = \frac{f_i}{f_\mu}$$

where p_i is the probability of selecting an individual, f_i is its fitness, and f_μ is the mean fitness.[2]

For about 15 years, fitness-proportional selection was the only sort of selection used. This was so for two reasons: first, it seemed, intuitively, to be the right thing to do, and second, it guaranteed that an exponentially-increasing number of offspring be allocated to the fittest individual, as required by the Schema Theorem.

Fitness proportional selection has no parameters.

3.2.2 Fitness-Proportional Selection with Fitness Scaling

This selection scheme is very similar to straight fitness-proportional selection. Instead of objective fitness being used directly, it is first transformed by a linear (or non-linear) function. This transformed fitness is then used as if it were the actual fitness. The transformation function is used to either

[2]Fitness-proportional selection is also known as roulette-wheel selection, as its way of allocating offspring is identical to that obtained by selecting individuals on the basis of spinning a roulette wheel divided into segments representing individuals where the size of each segment is proportional to the individual's objective fitness.

emphasize (or de-emphasize) differences in objective fitness. Although any function will do, it is not likely to be very useful unless it is monotonically increasing, such that the fittest individual receives the most offspring.

One common technique is "sigma scaling," which incorporates information about the population mean and variance. An example of sigma scaling is

$$p_i = \frac{1}{N}\left(1 + \frac{f_i - f_\mu}{2\sigma}\right)$$

where p_i is the probability that individual i will be selected, f_i is its fitness, f_μ is the mean fitness of the population, σ is the variance, and N is size of the population. This function allocates 1.5 offspring to individuals whose fitness is one standard deviation above the mean.

3.2.3 Rank Selection

In rank selection, individuals are first sorted according to their objective fitness. Their objective fitnesses are then discarded, and rank alone is used to determine how many times each individual gets to reproduce. Rank selection requires a function that maps ranks to expected offspring counts. Again, the function employed can be almost anything. Banzhaf et al. (1998, p. 132) give two examples. A linear function:

$$p_i = \frac{1}{N}\left(\eta^- + (\eta^+ - \eta^-)\frac{i-1}{N-1}\right) \tag{3.1}$$

where i is the individual's rank, η^-/N is the probability of the worst individual being selection, and η^+/N the probability of the best individual being selected. For the population size to stay constant, $\eta^- + \eta^+ = 2$ must hold.

And an exponential function:

$$p_i = \frac{c-1}{c^N - 1}\, c^N - i$$

where $0 < c < 1$.

3.2.4 Truncation Selection

To perform truncation selection, the individuals are first ranked in order of fitness. Only the TN best individuals are allowed to reproduce, where $0 < T \leq 1$ and N is the population size. The chosen individuals mate randomly, with each having an equal chance of being selected. This scheme mimics the breeding technique employed by, for example, plant breeders.

3.2.5 Tournament Selection

Tournament selection is a two-step process that involves first, randomly selecting a small number of individuals, t, (typically less than ten) and second, finding the fittest of this subset. Tournament selection lends itself to parallel implementations since it requires no global tallying.

3.2.6 Minimising the Stochastic Error

Baker (1987, p. 14) observed that selection is often performed in two stages: "(1) determination of the individuals' expected values; and (2) conversion of the expected values to discrete numbers of offspring." The first three selection schemes described are of this type: for each rank, they prescribe expected values. (Although they do not say how the expected values should be converted into actual offspring counts.)

It is the job of the first stage to establish that, for example, one particular individual should expect to get 3.6 offspring in this generation, that another should get 0.3, and so on. That is, the first stage defines a probability distribution. The second stage samples from this probability distribution, in this case resolving, in the first instance, the impossibility of 0.6 children by deciding if this particular individual should actually have 3, or 4, or some other number of offspring this generation. The second stage is thus concerned with the "stochastic error" made whilst sampling.

Baker proposed that the sampling algorithm should be evaluated according to three criteria:

bias "the absolute difference between an individual's actual sampling probability and his expected value."

spread the difference between the minimum and maximum possible offspring counts allocated to each individual. "Minimum spread" is defined as the "smallest possible spread which theoretically permits zero bias." That is, a spread of $[\lfloor v \rfloor, \lceil v \rceil]$, where v is the expected number of offspring. Spread is related to the offspring count variance.

efficiency the sampling algorithm should not increase the GAs overall time complexity.

Goldberg (1989, p. 115ff.) surveys some different sampling algorithms. Most work by initially allocating to each individual the whole number of offspring that they are due. They then play around with the fractional parts. As Baker (1987, p. 14) observed, all of these methods "fail to provide both zero bias and Minimum Spread." This led him to develop Stochastic Universal Sampling (SUS), a method that overcame these deficiencies.[3]

Baker's algorithm is both simple and efficient. Given a set of n individuals and their associated expected values, SUS effectively arranges them on a roulette wheel, where the size of each slice is proportional to the expected value (as with roulette-wheel selection). Next, a second wheel marked with n equally spaced pointers is placed over the first, which is then spun. The pointers then indicate which individuals are to reproduce, and the number of times they are allowed to do so. This approach yields the sought-after characteristics of minimum spread and zero bias since no less than $\lfloor v \rfloor$ and no more than $\lceil v \rceil$ pointers can fall over a slice of size v.

[3] I am unable to explain why Goldberg's book, published in 1989, discusses the problems with existing sampling algorithms, but does not address Baker's 1987 solution. Baker's paper does appear in the Bibliography, but I am unable to find where in the book it is cited. It does not appear to be mentioned in any of the sections relevant to selection.

3.3 Characterising Selection Schemes

Several different selection schemes are described above. They all perform the same function—allocating offspring to individuals—but they have quite different ways of doing so, and have quite different effects. In this section, we present some of the work that has gone into quantifying these differences.

Over the last decade, significant contributions have been made by Goldberg and Deb (1991) and Bäck (1994), but the most authoritative and comprehensive results are those of Blickle and Thiele (1997). The following section rests heavily on their work.

Initially, selection schemes were compared by means of the effect they had on a population made up of one more-fit individual mixed in amongst many (equally) less-fit ones. The principal measure was *takeover time*, which was the number of generations required for the population to converge on (descendants of) the more-fit individual. Goldberg and Deb (1991) list the takeover times of several different selection schemes.

This fairly crude approach has recently been superseded by measures that describe the way in which selection schemes transform a population of normally-distributed fitnesses. (This approach has been used by population geneticists for more than 30 years—see, for example, Crow and Kimura (1970, p. 225).) As well as being a more realistic model of the fitness distribution, it also allows for a wider variety of descriptive measures.

Three measures are described here: reproduction rate, selection intensity, and loss of diversity. Between them they capture a great deal of how an EA responds to selection. They can only be generated for the rank-based selection schemes (tournament, truncation and ranking), but as these approaches now dominate, this is not a serious shortcoming.

3.3.1 Reproduction Rate

The *reproduction rate* is the ratio of the number of individuals with a certain fitness value before and after selection. That is, the expected number of

offspring born to an individual of a certain fitness (or rank). The graph of reproduction rate versus fitness (or rank) is known as the *selection profile*. Table 3.1 gives the reproduction rate of three selection schemes (after Blickle and Thiele, 1997, p. 56); the corresponding selection profiles are plotted in Figures 3.1, 3.2, and 3.3.

Selection Method	Reproduction Rate
Tournament	$R_T(t,r) = N\left(\left(\frac{N-r+1}{N}\right)^t - \left(\frac{N-r}{N}\right)^t\right)$
Truncation	$R_\Gamma(T,r) = \begin{cases} 0 & \text{if } r > TN \\ 1/T & \text{otherwise} \end{cases}$
Linear Ranking	$R_R(\eta^-,r) = \frac{N\eta^- - 1}{N-1} + \frac{1-\eta^-}{N-1}(2(N-r)-1)$

Table 3.1: Reproduction rate of three different selection schemes. N is the size of the population, r is the rank ($1 \leq r \leq N$, 1 is best), t the tournament size, T the proportion of individuals that get to reproduce, and η^-/N is the probability that the worst individual will be selected.

3.3.2 Selection Intensity and Variance

Selection intensity is the expected *change in mean fitness* from one generation to the next. A normally-distributed population of fitnesses, $N(0,1)$, is used for comparison purposes. Table 3.2 lists the selection intensity of three different selection methods (from Blickle and Thiele, 1997, p. 62), and Table 3.3 the corresponding post-selection *fitness distribution variance*. (The variance is 1 before selection.)

3.3.3 Loss of Diversity

Loss of diversity is simply the number of individuals that do not get to reproduce, or equivalently, the proportion of individuals not picked during

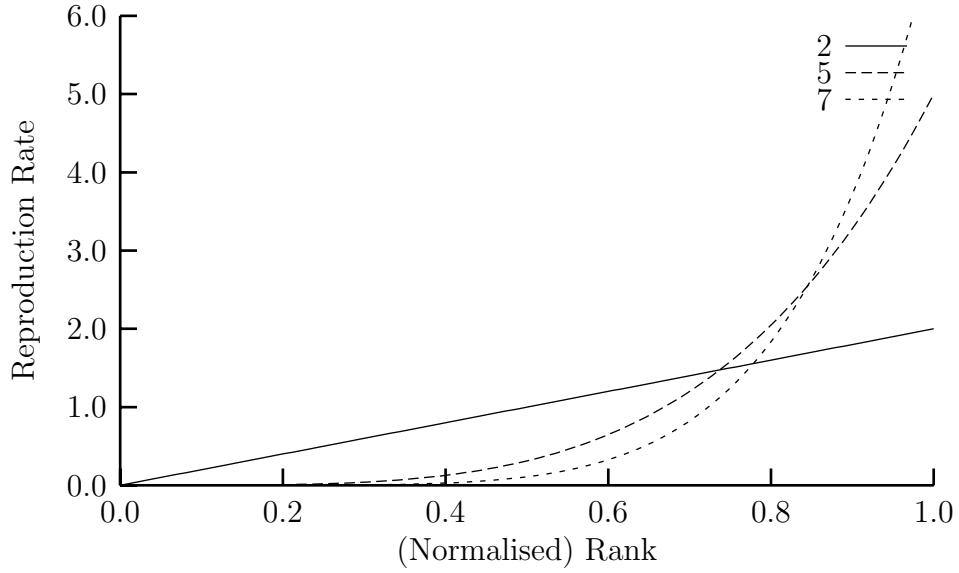

Figure 3.1: Tournament Selection selection profile, for different values of t. N is "large"; for typical values of N ($N > 50$), the difference is less than 5%. Fitness increases as rank does.

Selection Method		Selection Intensity
Tournament	$I_T(t) =$	$\sqrt{2(\ln t - \ln(\sqrt{4.14 \ln t}))}$
Truncation	$I_\Gamma(T) =$	$\frac{1}{T}\frac{1}{\sqrt{2\pi}} \exp(-f_c^2/2)$
Linear Ranking	$I_R(\eta^-) =$	$\frac{1-\eta^-}{\sqrt{\pi}}$

Table 3.2: Selection intensity of three different selection schemes. f_c is determined by $T = \int_{f_c}^{\infty} \frac{1}{\sqrt{2\pi}} \exp(-f^2/2)\, df$. For tournament selection the result is an approximation.

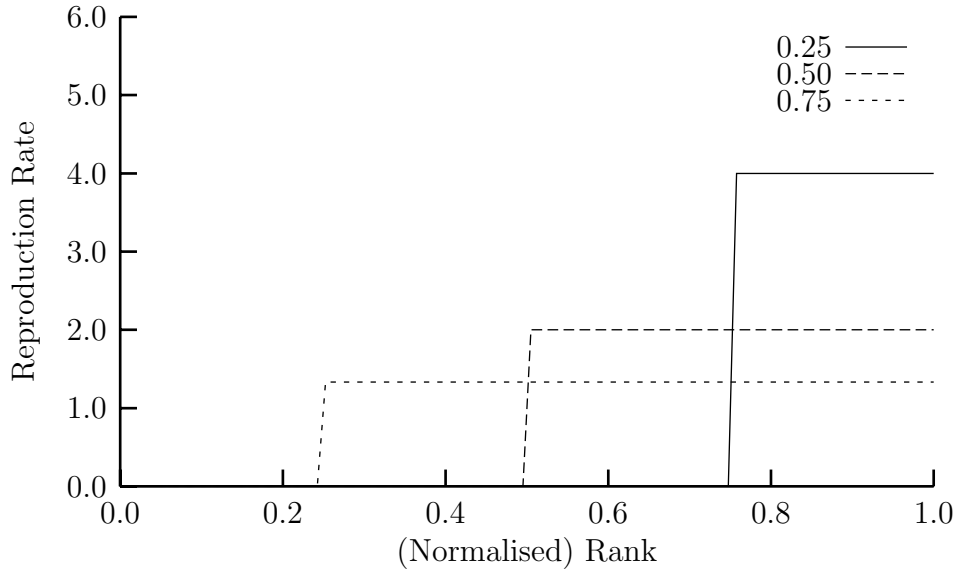

Figure 3.2: Truncation Selection selection profile, for different values of T. N is "large"; for typical values of N ($N > 50$), the difference is less than 5%. Fitness increases as rank does.

Selection Method		Selection Variance
Tournament	$\Phi_T(t) =$	$\frac{0.918}{\ln(1.186 + 1.328t)}$
Truncation	$\Phi_\Gamma(T) =$	$1 - I_\Gamma(T)(I_\Gamma(T) - f_c)$
Linear Ranking	$\Phi_R(\eta^-) =$	$1 - I_R(\eta^-)^2$

Table 3.3: Selection Variance. For tournament selection the result is an approximation.

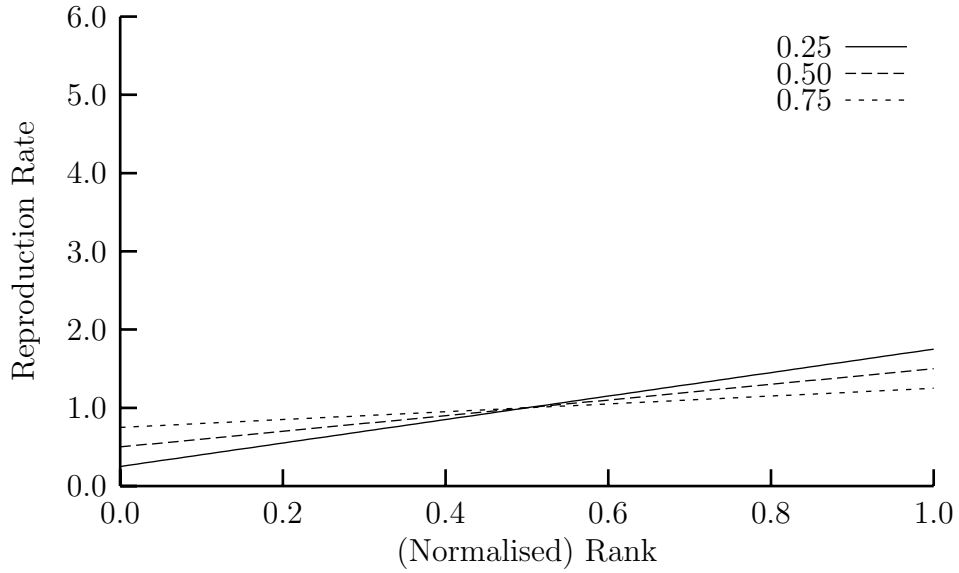

Figure 3.3: Linear Ranking selection profile, for different values of η^-/N. N is "large"; for typical values of N ($N > 50$), the difference is less than 5%. Fitness increases as rank does.

the selection phase. The amount of diversity lost is related to the selection intensity—as selection intensity increases, it becomes more and more difficult to become a parent. Table 3.4 lists the loss of diversity for three different selection schemes (from Blickle and Thiele, 1997, p. 65). Note that this measure is independent of the fitness distribution.

Selection Method		Loss of Diversity
Tournament	$D_T(t) =$	$t^{-\frac{1}{t-1}} - t^{-\frac{t}{t-1}}$
Truncation	$D_\Gamma(T) =$	$1 - T$
Linear Ranking	$D_R(\eta^-) =$	$\frac{1-\eta^-}{4}$

Table 3.4: Loss of diversity of three different selection schemes.

3.3.4 Graphs of Selection Schemes

The selection intensity, selection variance and loss of diversity for each the three selection schemes for which they have been calculated have been plotted below: tournament selection in Figure 3.4, truncation selection in Figure 3.5 and linear ranking in Figure 3.6. Note that tournament selection is "backwards": selection intensity decreases toward the right, in contrast to the other two schemes where selection intensity increases as the relevant parameter does.

Linear ranking selection is not capable of exerting much selection pressure: the most selection intensity it can muster is only slightly more than 0.5. For the same selection intensity, truncation selection exhibits a greater loss of diversity, and a lower variance than tournament and ranking selection. There is no obvious relationship between selection intensity, selection variance, and loss of diversity.

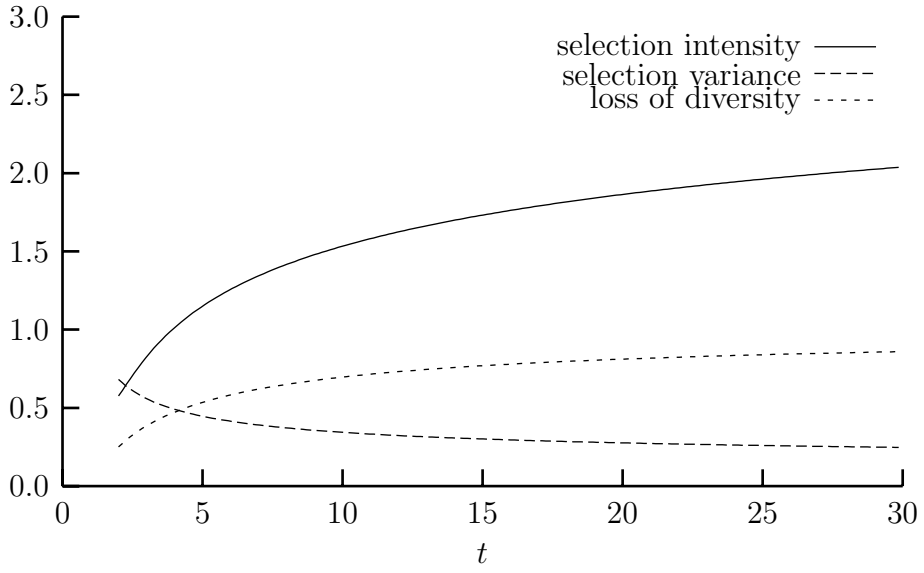

Figure 3.4: Tournament Selection. t is the tournament size.

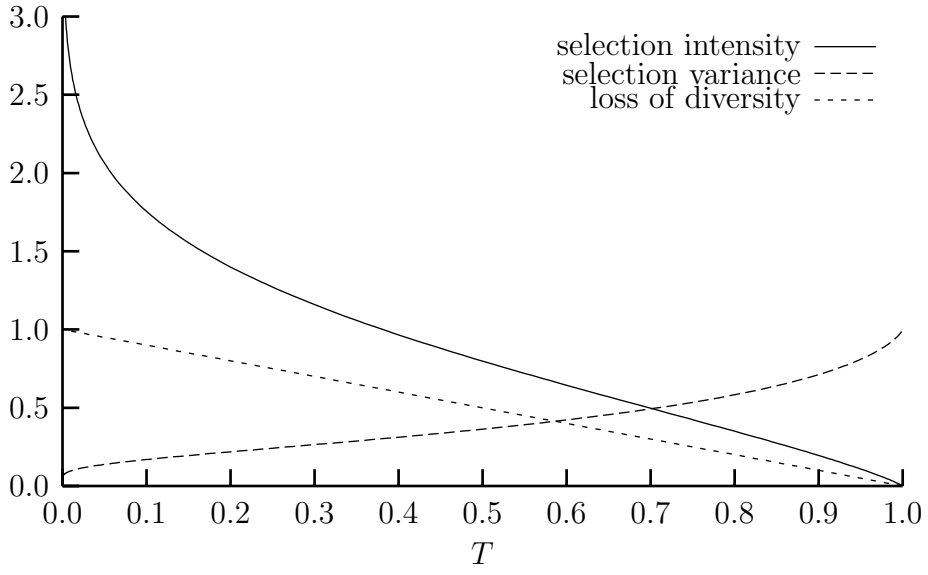

Figure 3.5: Truncation Selection. T is the proportion allowed to breed.

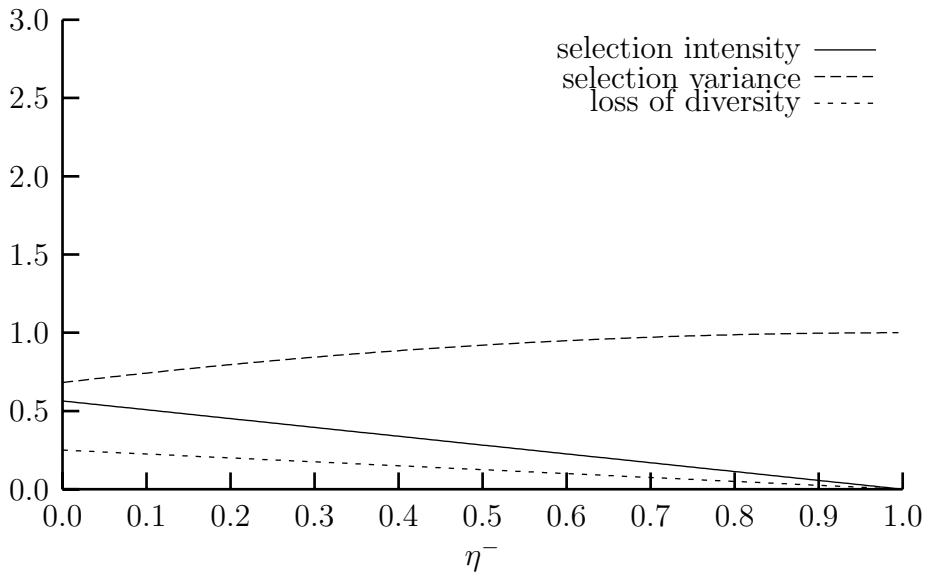

Figure 3.6: Linear Ranking Selection. η^-/N is the probability that the worst individual will be selected.

3.4 Choosing a Selection Scheme

For any given problem, how do we decide which selection scheme to use? This question is often answered with a discussion that begins and ends with the observation that both too much, and too little selection pressure is bad: too much pressure produces too high a rate of convergence, potentially causing the population to converge on the wrong region of the search space, and too little produces too low a rate of converge such that the solution may never be found. For example, Goldberg and Deb (1991, p. 87) say that the right approach "might be to slow down convergence enough so that errors are rarely made" (p. 88) *or* it might be to ensure "very high growth ratios ... by grabbing those building blocks you can get as fast as you can, thereafter restoring the missing building blocks through mutation." (p. 88). They conclude: "The question [of which selection scheme should be used] is a difficult one, and despite limited empirical success in using this method or that, a general answer remains elusive."

Mitchell (1996, p. 166) writes: "Below I will describe some of the most common methods ... these descriptions do not provide rigorous guidelines for which method should be used for which problem; this is still an open question for GAs." Banzhaf et al. (1998, p. 129–130) rightly say that "the method of selection to be applied under particular circumstances is one of the most important decisions to be made in a GP run," but then go on to provide no guidance at all to those facing this decision, aside from very weakly recommending tournament selection, primarily because it is easy to implement: "Tournament selection has recently become a mainstream method for selection, mainly because it does not require a centralized fitness comparison between all individuals. ... With fitness-proportional selection, the communication overhead between evaluations would be rather large." (Banzhaf et al., 1998, p. 133)

Though much work has gone into devising and characterising selection schemes, no-one knows when one selection scheme is to be preferred over another: nothing seems to guide practitioners as much as trial and error. We do not even know if it is beneficial for the sampling algorithm to exhibit zero bias and (more suspiciously) minimum spread, as Baker (1987) baldly states. (Sampling is discussed further in Section 4.2.)

Should the actual offspring count be as close as practicable to the expected count? Nature does not appear to support this hypothesis: in some natural breeding groups, brood sizes vary wildly from one individual to another, exhibiting much more variation than can easily be explained away by differences of phenotype or behaviour. A more likely explanation for the variance is that the correlation between genotype and reproductive success is low such that chance plays a large part in determining which individual becomes the (say) alpha male. If this is the case, we do not have a situation of minimal spread.

Many of the selection schemes described above are really sets of selection schemes since their behaviour varies according to the value of some parameter. Bäck (1994, pp. 57–58) gives three desirable properties of such sets:

1. "The impact of the control parameter(s) on selective pressure should be simple and predictable."

2. "One single control parameter for selective pressure is preferable."

3. "The range of selective pressure that can be realized by varying the control parameter should be as large as possible."

These criteria are reasonable. However, whilst they give us reasons to choose one set over another, they do not give us any reason to chose any one member of the set over another.

Chapter 4

Selection Profiles

4.1 Introduction

About the only thing that selection schemes appear to have in common with each other is their function, or behaviour. Selection schemes are passed a list of individuals; they are required only to return a subset of that list to serve as parents of the next generation. This is what defines a selection scheme—nothing at all is said about *how* it must make its selections. (This is why selection schemes are able to be, in terms of the way they make their selections, quite different from each other.) Nevertheless, despite their seemingly dissimilar internals, every rank-based selection scheme is almost completely described by its selection profile—a selection scheme can be thought of as nothing but a function that maps ranks to offspring counts.[1]

There are two flavours of selection scheme: selection schemes with "memory" and selection schemes without. Memoryless selection schemes "select" (return) one individual at a time, without any knowledge of previous selections. Because of this, memoryless schemes can be used to select a small number of individuals at a time. Some EA systems—such as those employ-

[1]Selection schemes that directly use the fitness of an individual—rather than its relative ranking—cannot be represented in this way. We shall ignore such schemes for the remainder of this chapter.

ing steady-state evolution, in which only a part of the population is replaced at any one time—require a selection scheme with this ability. Tournament selection is an example of a memoryless selection scheme.

In contrast, selection schemes with memory select an entire population at once, knowing exactly which individuals have and haven't been selected. This approach gives much greater control over the selection process, at the cost of some flexibility. Truncation selection is an example of a selection scheme with memory.

Both types of selection scheme have a selection profile, since it is possible to calculate (or compute) the expected number of individuals allocated to each rank under both. However, only memoryless selection schemes are *uniquely* determined by their selection profile: two selection schemes possessing memory may have the same selection profile, but differ in the variance of the offspring count.

To illustrate, suppose one particular rank has an expected offspring count of v. Under one selection scheme, it might turn out that the actual offspring count can be modelled by an exponentially-distributed random variable; under another, the actual count might be restricted to either $\lfloor v \rfloor$ or $\lceil v \rceil$. Such variation is only possible, however, if the selection scheme has "memory"—under a memoryless selection scheme, the variance is strictly a function of the expected offspring count.

In this chapter we investigate and explore the idea that a selection scheme is completely defined by two things: its selection profile and its variance (or spread). As implied by the discussion above, this means that the study does not consider (memoryless) selection schemes that are used to select anything less than a full generation at a time; nor can it include non-rank-based selection schemes. Fortunately this does not exclude the most popular selection schemes. We first examine the role of spread in the success (or otherwise) of selection schemes. Second, we use a meta GA to evolve selection profiles in an attempt to discover which selection profiles suit which problems.

4.2 The Importance of Spread

4.2.1 Introduction

Figure 4.1 depicts the selections made by tournament selection ($t = 5$) on one particular run. (Compare this to Figure 3.1, a graph of the analytic solution.) It is clear that tournament selection does not exhibit minimal spread: though the most fit individual is expected to have exactly 5 offspring, tournament selection often gives it several more, or several less.[2]

Tournament selection is a popular and effective selection scheme. Is it successful because of the level of spread it generates, or in spite of it? This experiment provides an empirical answer to the question.

4.2.2 Method

Selection profiles map ranks to expected numbers of offspring. This figure is a real number—at each generation, the *expected* number of offspring (a real value) must be converted into the *actual* number of offspring (an integer value) by a sampling algorithm. The sampling algorithm is allowed to dole out whatever actual values it likes as long as the appropriate expected value is maintained. Consequently, the sampling algorithm determines the variance (and spread).

In this experiment, we shall compare three selection schemes: (1) stardard tournament selection; (2) a selection scheme based on the tournament selection profile, but using with SUS sampling; and (3) a scheme based on the tournament section profile, but using EXP sampling.

SUS sampling is explained in Section 3.2.6. Figure 4.2 shows the selec-

[2]In fact, tournament selection scores quite poorly on the minimal spread criterion. If the entire population has exactly the same fitness, for example, then tournament selection is equivalent to fitness proportional selection (or any other scheme that picks individuals randomly, and with replacement). In turn, this means that only about 63% of the individuals will be selected to reproduce, since $\lim_{N\to\infty}(1-(1/N))^N = e^{-1} \approx 0.37$. (Under minimal spread, each individual would be selected exactly once.)

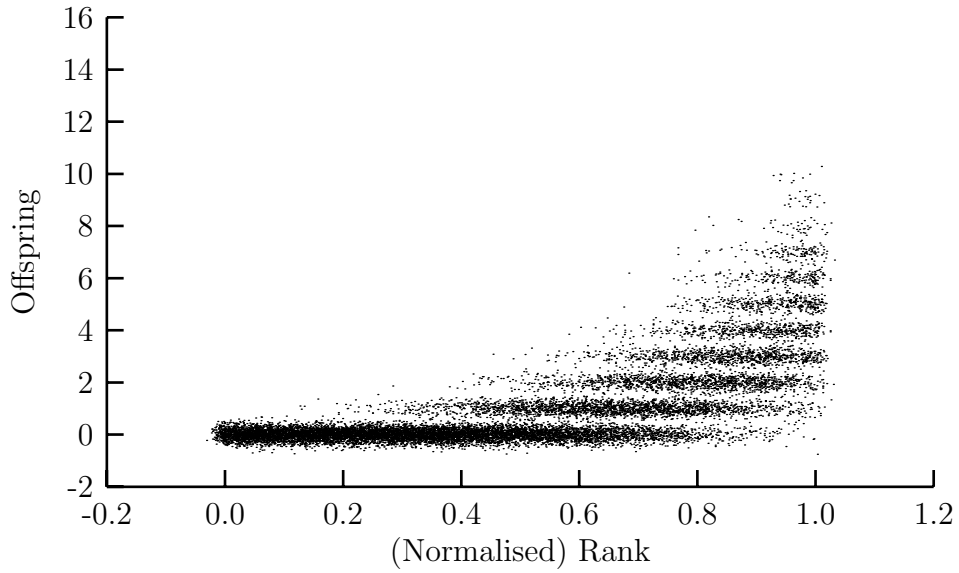

Figure 4.1: Tournament selection: The selections made by tournament selection ($t = 5$). The population size is 50; 300 runs are represented. Each point represents the number of offspring an individual of a particular rank happened to have on one specific run. (There are 300 sets of points, one for each run.) A small amount of noise has been added to each of the points to separate them. See also Figure 3.1, a graph of the analytic solution.

tions made by a tournament-based algorithm employing SUS as the sampling algorithm. SUS produces very little variance.

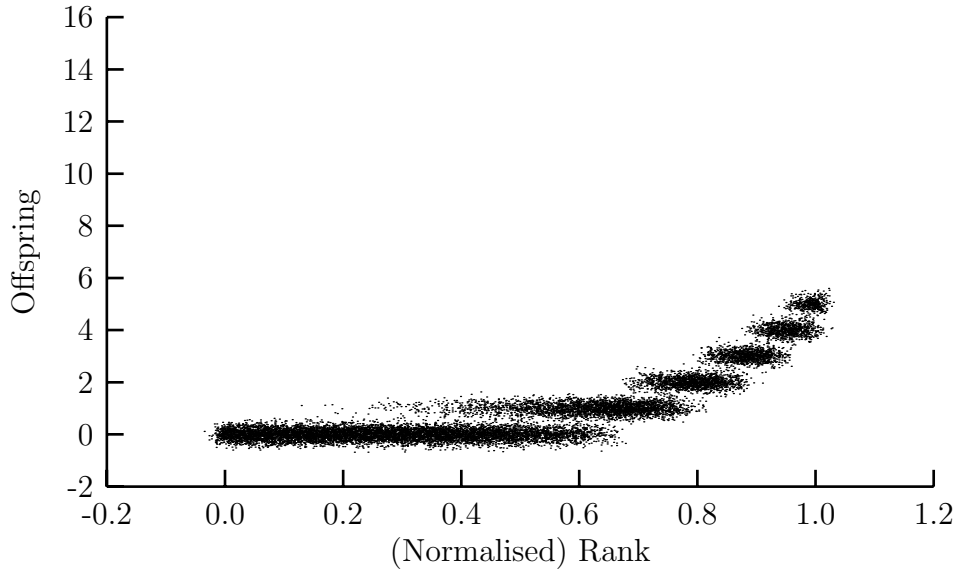

Figure 4.2: Tournament/SUS selection: The selections made by a selection scheme with the same selection profile as tournament selection, but with SUS (Section 3.2.6) as the sampling algorithm. (See Figure 4.1 for further details.)

EXP sampling generates (close to) exponentially distributed actual offspring counts. It is performed as follows. We start with an array of offspring counts, where the value at index i is the expected number of offspring allocated to the individual at rank i. Each offspring count is then replaced by a sample from an exponentially-distribution random variable with the same mean. After each rank's count has been perturbed in this way, the sum is found, and the counts are scaled such that the total is reset to whatever it was previously. The resulting rank/offspring count array is passed to SUS, which converts the array of (real) expected values into (integral) actual values. Figure 4.3 shows the selections made by the EXP sampling algorithm. EXP sampling exhibits a large variance.

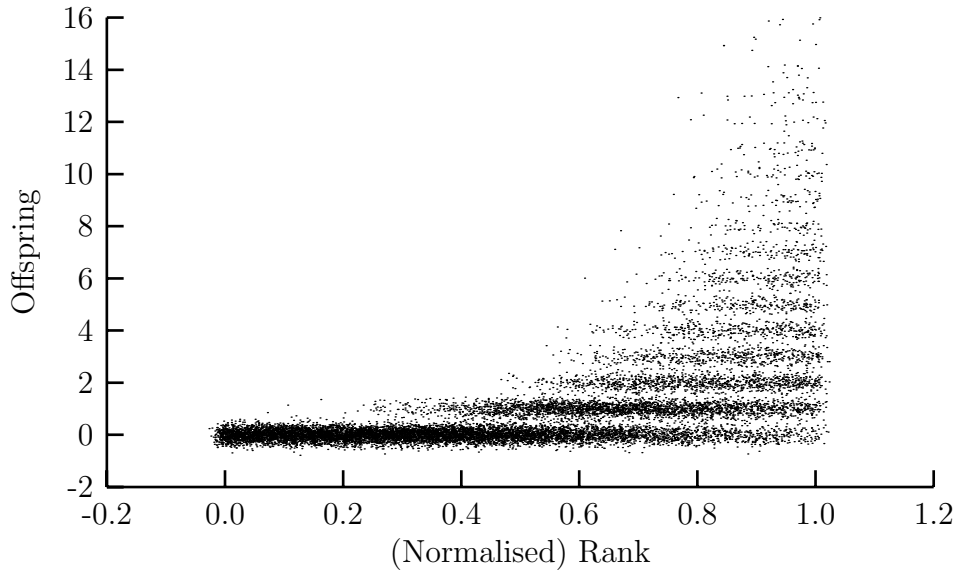

Figure 4.3: Tournament/EXP selection: The selections made by a selection scheme with the same selection profile as tournament selection, but with EXP as the sampling algorithm. Note that some points did not fit onto the graph. (See Figure 4.1 for further details.)

4.2.3 Results and Discussion

The mean number of generations required to find the optimal solution for several common test problems is shown in Table 4.1. On each occasion, tournament and SUS selection recorded roughly equivalent figures (the difference between the two was less than the standard error), whilst EXP proved slightly less effective.

This experiment involved selection schemes with both extremely low and extremely high variance. Little difference was observed between the two, which suggests that the variance is largely irrelevant. We therefore cannot conclude that minimal spread is a desirable property of selection schemes.

Problem	Tournament	SUS	EXP
ONEMAX[a]	10.600 (0.196)	10.460 (0.172)	11.560 (0.190)
DEJONG3[b]	10.260 (0.176)	10.200 (0.178)	10.580 (0.198)
Royal Road[c]	27.860 (3.100)	30.240 (3.118)	40.320 (3.614)

[a]See Section 2.5.1—the genome is 50 bits long. Population = 100, crossover = single-point (uniform), mutation = 0.01, runs = 50.

[b]Variant of DeJong's third test function—10 real values are used instead of 5; see Goldberg (1989, p. 108). Population = 100, crossover = Gaussian blend (Section 4.3.2), runs = 50.

[c]Variant of Royal Road test function $R1$—32 bits are used instead of 64; see Forrest and Mitchell (1993). Population = 128, crossover = single-point (uniform), mutation = 0.05, runs = 50.

Table 4.1: The mean number of generations (standard error in parentheses) required for three different selection schemes to find the optimal solution to several different problems. Tournament selection was performed with $t = 7$; SUS and EXP have an equivalent selection profile (but different variance).

This result seems to contradict Baker (1987) who argues that minimal spread is a desirable property of selection schemes (Section 3.2.6). Baker provides very little evidence for this claim, simply saying that "All of the GAs' theoretical support presupposes the ability to implement the intended

expected values" (Baker, 1987, p. 14). This statement is puzzling since the theoretical support referred to can only be the Schema Theorem, and the Schema Theorem is expressly defined in terms of expected values; it considers the variance irrelevant.

A slavish devotion to the expected values is odd for other reasons too. First, fitness itself (whence the expected values come) is subject to noise from several sources: it is only an estimate of the value we want (Section 2.6.1) and in addition, fitness functions are often non-deterministic. Second, it bespeaks a strange prejudice against one source of randomness. GAs employ random numbers almost everywhere. Why should selection not be random too?

4.3 Evolving Selection Profiles

4.3.1 Introduction

A selection scheme is completely described by two things: its selection profile,[3] and its variance (spread). In the previous section, we demonstrated that the variance does not appear to have much influence on the final outcome. (At least if it is not extremely high.) Consequently, the selection profile on its own can be considered an almost complete description of the selection scheme. In turn, this means that we can mount a study of selection schemes via their selection profiles.

A meta GA uses one GA (the *outer GA*) to control the parameters of another (the *inner GA*). Meta GAs are often used to optimise parameters

[3] "Reproduction rate" and "selection profile" are practically synonymous terms, since the selection profile is merely a graph of the reproduction rate. As terms relating to selection schemes, however, we feel that the latter is preferable (at least for the purposes of this section), since it prompts one to think of selection schemes in visual, rather than symbolic terms. Selection is a simple process, even though selection schemes may be described by a passel of symbols. Thinking of selection schemes as nothing more than graphs restores some of their inherent simplicity.

such as population size and crossover and mutation rates (e.g., Grefenstette, 1986). In this section we use a meta GA to evolve selection profiles.

4.3.2 Method

Representation

Effective selection profiles are likely to be both relatively simple and smooth. Indeed, we would be surprised if anything other than a monotonically increasing selection profile (the higher the (normalised) rank, the higher the offspring count) turned out to be the most effective. In addition, evolving an approximation to such a function must rate as one of the easiest tasks an EA has ever been set. Accordingly, we shall employ a very simple and transparent representation.

We chose to represent the selection profile by a vector of (real) numbers, $\vec{v} = \{v_0, v_1, \ldots, v_k\}$, where $v_i \geq 0$. The values are interpreted as the value ("height") of the selection profile at k equally-spaced points. (The least-fit individual can expect to get v_0 offspring, the most-fit v_k, etc.) To generate intermediate values, we fit a straight line to the nearest two points and interpolate.

The values that make up the vector are stored not as bit strings, but "directly" as floating point numbers. We believe this is the most appropriate way for a GA to encode real numbers (see Michalewicz, 1996; Salomon, 1996).

Genetic Operators

Since the selection profiles are not stored as bit strings, we cannot perturb them via crossover and bit-flipping. Instead, we use a technique we call *Gaussian Blend*.[4] After selecting two parents, we end up with a pair of values for each of the k elements of the vector. To reduce each pair to a single number that in some way resembles both parents, we consider the values to be samples drawn from a Gaussian distribution; sampling from this distribution

[4]Suggested by Lloyd Allison.

yields a single value suitable for the child. That is, the child's value is drawn from the Gaussian (normal) distribution $N(\mu, \sigma)$, where $\mu = (p_1 + p_2)/2$, $\sigma = ((p_1 - \mu)^2 + (p_2 - \mu)^2)/2$, and p_1 and p_2 are the parents' values. Values less than zero are clipped to zero.

This technique has three advantages: (1) it requires no parameters; (2) crossover and mutation are combined in one operation; and (3) the rate of mutation is proportional to the amount of variation present in the population at the time. It also generalises to any number of parents, although in this work only two parents were ever used.

Other Implementation Details

The meta GA has two levels: an outer GA and an inner GA. The outer GA evolves the selection profiles. The fitness of selection profile P is defined to be -1 times the number of generations needed for the inner GA to find the solution when using a selection scheme derived from P. (-1 so that fewer generations are better.) Sometimes the solution cannot be found within some predefined maximum number of generations, m. (This could happen if the selection profile likes to select individuals of low rank, for example.) In this case, the fitness is $-m$ minus the mean fitness of the inner GA's population at the time the run was aborted. This results in a ranking where "successful" selection profiles come first (in order of speed), followed by "unsuccessful" selection profiles (in order of mean fitness).

Each selection profile was represented by a vector of 9 real numbers, each of which was initialised to an exponential random value with a mean of 10. Each profile was then normalised so that the mean over $[0, 1]$ was 1 (i.e., so that the integral over $[0, 1]$ was 1). We used exponentially-distributed random values rather than, say, normally-distributed values because the exponential distribution has a greater variance, which adds diversity to the initial population.

The outer GA used tournament selection with a tournament size of 5 or 7 to select the selection profiles themselves. SUS sampling was used inside

the inner GA to generate actual offspring counts from the expected counts provided by the selection profiles.

Brief preliminary experiments showed that a meta GA such as this, employing the representation and genetic operators described above, was able to converge on a target function within a small number of generations (around 10) under a wide variety of initial conditions. (This is not surprising: the task is trivial.) We can therefore be confident that the selection profiles that evolve are close to the optimal selection profiles.

We found that a small population of around 30 individuals was ample; for each experiment, we used as small a population as possible since the population size is directly related to the number of fitness evaluations, and fitness evaluation heavily dominates the running time. (In a randomly-generated population of this size, about five selection profiles succeed in evolving a solution to the problem set by the inner GA.) The outer GA was stopped as soon as successive generations yielded no discernable improvement in the mean fitness. This generally required around 15 generations.[5]

4.3.3 Experiment 1: Results and Discussion

In this experiment we evolved selection schemes over two different problems.

Figure 4.4 shows the selection profile evolved over the ONEMAX problem. It is (mostly) monotonically increasing, as expected, and allocates few offspring to individuals ranked in the lower 80% of the population.

The evolved selection profiles required (on average) 19.76 generations to find the optimal solution. This compares with a mean of 19.69 for standard

[5]The exact figure, here and elsewhere, is not important. For the most part we are not interested in the exact shape of the evolved selection profile; we are interested in the general shape (and the fact that it evolved at all). We do not claim that the evolved selection profiles are optimal; we would not be surprised if—as is almost always the case with GAs—different control parameters led to better performance. The purpose of this chapter is to demonstrate the usefulness of the selection profile concept and to use it to draw some general conclusions about the process of selection. See Koza (1992, p. 115–116) for a similar argument.

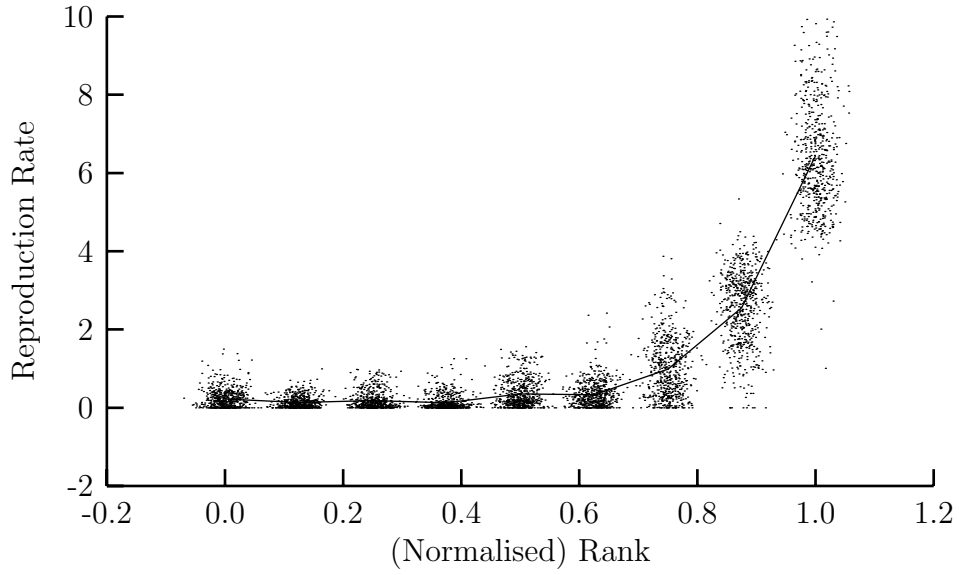

Figure 4.4: Mean selection profile as evolved over the ONEMAX problem. This graph depicts the selection profile vectors present in a population of size 30 after 15 generations. 20 runs are represented. The selection vectors were of length 9; each point represents one of the 9 values. (Noise has been added to x-values of these points.)

tournament selection ($t = 6$). The difference is well within one standard error.

The selection intensity of the mean selection profile is 1.156, the selection variance is 0.743, and the loss of diversity is 0.534. (Recall that the pre-selection mean is 0, and the variance 1.) These were determined experimentally, by performing selection on a population of normally-distributed fitnesses.

Figure 4.5 shows the selection profile evolved on the Lawnmower Problem (Koza, 1994, Chapter 8), a problem where "the goal is to find a program for controlling the movement of a lawnmower so that the lawnmower cuts all the grass in the yard" (Koza, 1994, p. 226). This problem involves evolving programs, not bit strings, and is significantly more complex than ONEMAX. The ADF-enabled 8-by-8 version of the problem was used, as implemented by `lilgp`.[6]

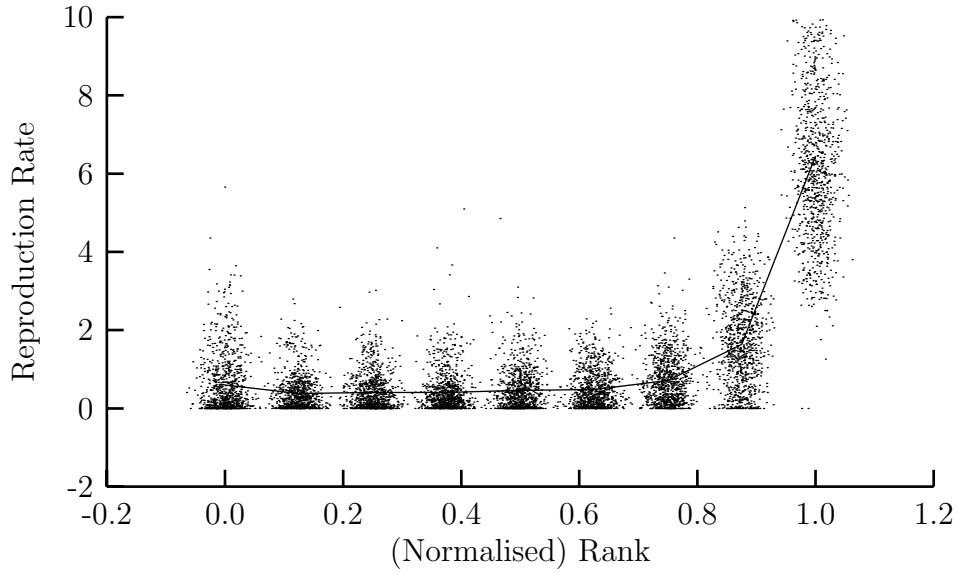

Figure 4.5: Selection profile evolved over the Lawnmower problem. 32 runs are represented. Refer to Figure 4.4 for further details.

[6] `http://garage.cps.msu.edu/software/lil-gp/`

The evolved selection profiles required (on average) 13.56 generations to find the optimal solution. This compares with a mean of 13.25 for standard tournament selection ($t = 6$). The difference is well within one standard error.

The selection intensity of the mean selection profile is 0.893, the selection variance is 1.243, and the loss of diversity is 0.408.

Figure 4.6 shows the (mean) selection profiles of the ONEMAX and Lawnmower problems, together with the selection profile of tournament selection ($t = 6$). The three selection profiles are remarkably similar. They have the same general shape, and they peak at almost exactly the same value.

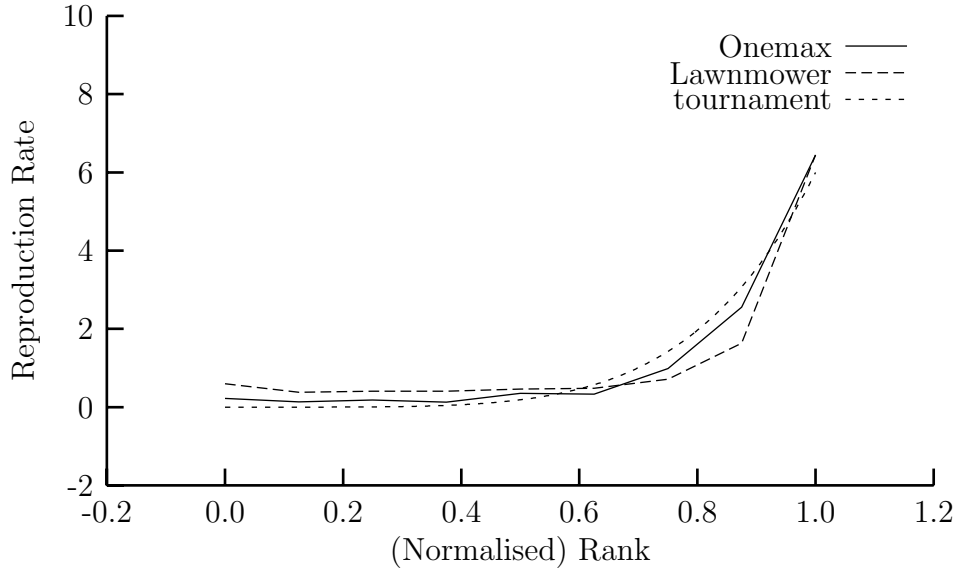

Figure 4.6: Selection profiles evolved over the ONEMAX (see Figure 4.4) and Lawnmower (see Figure 4.5) problems, together with the profile of tournament selection ($t = 6$).

Though the two evolved selection profiles perform about as well as tournament selection, and though the three appear (visually) to be roughly equivalent, their selection intensity, variance, and loss of diversity vary considerably:

	ONEMAX	Lawnmower	Tournament
Selection Intensity	1.156	0.893	1.257
Selection Variance	0.743	1.243	0.415
Loss of Diversity	0.534	0.408	0.582

This suggests that the measures are too sensitive: they give the impression that large differences exist when in fact they are either small or non-existent. (Note also that the Lawnmower-evolved selection profile actually *increases* variance, unlike every one of the selection schemes described in Chapter 3.)

4.3.4 Experiment 2: Results and Discussion

The first experiment demonstrated the feasibility of evolving selection profiles. In this experiment we explore the relationship between selection profiles and problem difficulty. For the inner GA, we chose the $R1$ problem from the Royal Road test suite (Forrest and Mitchell, 1993). In Forrest and Mitchell's canonical version, individuals are 64 bits long, and their fitness is determined by the number of specially-chosen schemata they match. For the standard 64-bit problem, there are 8 fitness-defining schemata, each defined over a different set of 8 consecutive bits. The first consists of 8 `1`s followed by 56 `*`s, the second consists of 8 `*`s followed by 8 `1`s followed by 48 `*`s, and so on. If n is the number of schemata a bit string matches, then its fitness is $(n \times 8)/64 = 0.125n$.

Figure 4.7 shows the selection profiles that evolved under the $R1$ problem, where the length of the bit string varied from 16 to 48 bits. As the bit string became longer, and the problem became harder (the 16-bit problem required about 40 generations to solve, the 48-bit problem 120) the selection profile grew less aggressive, selecting less-fit individuals more often. A limit seems to have been reached around 40 bits—the selection profiles for the 40 and 48 bit problems are practically identical.

This result is to be expected. As the bit strings grow in length, it becomes more and more important to preserve diversity. Suppose that, on the 64-bit

problem, a single bit string matches two fitness-defining schemata (the first and the third, say), but that several match one of the other six schemata. If the selection profile is very steep, then descendents of the most fit bit string will come to dominate the population within a few generations. The less fit bit strings will be wiped out, and with them the strings that match fitness-defining schemata other than the first and third. These strings would have been useful crossover partners to the most-fit bit string; once they go, the schema-matching bits they held can only be restored through the slow process of mutation.

A less aggressive selection profile becomes increasingly beneficial as the bit string grows in length. It preserves more of the less-fit individuals, which can then be crossed with the more-fit individuals to produce even fitter offspring.

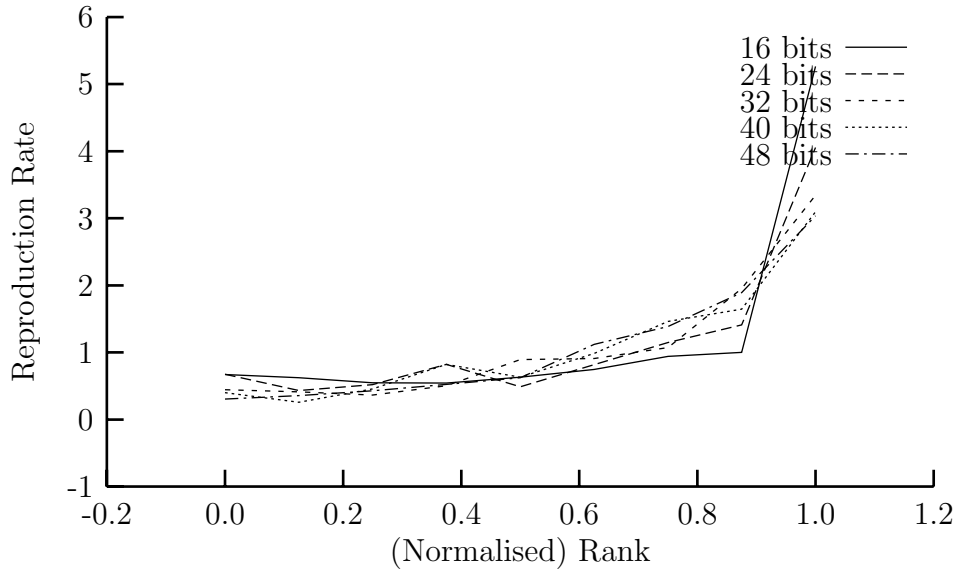

Figure 4.7: Selection profiles evolved over problem $R1$ (see Forrest and Mitchell, 1993). Refer to Figure 4.4 for further details.

4.3.5 Experiment 3: Results and Discussion

Evolutionary Algorithms typically use the same selection scheme throughout their run. This practice seems a odd when one considers that the characteristics of the population change markedly over the course of a run. Initially, the population is very diverse; as time goes on it becomes more and more homogenous as it converges on a single point. It seems likely that a different style of selection is needed at different times in order to maintain the right balance between exploration and exploitation. Whilst rank-based selection schemes might help with this, it seems likely that EAs could benefit from a different selection scheme being used at different stages of the run.

Previous research has demonstrated that adaptive genetic operators—operators whose behaviour changes over the course of the run—can improve the performance of an EA (Angeline, 1996; Teller, 1996; Spears, 1995). In this experiment we investigate whether a similar result holds for selection schemes.

The basic experimental procedure is the same as that of the previous experiment except for these two differences: (1) the outer GA evolves individuals made up of two selection profiles; and (2) the inner GA uses the first selection profile whilst the fitness of the most fit individual is less than or equal to some specified "switch point," and the second otherwise. (Switching is not "sticky"; if the fitness of the most fit individual drops below the switch point, the first selection profile is re-used.)

In the first experiment, the inner GA ran the ONEMAX problem, and the "switch point" was set at 0.75. This resulted in a switch at around the third generation.[7] Figure 4.8 shows the selection profile that evolved during

[7]Note that though the first selection profile directly affects the time at which the switch point is reached, its influence extends beyond this. Though selection profile P_1 may guide the population to the switch point by the third generation, and P_2 by the fourth generation, this does not mean that the *total* number of generations required for P_1 to find an admissible solution is one less than the total required for P_2 (even if the same second phase selection profile is employed). The total depends a great deal on the characteristics of the population at the time the switch takes place.

phase 1; Figure 4.9 shows the selection profile that evolved during phase 2; Figure 4.10 shows both.

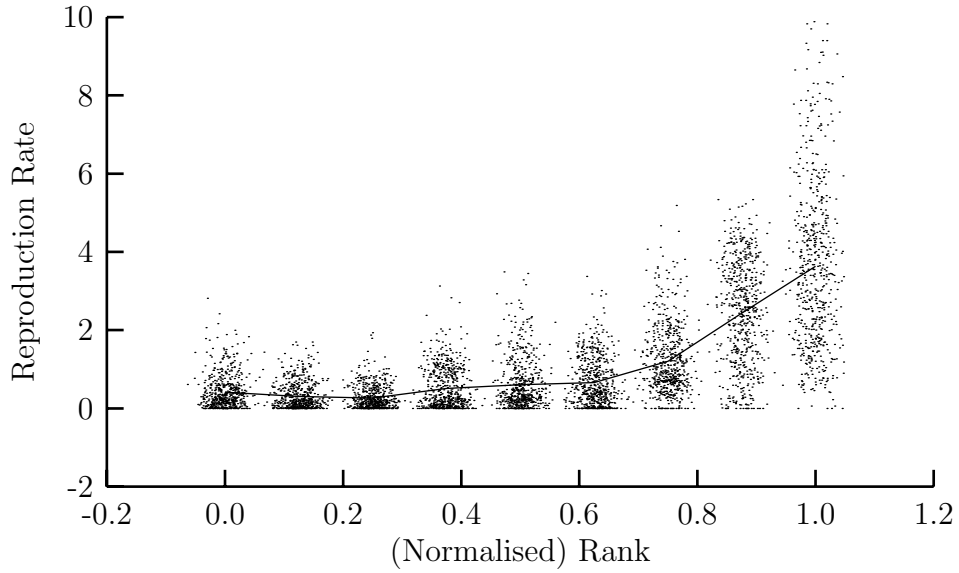

Figure 4.8: Phase 1 selection profile evolved over ONEMAX. The graph depicts the phase 1 selection profile vectors present in a population of size 30 after 15 generations; 20 runs are represented. The selection vectors are of length 9; each point represents one of the 9 values. The phase 1 selection profile was used when the fitness of the most fit individual was less than or equal to 0.75, and the second used otherwise. (Noise has been added to x-values of these points.)

There is a small difference between the two selection schemes. The optimal phase 2 selection scheme is more aggressive than the first, giving a greater preference to high-ranking individuals. This is reflected in the statistics:

	Phase 1	Phase 2
Selection Intensity	0.775	1.128
Selection Variance	0.930	0.794
Loss of Diversity	0.361	0.515

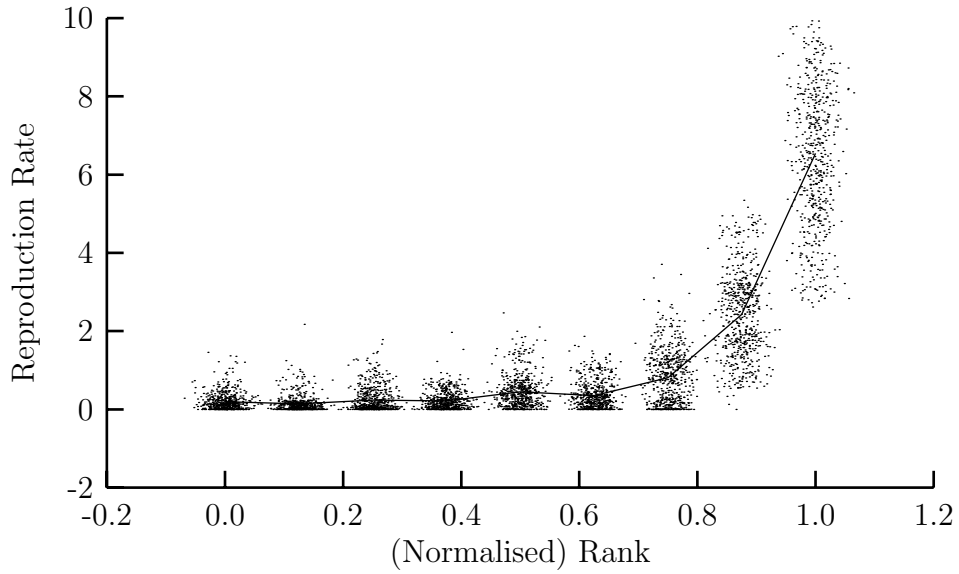

Figure 4.9: Phase 2 selection profile evolved over ONEMAX. Refer to Figure 4.8 for further details.

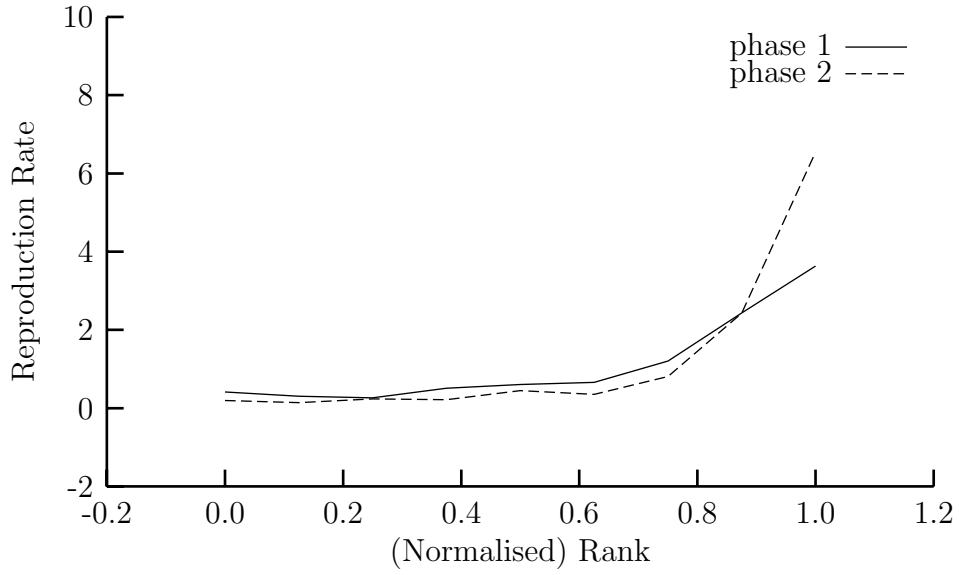

Figure 4.10: Phase 1 and phase 2 selection profiles, evolved over ONEMAX. Refer to Figure 4.8 for futher details.

(Again, the figures seem to suggest a much greater difference between profiles than is apparent from the graph.)

The fact that two distinct selection profiles evolve would appear to demonstrate that as hypothesised, different stages of the run benefit from different selection schemes. However, the mean run time (21.21 generations) for the two-phase selection profile was not any shorter than that of the single-phase profile. This presents us with some interpretational difficulties. We are faced with a situation where different profiles have evolved, despite failing to improve performance.

One possible answer is that the fitness function is in some way misleading—even though it is brutally simple. The mean fitness of the population moves from almost 100 generations to 20 generations (these being the amount of time necessary for the inner GA to find the optimal solution) within about 15 generations. This rapid rate of increase could bias the outer GA's search through the space of selection profiles. Further research is required to answer this question. (On this particular problem, other switch points led to a phase 2 selection profile that was still more aggressive than stage one, although the difference between the two was not as great.)

Chapter 5

Summary of Results and Conclusions

- This thesis began with a critical examination of the Schema Theorem. The Schema Theorem describes, via a lower bound, how the frequencies of schemata change over time. This result is valid (though not particularly useful). However, the Schema Theorem has also been used, by way of analogy to the Two-Armed Bandit problem, to show that GAs go about solving problems in an optimal way. We argue that the analogy is weak, and the claim dubious as a result.

- The Building Block Hypothesis purports to explain how a GA puts together admissible solutions. According to the hypothesis, short, low-order schemata of high fitness are assembled into long, high-order schemata of even greater fitness. This process continues until the solution is found. We present some arguments to show that the Building Block Hypothesis, when cast into a testable form, is of questionable utility.

- Whilst acknowledging that EAs have grown ever more capable over the last ten years (to the point where they can now solve non-trivial problems), we argue that there is a limit to what they can achieve. We

do not believe, as some do, that one day EAs will be able to perform the highest of human tasks. This limitation is mostly due to the role played by the fitness function. Nature does not have an explicit fitness function; in EAs the fitness function (1) measures the wrong quantity and (2) leads the search away from potentially crucial preadaptations. (Systems lacking an explicit fitness function (such as A-life systems) may not be subject to these limitations.)

- We described several different selection schemes, and presented three measures that quantify their behaviour: selection intensity, selection variance, and loss of diversity.

- Finally, we argued that selection schemes can be compared via their selection profiles. As part of this work, we showed that the amount of spread generated by a selection scheme has little or no impact on its effectiveness. We evolved selection profiles with a meta GA and found that the most effective selection schemes had a monotonically increasing selection profile that rises sharply over the last 20% of ranks. This profile closely matched that of tournament selection. We confirmed that more difficult problems warrant less aggressive selection schemes. We also found that it is useful to employ different selection schemes at different stages of the run. These results are preliminary, and invite further work.

Appendix A

How Many Schemata are Processed?

GAs *implicitly* process schemata; *explicitly*, they process bit strings. A single bit string of length l contains 2^l schemata and so a population of n bit strings contains somewhere between 2^l and $n \times 2^l$ distinct schemata. The actual value is not likely to be very close to either the lower or the upper bound. Goldberg (1989), following Holland (1975), estimates that the number of schemata processed is "proportional to the cube of the population size and is thus of order n^3, $O(n^3)$"; he remarks that this estimate is "widely quoted," but "poorly understood" (Goldberg, 1989, pp. 40 and 41). Below, we describe Holland and Goldberg's estimates of the number of schemata we can expect to find in a population of n randomly-chosen bit strings. We further provide an estimate that is more accurate than either.

A.1 Holland's Estimate

Holland estimates the number of schemata represented in a given population in two steps. First, considering a single schema partition P (a schema partition is the set of all schemata that are instantiated over a given set of positions; the set $\{\texttt{*0*0}, \texttt{*0*1}, \texttt{*1*0}, \texttt{*1*1}\}$ is a schema partition), he finds

λ, the expected number of instances of schema $H \in P$. Second, he uses the Poisson distribution to produce an estimate of $r(n, N)$, the chance that H matches at least n individuals in a population of N random bit strings. Holland says:

> On the assumption that elements of α [the search space] are tried at random, uniformly (elements equally likely) and independently, we can use the Poisson distribution to determine the number of schemata receiving at least $n < N$ trials [N is the population size]. The basic parameter required is the average number of trials per schema for any set of schemata defined on h positions [h is the order]. The value of this parameter is just N/k^h [k is the size of the alphabet; N/k^h because each bit string matches exactly one of the k^h schemata] since there are i^h schemata defined on a fixed set of h positions. The Poisson distribution then gives
>
> $$r(n, N) = \sum_{n'=n}^{\infty} ((N/k^h)^{n'}/n'!) \exp(-N/k^h)$$
>
> as the proportion of schemata defined on the positions $i_1, \ldots, k_h$ and receiving at least n out of N trials. (Holland, 1975, p. 72)

Holland uses the Poisson distribution as an approximation to the true distribution. As it turns out, for "reasonable" values of n, N, and h, the estimate is very close to correct. Below, we argue that the true distribution is binomial; we also provide some feeling for the magnitude of the error.

A.2 Goldberg's Estimate

What follows is Goldberg's estimate[1] as described in his book (Goldberg, 1989).

[1] Goldberg claims that his estimate is inspired by Holland's but I am unable to see the resemblance. The argument is, however, similar to that of Holland (1988, pp. 120–1) and Booker et al. (1989, p. 263).

Goldberg's estimate is produced by first counting the number of schemata with a defining length of l_s or less represented by an individual bit string of length l. There are $l - l_s + 1$ unique sliding "windows" of length l_s, and each window contains 2^{l_s-1} schemata (to ensure uniqueness we must, at the same point within each window, fix one bit). Thus, the number of schemata of length l_s or less is not less than $2^{l_s-1} \times (l - l_s + 1)$.[2]

We now know the number of schemata that inhabit a single bit string. Unfortunately, because the schemata are not necessarily unique, we can't simply multiply this figure by the population size to find the total number of schemata. (We expect fully half the population to match `1*···*`, for example.) We must take steps to ensure that our count contains no duplicates.

Goldberg realised that every schema within an order-n schema partition can expect to be represented once in a population of size 2^n since a schema partition of order n contains 2^n distinct schemata. (Schemata of lesser order can expect to be represented more than once.) Thus, Goldberg reasons, we can expect that a population of size $2^{l_s/2}$ will contain all the schemata of order $l_s/2$, as well as all the $(l_s/2) - 1$ schemata (some multiple times), all the $(l_s/2) - 2$ schemata, and so on. The schema of order $l_s/2$ and *more*, on the other hand, can expected to be represented only once; they exist once by virtue of being present in the population, but the population is not big enough for them to be represented any more than that. Since the number of schemata is binomially distributed (with respect to order), the schemata of order $l_s/2$ and more number exactly half the total number of schemata. Consequently, n_s, the number of schemata represented in a population of size

[2]Goldberg presents this figure as the true number of schemata, but in actual fact the number is slightly greater: some schemata are not counted by this sliding window system. The counting system uses a window,

`%%%%1*****`

that slides to the right. (The `%` characters indicate the positions that take on either the value that appears at that point in the bit string or the `%` character.) Clearly this counting method fails to enumerate some schemata, such as `1*********`.

$n = 2^{l_s/2}$ can be estimated as follows:

$$\begin{aligned} n_s &\geq \frac{2^{l_s/2} \times (l - l_s + 1)2^{l_s - 1}}{2} \\ &\geq \frac{(l - l_s + 1)n^3}{4} \end{aligned} \tag{A.1}$$

which leads directly to the $O(n^3)$ lower bound.

Goldberg's estimate has several flaws:

1. It clearly does not hold for all n. If n is large enough, we would expect all the schemata, of every defining length and order to be represented. Therefore, even if the rate of growth is $O(n^3)$ over part of the range (to interpret $O(\cdot)$ charitably), there is a point at which the rate of growth will fall below $O(n^3)$, but Goldberg's estimate does not indicate when this occurs.

   One suspects that n does not have to be very large for the $O(n^3)$ growth rate to not apply. The expected number of copies of schemata of order 10 and less (which implies a defining length of 10 or less) is 1 or greater in a population of $1024 = 2^{10}$ bit strings, for example. In a population of $2048 = 2 \times 2^{10}$ bit strings, the expected number is at least 2. We would expect most of the possible schemata to be represented at this stage.

2. The $O(n^3)$ estimate only holds when $l_s = 2 \log_2 n$, not for any combination of n and l_s. (In other words, the $O(n^3)$ bound only holds if we allow the length of the schemata included in the count to increase as n does.) This constraint comes about because the population size was specifically chosen to ensure that the expected number of schemata of a given order was no more than one: "By choosing $[n]$ in this manner, we expect to have one or fewer of all schemata of order $l_s/2$ or more." (Goldberg, 1989, p. 41) Thus, in equation (A.1), l_s and n are not independent, and the only claim that can be made is that if n happens to be equal to be $2^{l_s/2}$, we can expect $c \times n^3$ schemata of length l_s or less to be represented.

It is strange that Goldberg (1985)—a technical report upon which the schema-counting section of Goldberg (1989) is clearly based—recognises exactly this problem. In a section inexplicably missing from his book, Goldberg acknowledges that (for the above reaons) "by selecting a particular size population we limit the applicability of the equation" (Goldberg, 1989, p. 4). He goes on to say that arguments such as the following are "faulty": "'Since the useful schema processing is of order n^3, to obtain increased schema leverage, simply increase your population size'" (p. 4)—even though this is precisely the conclusion one would draw from the schema-counting discussion in his book. And finally, Goldberg says that because "the estimate for n_s depends upon a *particular* choice of population size and any deviation in population size invalidates the derivation ... [then] if we are to shed any light on the population size questions, we need to (1) count the number of unique schemata expected in a given population and (2) derive a rational figure of merit for calculating an optimal population size." (p. 4) If no light has been shed on the problem thus far, what good is the $O(n^3)$ estimate?

We note in passing that the selection of $n = 2^{l_s/2}$ in fact maximises the per-bit-string number of instances of schemata of defining length l_s or less. That is, the ratio n_s/n is maximised when $n = 2^{l_s/2}$. If n is any smaller or larger, we get less schemata on a per-bit-string basis.

3. The argument involves sizing our population so that all the schemata of a order d can expect to be represented at least once. (That is, their expected number of instances is 1 or more.) Goldberg basically assumes that in this event, every schema of order d or less can expect to be represented. But this is different to saying that we can expect every schema (i.e., all of them) to be represented!

   Suppose we place 100 balls into a box, from which we sample 100 times, with replacement. In this situation, each ball can expect to be selected

once. However, it is likely that some balls will be selected twice, and hence there is a chance that a particular ball will not be selected at all. This chance is $(1 - (1/100))^{100} \approx 0.37$. That is, we can expect that of the 100 balls, only 63 different balls will be actually be selected.

We cannot expect all the schemata of order $(l_s/2) - 1$ to be present in a population of size $2^{l_s/2}$; such a population will contain duplicates.

A.3 An Alternative Estimate

There are $\binom{l}{d}2^d$ distinct schemata of order d in a bit string of length l, since there are d "fixed" (or instantiated) positions, each of which can take one of two values (0 or 1), and there are $\binom{l}{d}$ distinct ways to choose which d of l positions are fixed. A randomly chosen string matches a schema of order d with probability 0.5^d, since there are d positions at which the value of the string and the schema must match, and at each position, there is a 0.5 chance that the two will match.

The probability that *at least one* randomly-chosen bit string in a population of size n matches a schema of order d is then

$$1 - \left(1 - 0.5^d\right)^n$$

and thus the expected number of schemata of order d that match at least one bit string is

$$M'(l, n, d) = \binom{l}{d} 2^d \left(1 - \left(1 - 0.5^d\right)^n\right) \tag{A.2}$$

(The prime indicates that this is a restricted version of the formula; a more general version is developed later.)

To get this result we multiplied the number of schemata in a given order by the chance that a single schema of that order matches at least one member of the population. But it would seem that the latter value is not constant, and that it depends on the schemata that are already known to exist. Suppose

that we are trying to determine which schema of order two are represented in a population. We might find that the following situation holds:

Schema	Exists?
$H_1 = \texttt{00***}$	Yes
$H_2 = \texttt{01***}$	No
$H_3 = \texttt{10***}$	Yes
$\vdots$	$\vdots$
$H_k = \texttt{*1*1*}$	?
$\vdots$	$\vdots$

The chance that H_k exists is not independent, but is affected by the knowledge that H_1 and H_3 do exist, and that H_2 does not. (The problem is particularly acute if the population size is small. If there are only two individuals, for example, then we know immediately that H_k does not exist.)

Though I have not yet been able to prove that equation (A.2) holds, empirical evidence suggests that it does, or at least nearly so. Knowing that a particular schema exists lowers the chance that some others do, but it also raises the chance of an equal number.

The total number of distinct schemata matched in a population of n bit strings of length l is then

$$\begin{aligned} T'(l,n) &= \sum_{i=0}^{l} M'(l,n,i) \\ &= \binom{l}{d} \sum_{i=0}^{l} \left(1 - \left(1 - 0.5^i\right)^n\right) \\ &= \binom{l}{d} \left(l - \sum_{i=0}^{l} \left(1 - 0.5^i\right)^n\right) \end{aligned}$$

Figure A.1 shows the proportion of schemata that match at least one individual in a population of n bit strings of length 10. Just over 90% are represented in a random population of 500 bit strings. (Amongst those missing are approximately 500 schemata of order 10. There are $2^{10} = 1024$

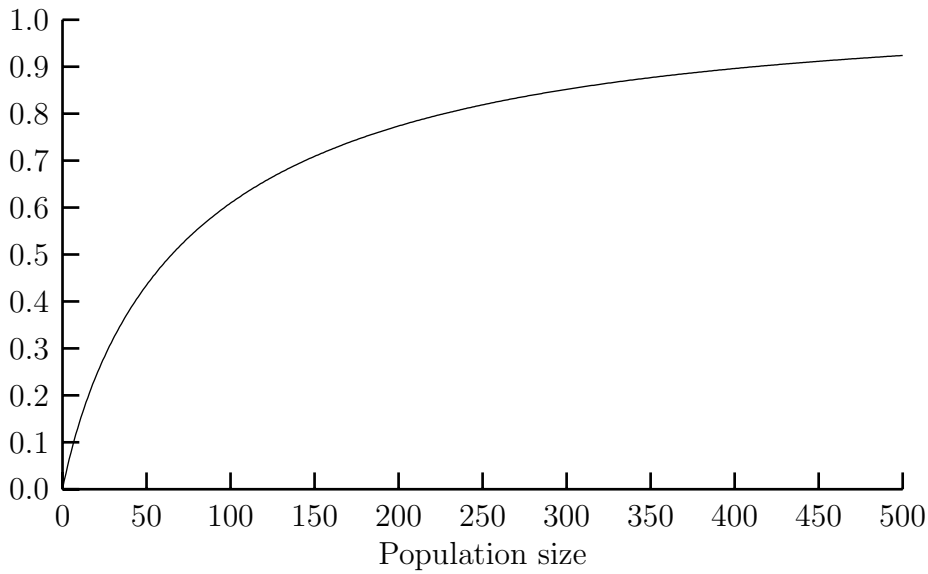

Figure A.1: The proportion of schemata that match at least one individual in a population of n bit strings of length 10.

distinct members of this set, which is about 500 more than there are strings in the population. On the other hand, we can expect that every one of the $\binom{10}{1}2^1 = 20$ order-1 schemata will be represented.) The total number of schemata is $\sum_{i=0}^{10}\binom{10}{i}2^i = 59049$. (Which can be seen to be equal to 3^{10}, which is the answer we get if we consider each schema to be a bit string composed of letters from an alphabet of size 3.)

We can also determine the expected number of schemata of order d that matched at least a, and at most b, individuals in a population of size n. (Typically, $b = n$; we want to know the number of schemata that match *at least* a times.) This quantity is

$$M_a^b(l,n,d) = \binom{l}{d}2^d\left(\sum_{i=a}^{b}\beta(0.5^d,n,i)\right) \tag{A.3}$$

where $\beta(p,n,r)$ is the binomial distribution, $\beta(p,n,r) = \binom{n}{r}p^r(1-p)^{n-r}$ and l is the length of the bit strings.

The total number of distinct schemata matched by between a and b in-

dividuals in a population of n bit strings is thus

$$\begin{aligned} T_a^b(l,n) &= \sum_{i=0}^{l} M_a^b(l,n,i) \\ &= \sum_{i=0}^{l}\sum_{j=a}^{b} \binom{l}{i} 2^i \beta(0.5^i, n, j) \end{aligned} \tag{A.4}$$

Sometimes we want to find the number of schemata with a defining length of f or less that are represented by at least r individuals. (Such a figure is useful, for example, if we wish to determine how many schemata are likely to survive crossover.) We can calculate this figure by employing a variation of Goldberg's "sliding windows" technique.

Suppose our window is of width f. Then $T_r^\infty(f,u)$ is the number of distinct schemata we can expect to find within this particular window. Unfortunately, multiplying this figure by however many unique positions the window can take will count some schemata more than once. So instead, we divide the window into two sections. The first consists of one fixed bit that can be either `0` or `1`, but not `*`. The second section is of length $f-1$, and is unrestricted in content. The number of schemata contained by this two-part window is then $T_r^\infty(f-1,n)$. (If there are $T_r^\infty(f-1,n)$ schemata of length $f-1$ represented in a population of size n, then if we add one bit to the start of the schema, and require that it be fixed, exactly the same number of schemata as before must be represented, since the first bit of any randomly chosen bit string must be either 0 or 1—nothing previously included is now excluded, and vice versa.)

Since the first bit is defined (fixed) in every one of these windows, no two window positions along the bit string will yield any duplicate schemata. Hence, an upper bound for the expected number of schemata of defining length f or less that are represented at least r times in a population of size n is

$$S_r(l,n,f) \leq 1 + (l \times T_r^\infty(f-1,n)) \tag{A.5}$$

where l is the length of the bit string. (This is an upper bound because we advance the window l times; after position $l-f$, part of the window extends

past the "end" of the bit string, which means that we count schemata that can't possibly exist. (The overestimate grows smaller as f/l does.) The "1+" is for the schema $*\cdots*$ which is otherwise not counted.)

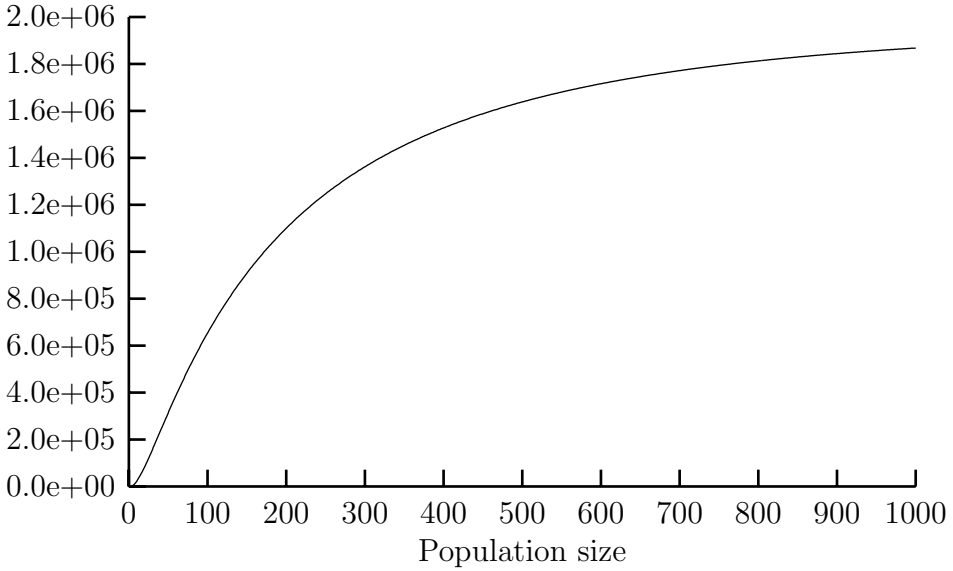

Figure A.2: The number of schemata of defining length 10 or less that match at least three individuals in a population of n bit strings of length 100.

Figure A.2 shows the number of schemata of defining length 10 or less and length 30 or less that match at least three individuals in a population of n bit strings of length 30. The number of schemata is vastly greater than the number of bit strings, but the rate of growth is not $O(n^3)$. Figure A.3 is the same, except that it shows the number of schemata of defining length 30 or less. Over this range of population sizes, the rate of growth is somewhat closer to $O(n^3)$.

A.4 Comparison of Measures

Figure A.4 shows the ratio between Holland's Poisson-based estimate of the number of schemata of order 5 and order 10 that are represented at least 3

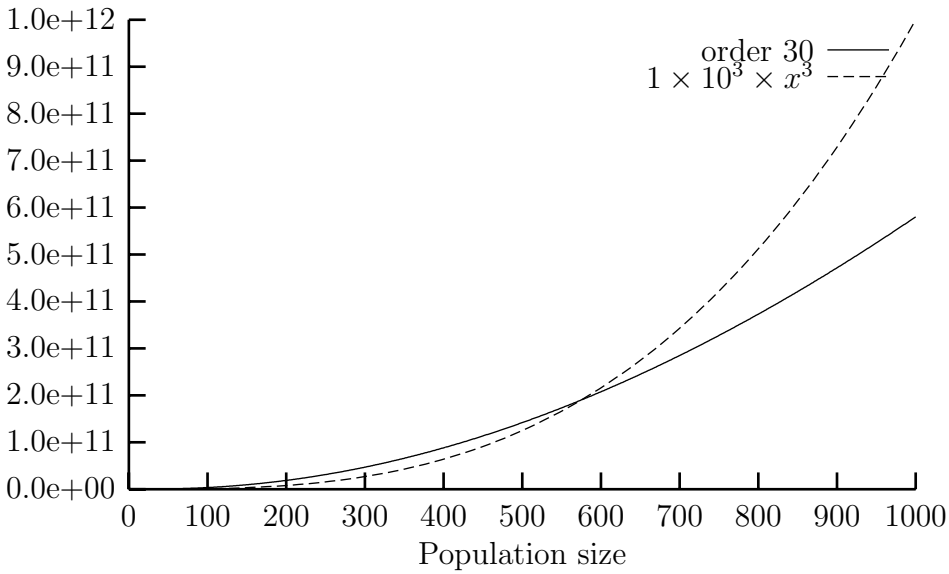

Figure A.3: The number of schemata of defining length 30 or less that match at least three individuals in a population of n bit strings of length 100.

times in a population of size n and the estimate developed on the previous section. The difference decreases as the population gets larger; it is small for typical population sizes, and the measures are clearly of the same order. Note that length of the bit string itself does not enter into the calculation.

Goldberg's $O(n^3)$ estimate cannot be meaningfully compared with either Holland's estimate or the binomial-based estimate presented above.

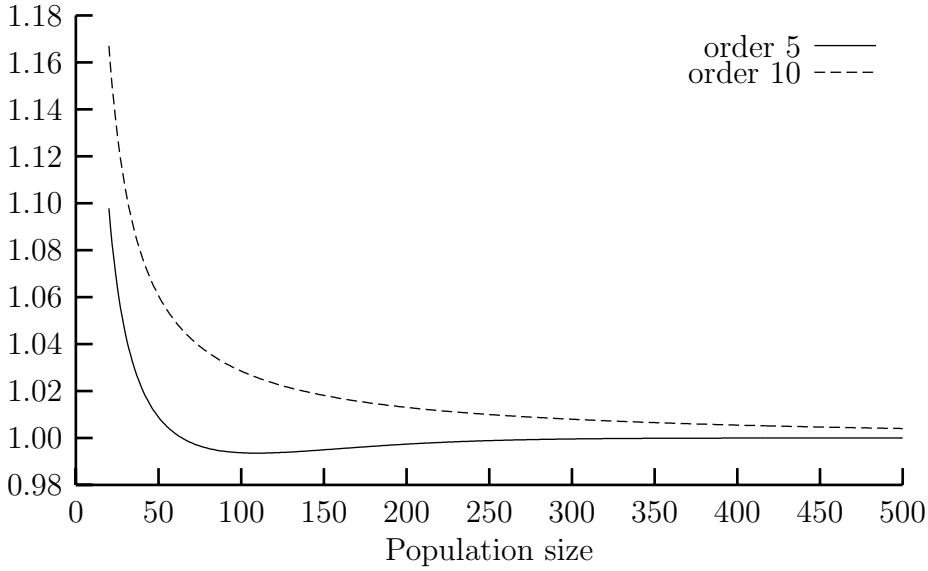

Figure A.4: The graph of h/t, where h is Holland's estimate and t is the estimate developed in Section A.3. The number to estimate is the number of schemata of order 5 and order 10 that are represented at least 3 times in a population of size n.

Bibliography

Peter J. Angeline. Two self-adaptive crossover operators for genetic programming. In Peter J. Angeline and Jr. K. E. Kinnear, editors, *Advances in Genetic Programming 2*, chapter 5, pages 89–110. MIT Press, Cambridge, Massachusetts, 1996.

Thomas Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 57–62, 1994.

James Edward Baker. Reducing bias and inefficiency in the selection algorithm. In John J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*. Erlbaum, 1987.

Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic programming—an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann, San Mateo, California, 1998.

Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, May 1997. URL `http://www.handshake.de/user/blickle/publications/ECfinal.ps`. Quotations are taken from the on-line version.

L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.

William H. Calvin. The emergence of intelligence. *Scientific American*, pages 79–85, October 1994.

James F. Crow and Motoo Kimura. *An Introduction to Population Genetics Theory*. Harper & Row, New York, 1970.

Robert Dorit. Book review: *Darwin's Black Box: The Biochemical Challenge to Evolution*. *American Scientist*, September/October 1997. URL `http://www.amsci.org/amsci/bookshelf/Leads97/Darwin97-09.html`.

William Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, 3rd edition, 1968.

Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126. Morgan Kaufmann, San Mateo, California, 1993.

Douglas J. Futuyma. *Evolutionary Biology*. Sinnauer Associates, Inc., Sunderland, Massachusetts, third edition, 1998.

David E. Goldberg. Optimal initial population size for binary-coded genetic algorithms. Technical report, Tuscaloosa: University of Alabama, 1985. TCGA Report No. 85001.

David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, Massachusetts, 1989.

David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Mateo, California, 1991.

John J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1): 122–128, 1986.

John J. Grefenstette. Deception considered harmful. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 75–91. Morgan Kaufmann, San Mateo, California, 1993.

John J. Grefenstette and James E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 20–27, San Mateo, California, 1989. Morgan Kaufmann.

John Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975. Quotations and page numbers taken from the 2nd edition, MIT Press, 1992.

John H. Holland. The dynamics of searches directed by genetic algorithms. In Y. C. Lee, editor, *Evolution, Learning and Cognition*, pages 111–127. World Scientific, Singapore, 1988.

David Hume. *A Treatise of Human Nature.* 1739.

Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem difficult for genetic algorithms. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192, San Francisco, 1995. Morgan Kaufmann.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Massachusetts, 1992.

John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, Massachusetts, 1994.

William B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming.* Kluwer Academic Publishers, Boston, 1998.

Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, third edition, 1996.

Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts, 1996.

Una-May O'Reilly. *An Analysis of Genetic Programming*. PhD thesis, Carleton University, 1995. URL `http://www.ai.mit.edu/people/unamay/thesis.html`.

Ralf Salomon. The influence of different coding schemes on the computational complexity of genetic algorithms in function optimization. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature*, pages 227–235, New York, 1996. Springer-Verlag.

William M. Spears. Adapting crossover in evolutionary algorithms. In John R. McDonnell, Robert G. Reynolds, and David B. Fogel, editors, *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 367–384, Cambridge, Massachusetts, 1995. MIT Press.

Astro Teller. Evolving programmers: The co-evolution of intelligent recombination operators. In Peter J. Angeline and Jr. K. E. Kinnear, editors, *Advances in Genetic Programming 2*, chapter 3, pages 45–68. MIT Press, Cambridge, Massachusetts, 1996.

Michael D. Vose. Generalizing the notion of schema in genetic algorithms. *Artificial Intelligence*, 50:385–396, 1991.

David H. Wolpert and William G. Macready. No Free Lunch theorems for search. Technical Report 95-02-010, The Santa Fe Institute, 1995. URL `http://www.santafe.edu/sfi/publications/Abstracts/95-02-010abs.html`.

David H. Wolpert and William G. Macready. On 2-armed Gaussian Bandits and optimization. Technical Report 96-03-009, The Santa Fe Institute,

1996. URL `http://www.santafe.edu/sfi/publications/Abstracts/96-03-009abs.html`.