Chapter 3

Peter J.B. Hancock
Department of Psychology
University of Stirling,
Scotland, FK9 4LA

pjh@compsci.stirling.ac.uk

Selection Methods for Evolutionary Algorithms

Abstract

Selection pressure can have a decisive effect on the outcome of an evolutionary search. Try too hard, and you will end up converging prematurely, perhaps on a local maximum, perhaps not even that. Conversely, too little selection pressure, apart from wasting time, may allow the effects of genetic drift to dominate, again leading to a suboptimal result. In nature, there are two aspects to breeding success: surviving long enough to reach reproductive maturity, and then persuading a mate to be your partner. In simulations, such subtleties are mostly the province of artificial life experiments where, for example, an animal that fails to find enough food may die. In such systems it is possible for the whole population to die out, which may be realistic but does rather terminate the search. In most Evolutionary Algorithms (EA), therefore, a more interventionist approach is taken, with reproductive opportunities being allocated on the basis of relative fitness. There are a variety of selection strategies in common use, not all of which use the fitness values directly. Some order the population, and allocate trials by rank, others conduct tournaments, giving something of the flavour of the natural competition for mates. Each of the schools of EA has its own methods of selection, though GA practitioners in particular have experimented with several

algorithms. The aim of this chapter is to explain the differences between them, and give some indication of their relative merits.

At its simplest, selection may involve just picking the better of two individuals. Rechenberg's earliest Evolution Strategy proceeded by producing a child by mutation of the current position and keeping whichever was better. Genetic algorithms require a substantial population, typically of the order of a hundred, in order to maintain diversity that will allow crossover to make progress. Holland's original scheme for GAs assigned each individual a number of offspring in proportion to its fitness, relative to the population average. This strategy has been likened to playing a two-armed bandit, with uncertain payoffs. How should one best allocate trials to each arm, given knowledge of the current payoff from each? The best strategy turns out to be to give an exponentially increasing number of trials to the apparently better arm, which is exactly what fitness proportional selection does for a GA. However, the approach suffers from a variety of problems, which will be illustrated, along with possible solutions, below.

A full comparison of selection methods might involve their use on a range of tasks and the presentation of large tables of results. These would probably be ambiguous, since it seems unlikely that there is any one best method for all problems. Instead, this chapter follows the lead of Goldberg and Deb, who compared a number of the common GA selection methods in terms of their theoretical growth rate and time complexity. They considered an extremely simple problem, where there are just two string values, arbitrarily 1 and 1.5. The initial population contains one copy of 1.5. They then looked at how quickly different selection methods would cause this string to take over the population, without any mutation or other genetic operators. Three similar simple problems are used here. In all cases, results reported are the average of 100 runs.

1. Take-over. A population of N=100 individuals are initialised with random values between 0 and 1, except for one, which is set to 1. This population is acted on by selection alone, to give take-over curves analogous to those produced by Goldberg and Deb. However, the range of values in the initial population allows observation of the worst values, as well as the best. If poor individuals are removed too quickly, genetic diversity needed for the final solution may be lost.

2. Growth. Some of the selection schemes considered produce exponential take-over rates. To allow comparisons under slightly more realistic conditions, mutation was added. The whole population is initialised with random values in the range 0-0.1. When an individual is reproduced, the copy has added to it a Gaussian random variable, with standard deviation of 0.02, subject to staying within the range 0-1. The population gradually converges towards 1, at a rate mostly determined by the selection pressure, though clearly limited by the size of the mutation.

3. Noise. Many target objective functions are noisy, and one of the claims made about Genetic Algorithms is that they are relatively immune to its effects. As will be seen, the degree of immunity depends on which selection method is used.

The task is the same as the previous one, except that another Gaussian random variable is added to each individual's value. The noisy score is used to determine the number of offspring allocated, the true value is then passed on to any children, subject to small mutation as before.

The time complexity of the different algorithms is not considered here, because it is rarely an issue in serious applications, where the time taken to do an evaluation usually dominates the rest of the algorithm. If this is not the case, then the whole run will probably only take a few seconds: one or two more shouldn't hurt!

On the other hand, stochastic effects, ignored by Goldberg and Deb, are considered here. A selection algorithm might specify 1.6 offspring for a given individual. In practice, it will have to get a whole number, and there are different ways to do the required sampling. Some methods are prone to errors, such that even the best individual may not get any offspring. Where this happened during the take-over simulations, the best value was replaced, arbitrarily overwriting the first member of the population. If this were not done, the graphs would be more affected by the particular number of runs that lost the best value than by real differences in the take-over rate in the absence of such losses. Suppose two sets of 100 runs of an algorithm are conducted, where the best string is lost with probability 0.5 on any one run. One set of runs might lose the best, say, 48 times, the other 55. The latter will appear to grow more slowly, simply because more zero values are being averaged in. The number of occasions such replacement was needed will be reported.

The results of the simulations require interpretation — it is certainly not simply the case, for example, that faster growth rates are "better". A working assumption behind the interpretation offered below is that, other things being equal, greater diversity in the population is beneficial.

This chapter is only concerned with single "panmitic" population models, where all individuals compete for selection in a single pool. There are a variety of interesting parallel models, including multiple small populations that occasionally exchange individuals, and spatial populations, where each individual sees only its immediate neighbours. Such models would be difficult to compare meaningfully by the simple methods employed here.

Also not considered here are a variety of methods used to influence selection, usually to encourage diversity in the population. This might simply be to improve the search by preventing premature convergence or perhaps to allow multiple solutions to be found. Techniques such as niching (Deb and Goldberg 1989), sharing (Goldberg and Richardson, 1987), crowding (De Jong, 1975), mate-selection (Todd and Miller, 1991) and incest prevention (Eshelman, 1991) all find their place in the literature.

3.1 Fitness proportionate selection (FPS)

The traditional GA model selects strings in proportion to their fitness on the evaluation function, relative to the average of the whole population. Holland's original scheme actually suggested picking only one parent according to fitness. If a second is required for crossover, this is picked at random. This produces rather lower selection pressure, but results that are qualitatively similar to the now more common practice of picking both according to fitness (Schaffer, 1987). FPS unfortunately suffers from well-known problems to do with scaling. Suppose you have two strings, with fitness 1 and 2, respectively. The second string will get twice as many reproductive opportunities as the first. Now suppose that the underlying function is altered simply by adding 10 to all the values. Our two strings will now score 11 and 12, a ratio of only 1.09. It might be hoped that such a simple translation of the target function would have no effect on the optimisation process. In practice the selection pressure would be significantly reduced.

This scaling effect causes another problem. Suppose we are optimising a function with a range of 0-10. Initially, the random population might score mostly in the range 0-1. A lucky individual with a score of 3 will then be given a large selective advantage. It will take over the population, reducing, and eventually removing, the genetic diversity. If this potential hazard is avoided, the fitness of the population might improve, say to the range 9-10. Now, for the reason described in the previous paragraph, there will be very little selection pressure, and the search will stagnate. In summary: if there is little variation in the fitness of the strings, there will be little selective pressure. The problem of stagnation has been addressed by using a moving baseline: windowing and sigma scaling.

3.2 Windowing

One way to ameliorate the problem is to use the worst observed score as a baseline, and subtract that value from all the other fitnesses. This then converts our stagnating population in the range 9-10 back to the range 0-1. However, it will give the worst string a fitness of zero, and, as noted above, it is not generally wise to exclude weaker strings completely. The selection pressure is therefore usually reduced by using the worst value observed in the *w* most recent generations as a baseline, where *w* is known as the window size, and is typically of the order of 2-10. The dramatic effect of this moving baseline is shown in Figure 3.1a, which shows the increase in the number of copies of the optimal value under selection only. FPS initially converges rapidly, but then tails off as all of the population approaches a score of 1. Moving the baseline maintains the selection pressure, more strongly for smaller window size. Subtraction of the worst value also solves the problem of what to do about negative values. A negative number of expected offspring is not meaningful. Simply declaring negative values to be zero is not sufficient, since with some evaluation functions the whole population might then have a score of zero.

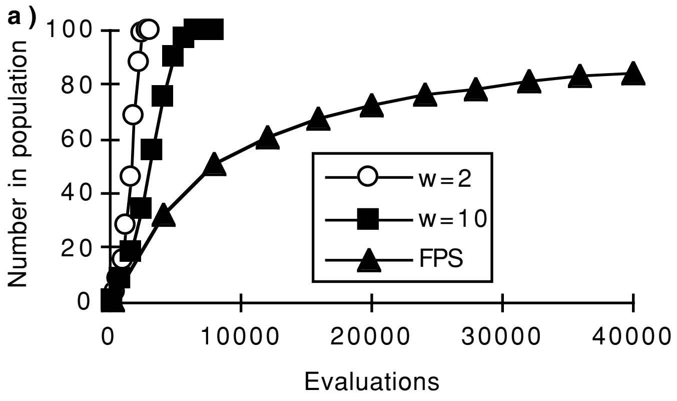

Figure 3.1a) Take-over rates for fitness proportionate selection, with and without baseline windowing.

3.3 Sigma scaling

As noted above, the selection pressure is related to the scatter of the fitness values in the population. Sigma scaling exploits this observation, setting the baseline *s* standard deviations (sd) below the mean, where *s* is the scaling factor. Strings below this score are assigned a fitness of zero, with a consequent potential for the loss of diversity. This method helps to overcome a potential problem with particularly poor individuals ("lethals") which with windowing would put the baseline very low, thus reducing selection pressure. Sigma scaling keeps the baseline near the average. It also allows the user to adjust the selection pressure, which is inversely related to the value of *s*. By definition, the average fitness of the scaled population will be *s* times sd. Thus an individual that has an evaluation one standard deviation above the average will get $(s+1)/s$ expected offspring. Typical values of *s* are in the range 2-5, with stronger selection again given by smaller values. The effect on take-over rate is shown in Figure 3.1b, for *s* values of 2 and 4: selection pressure is rather greater than with a window of size 2.

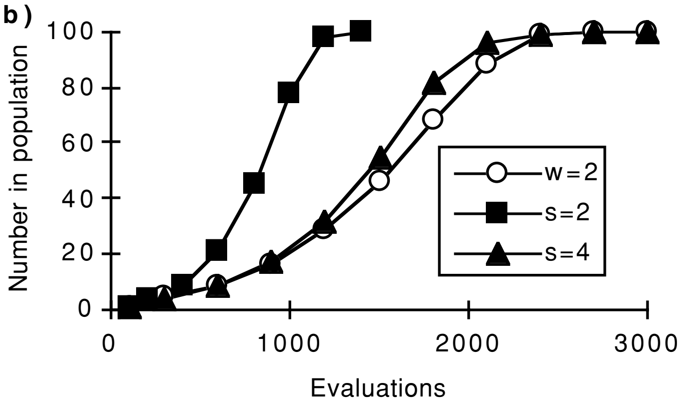

Figure 3.1b) Take-over rates for window, and sigma-scaling baseline methods. Note the x scale change.

These moving baseline techniques help to prevent the search from stagnating, but may exacerbate the problem of premature convergence caused by a particularly fit individual because they increase its advantage relative to the average. The sigma scaling method is slightly better, in that good individuals will increase the standard deviation, thereby reducing their selective advantage somewhat. However, a better method is desirable.

3.4 Linear scaling

Linear scaling adjusts the fitness values of all the strings such that the best individual gets a specified number of expected offspring. The other values are altered so as to ensure that the correct total number of new strings are produced: an average individual still expects one offspring. Exceptionally fit individuals are thus prevented from reproducing too quickly.

The scaling factor *s* specifies the number of offspring expected for the best string and is typically in the range 1.2 to 2, again giving some control on the selection pressure. The expected number of offspring for a given string is given by:

$$1+\frac{(s-1)(fitness-avg)}{(best-avg)}$$

It may be seen that this returns *s* for the best, and 1 for an average string. There is still a problem for low-scoring strings, which may be assigned a negative number of offspring. It can be addressed by assigning them zero, but this would require that all the other fitness values be changed again to maintain the correct average. It also risks loss of diversity. An alternative is to reduce the scaling factor such that just the worst individual gets a score of zero:

$$s=1+\frac{(best-avg)}{(avg-worst)}$$

The algorithm may be summarised in the following C-code, which adds another variable *ms*, set to 1 less than the modified *s* value to save a subtraction in the for loop:

```
if (s > 1 + (best-avg)/(avg-worst) )
        ms = (best-avg)/(avg-worst);
else
          ms = s - 1;
for (i = 0; i< N; i++)
        fitness(i) = 1 + ms * (fitness(i) - avg)/(best - avg);
```

The effects on convergence rate are shown in Figure 3.2a. As expected, increasing the scaling factor increases the convergence rate. With a linear scaling factor of 2, the convergence is between that obtained from a window size of 2, and a sigma scaling factor of 2. At low selection pressures, the convergence rate is proportional to *s*. Thus in this simulation, the best value takes over the population in 4000 evaluations for *s*=1.2. With *s*=1.1, it takes 8000 evaluations.

This would suggest convergence in less than 1000 evaluations when *s*=2, where in fact it takes 2000. The reason is the automatic reduction in selection pressure caused by the need to prevent negative fitness values. In this application the convergence produced with *s*=2 is very similar to that produced with *s*=1.5. The effective selection pressure is therefore still determined to some extent by the spread of fitness values in the population. A very poor individual will effectively terminate the search, so it is worth monitoring the actual value of *s* during the run and if necessary discarding such lethals.

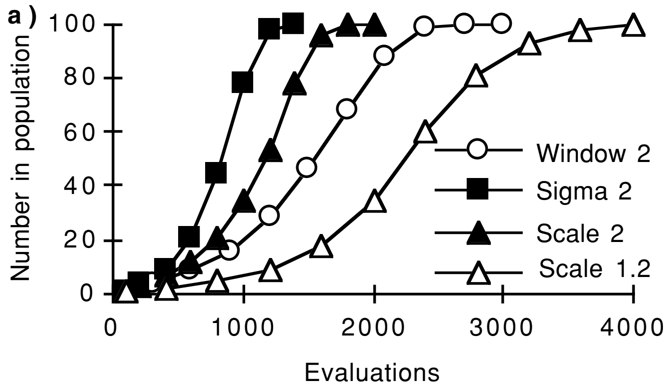

Figure 3.2a) Take-over rates for baseline window, sigma and linear scaling.

The growth rates in the presence of mutation for these scaling methods are shown in Figure 3.2b. All are quite similar, simple FPS being able to maintain selection pressure because of the range of fitness values caused by the mutation. Windowing and sigma scaling come out ahead precisely because they fail to limit particularly fit individuals. Fortuitous mutations are therefore able to reproduce rapidly.

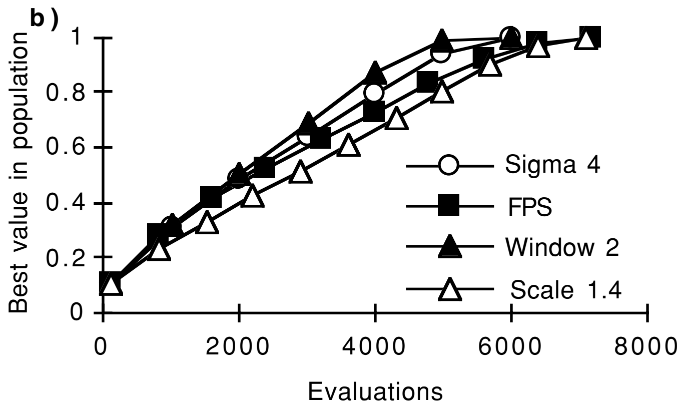

Figure 3.2b) Growth rates for FPS and three scaling methods.

3.5 Sampling algorithms

The various methods just described all deliver a value for the expected number of offspring for each string. Thus with direct fitness measurements, a string with twice the average score should be chosen twice. That is straightforward to implement, but there are obvious problems with non-integer expected values. The best that can be done for an individual with half the average fitness score, that expects 0.5 offspring, is to give it a 50% probability of being chosen in any one generation. Baker, who considered these problems in some detail, refers to the first process as selection, and the second as sampling (Baker, 1987).

A simple, and lamentably still common way to perform sampling may be visualised as spinning a roulette wheel, the sectors of which are set equal to the fitness values of each string. The wheel is spun once for each string selected. The wheel is more likely to stop on bigger sectors, so fitter strings are more likely to be chosen on each occasion. Unfortunately this simple method is unsatisfactory. Because each parent is chosen individually, there is no guarantee that any particular string, not even the best in the population, will actually be chosen in any given generation. This sampling error can act as a significant source of noise. The problem is well-known: De Jong suggested ways to overcome it in his 1975 thesis. The neatest solution is Baker's Stochastic Universal Sampling (SUS) algorithm (Baker, 1987), which produced the results of Figures 3.1 and 3.2. Figure 3.3 shows the difference in results for the two methods with fitness proportional selection. The rate of take-over of the best value is reduced, a reflection of the fact that the roulette wheel simulation lost the best value from the population an average of 9.1 times per run. Conversely, the worst value current in the population increases more rapidly, because it is quite likely for poor strings to be missed by the random selection. Both effects are likely to be deleterious to performance.

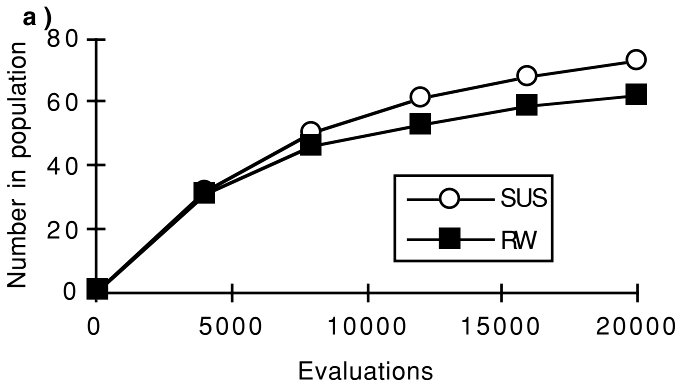

Figure 3.3a) Take-over rates for simple FPS, using roulette wheel (RW) and Baker's Stochastic Universal Sampling algorithm (SUS).

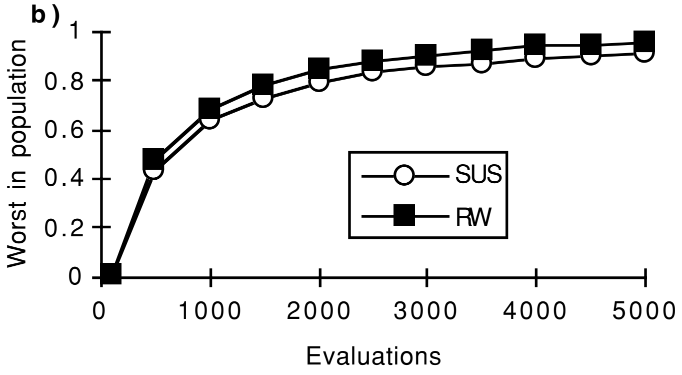

Figure 3.3b) Rise in the worst value in the population.

Baker's algorithm does the whole sampling in a single pass, and requires only one random number. The essence is to sum up the expected values, crediting the current string with an offspring every time the total goes past an integer. Thus if the initial random number is 0.7, and the first string expects 1.4 offspring, it will get two, since the total will be 2.1. If the random number is less than 0.6, it will get only one, since the total will be less than 2.

```
num = rand();
picked =1;
for (i=0; i<N; i++)
{
        num+= expected_off(i);
        while (picked < num);
        {
                parent[picked - 1] = i;
                  picked++;
        }
```

The only catch is that there will be sampling errors if the population is sorted, as it will be, for instance, for rank selection (see below). If two adjacent individuals each expect between 0.5 and 1 (exclusive) offspring, you can guarantee that one will get a child, while the other will not. The population should therefore be shuffled, adding to the complexity of the process. Many of the more obvious ways to do a shuffle produce a surprisingly biased result. An unbiased shuffle of the strings is given by the following code, where random_int returns an integer in the range i to N-1 inclusive:

```
for (i=0;i<N;i++)
{
        j=random_int(i ,N-1);
        temp = sorted_pop[j];
        sorted_pop[j]=sorted_pop[i];
        sorted_pop[i]=temp;
}
```

Alternatively, since copies of the strings will be required at some stage in the reproduction process, it may be better to shuffle them during the copying process. Baker's sampling algorithm can then be used to copy the winners back into the original array, ready for the application of reproduction operators:

```
for (i=0;i<N;i++)
{
        j = random_int(i ,N-1);
        new_pop[i] =sorted_pop[j];
        sorted_pop[j] =sorted_pop[i];
}
```

3.6 Ranking
Baker (1985) suggested rank selection in an attempt to overcome the deficiencies of the direct fitness based approach. First the population is ordered according to the measured fitness values. A new fitness value is then ascribed, inversely related to the string's rank. Two methods are in common use.

3.7 Linear ranking
The best string is given a fitness *s*, between 1 and 2. The worst string is given 2-*s*. Intermediate strings' fitness values are given by interpolation, assuming a *C* type array that starts at 0 for rank 1:

$$f(i) = s - \frac{(2i(s-1))}{(N-1)}$$

Since this prescription automatically gives an average fitness of 1, the fitness values translate directly as the expected number of reproductive opportunities. If *s* is set to 2, the worst string gets no chance of reproduction. In principle, *s* could be increased beyond 2 to achieve higher selection pressures, but then some of the worst strings would be given negative expected offspring. These could be truncated to zero, but then the remaining fitness values would need rescaling to give the correct total number of offspring. A simpler method of achieving higher selection pressures, which also gives some chance to the worst members of the population, is to use a non-linear ranking, such as that described in the next section. However, such high pressures are rarely needed. The selection pressure generated by linear ranking is proportional to *s*-1. Thus with *s*=1.1, convergence takes about 12,000 evaluations, with *s*=1.2 it takes 6000, and with *s*=2 it takes 1000.

In his 1985 paper, Baker discusses methods of preventing premature convergence. He proposes a measure of percentage involvement, being the proportion of the population that gets to produce an offspring in any generation. He suggests a target of at least 94%, which is given by linear ranking with *s* of about 1.1. Using *s*=2 gives only 75% involvement.

3.8 Exponential ranking
The best string is given a fitness of 1. The second best is given a fitness of *s*, typically about 0.99. The third best is then given s^2 and so on down to the last,

which receives $s^{(N-1)}$. The ascribed fitness values need to be divided by their average to give the expected number of offspring for each string.

```
s=0.99 ## or whatever ##
tot_fitness=0;
fitness=1;
for (i =0; i<N; i++)
{
        f(i) = fitness;
        tot_fitness += fitness;
         fitness *= s;
}
avg_fitness = tot_fitness/N;
for (i=0; i<N; i++)
        f(i) /= avg_fitness;
```

Those who are particularly conscious of cpu cycles may wish to trade storage for time and construct an array containing the expected number of offspring for each rank. Depending on the program's internal data structures, such an approach might do away with the need for any assignment of fitness at this stage of the selection procedure. However, some workers recommend varying the selection pressure during the run, in which case the array would need updating whenever s is changed.

The selection pressure is proportional to 1-s, thus s=0.994 gives twice the convergence rate of s=0.998. With s=0.999, convergence takes about 25,000 evaluations, with s=0.968, it takes about 700. Figure 3.4 shows a family of take-over curves, confirming that doubling the selection pressure halves the take-over time. For s below 0.98, the expression $N(1-s)$ gives a good approximation to the expected number of offspring for the best individual. In fact s=0.98 gives about 2.3 offspring to the best of a population of 100, but s=0.95 gives almost exactly 5, should that much pressure be wanted for some reason.

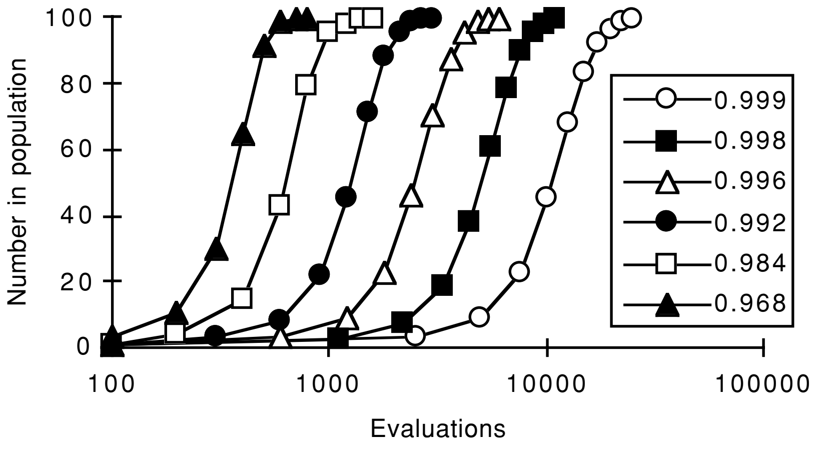

Figure 3.4. A family of take-over curves for exponential rank-based selection, at varying values of s.

The difference between the two ranking methods is illustrated in Figure 3.5. Figure 3.5a shows the expected number of offspring against rank for linear ranking with *s*=1.8 and exponential ranking with *s*=0.986. The expectation for the best individual is very similar, and as a consequence, so is the take-over time and the growth rate in the presence of mutation. However, the rate of loss of the worst value is considerably less for exponential ranking, as shown in Figure 3.5b, because of the increased chance given to weaker strings. For equivalent growth rates, exponential ranking ought to give a more diverse population, albeit at the expense of more average individuals.

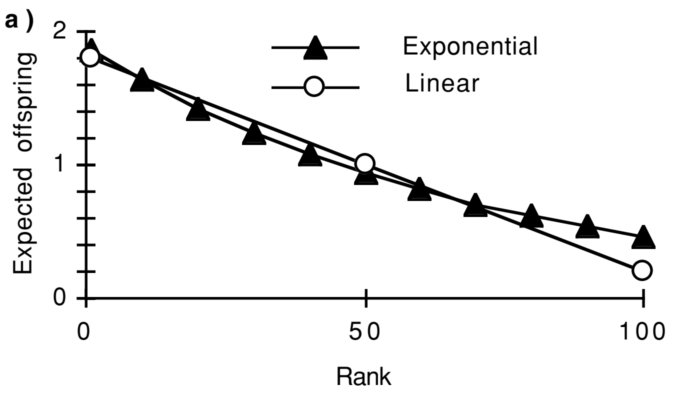

Figure 3.5a) Expected number of offspring for individual of given rank, for exponential ranking with s=0.986, and linear ranking with s=1.8.

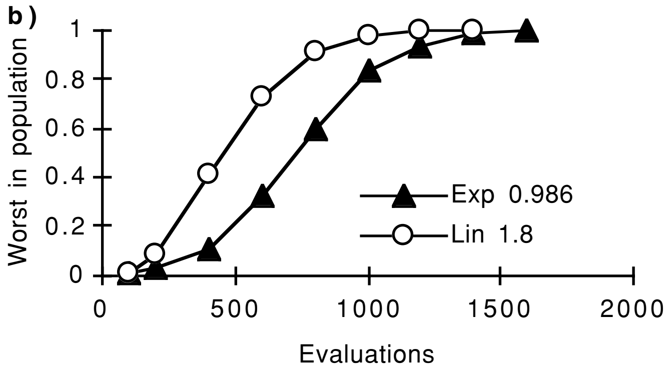

Figure 3.5b) Increase in worst value in the population, during take-over simulation.

Other kinds of non-linearity are possible, and possibly desirable. For instance, inverting the curve given by exponential ranking would improve the chances of above-average individuals at the expense of the worst. This would implement a softer version of the Top-n selection described below. Kuo and Hwang (1993) implement what they call disruptive selection, which uses a non-monotonic fitness function that gives more trials to good and bad strings, at the expense of intermediate values.

3.9 Tournament selection

In tournament selection, n individuals are chosen at random from the population, with the best being selected for reproduction. A fresh tournament is held for each parent-to-be. This method is quite popular, perhaps partly because of the echoes of the mating battles often seen in nature. Goldberg and Deb show that the expected result for a tournament with n=2 is exactly the same as linear ranking with s=2. That this is the case may be seen intuitively by considering various cases. Every string should expect to be picked twice. The best string in the population will win both its tournaments, while the worst will never win, and thus never be selected. A string with performance equal to the average will expect to win half its tournaments, and so be picked half as often as the best string, exactly as linear ranking specifies.

Tournament selection can be made to emulate linear ranking with s<2 by making it only probable that the better string will win. If the better string wins with probability of 0.5, the process reduces to random selection, with no bias in favour of better strings. This is equivalent to linear ranking with s=1. The conversion between probability in tournament selection and s in linear ranking is to double the probability, thus a probability of 0.8 is equivalent to s=1.6. The selection pressure generated by the tournaments may be increased by using n>2. With n=3, an average string can expect to win only a quarter of its tournaments. Since it should be selected for 3 tournaments, it can expect 0.75 offspring. The graph of expected offspring against rank becomes non-linear, resembling that produced by exponential ranking. Figure 3.6a shows the comparison: when the best string gets four offspring, exponential ranking still favours the worst strings at the expense of those somewhat above average.

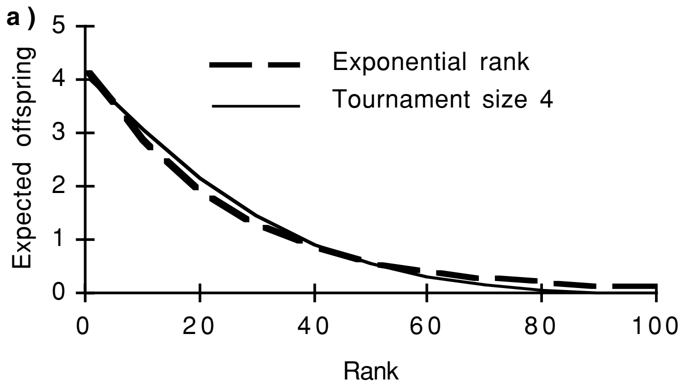

Figure 3.6a) Expected number of offspring for individual of given rank, for exponential ranking with s=0.96 and tournament selection with n=4.

Figure 3.6b shows the convergence of the population under tournament and linear ranking selection methods. The difference is caused by stochastic errors in tournament selection. Because each tournament is carried out individually, the method suffers from exactly the same sampling errors as roulette wheel selection and linear ranking, with Baker's selection procedure, should usually be used instead. It is interesting that Goldberg and Deb, who omitted consideration of stochastic effects, concluded that tournament selection was preferable to linear

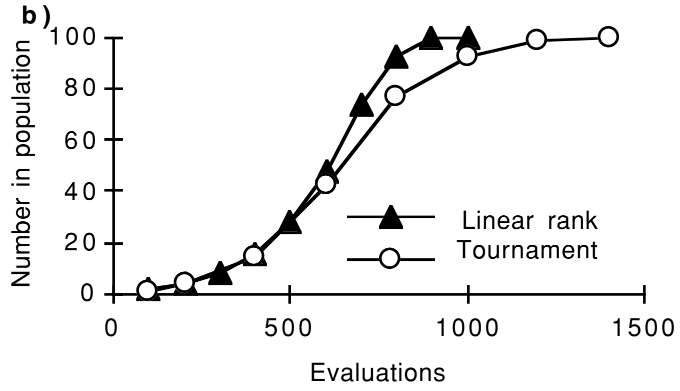

Figure 3.6b) Take-over rates for linear ranking, s=2 and tournament selection, n=2.

ranking because of the lower time complexity. This can be particularly significant in parallel models where a local tournament may be appropriate. Each processor can carry out the tournaments needed to generate the strings it will evaluate, whereas linear ranking selection needs to be carried out centrally, with the remaining processors waiting for the results to be distributed. Some models such as ASPARAGOS (Gorges-Schleuter, 1989) use a spatially distributed population, with tournaments being held only amongst nearby members of the population. However, those using tournament selection should be aware of the implied sampling errors.

Occasionally, holding tournaments may be the only way to do the evaluation, for instance when evolving game-playing programs. Generation of a rank ordering by playing every program against every other may be too time-consuming. In this case also, tournament selection is the natural way to proceed.

3.10 Genitor or steady state models

A major argument in the GA camp centres on whether to replace the whole population at a go (generational model), or some subset, often one or two (incremental model). Whitley, with his Genitor system, is one of the major proponents of one-at-a-time, or steady-state reproduction (Whitley and Knuth, 1988). Any of the above methods of selection could be used to pick the parents of the single offspring, but Whitley uses linear ranking (Whitley, 1989), and provides a neat algorithm for picking an individual from the population with the desired probability.

$$n = N \times \frac{s - \sqrt{(s \times s - 4 \times (s-1) \times rand())}}{2 \times (s-1)}$$

Steady-state reproduction inevitably carries the same kind of sampling errors as roulette wheel selection. Some users therefore employ stochastic tournament selection to pick the two parents, since it can't add any extra error and it obviates the need to sort the population.

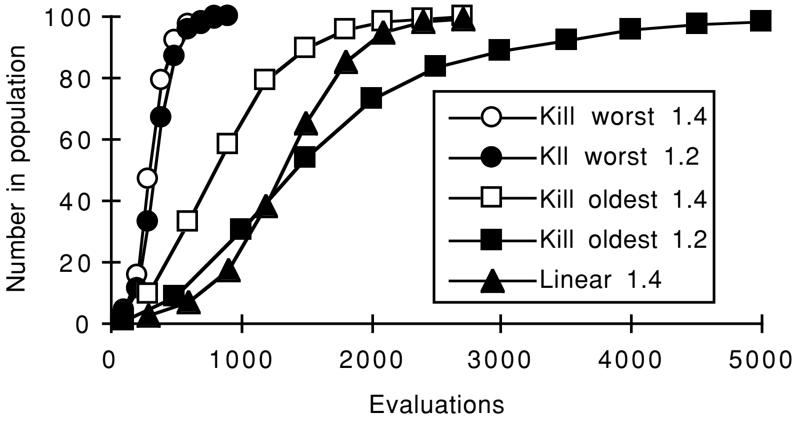

Figure 3.7. Take-over rates for incremental algorithms, with kill-worst and kill-oldest, and generational model using linear ranking.

Goldberg and Deb show that the Genitor model produces very high selection pressure. Most of this comes from always replacing the worst member of the population. Changing the linear ranking scale factor has very little effect. There are various ways to reduce the selection pressure. One is to pick an individual from the population to be replaced at random. This produces growth curves almost identical to a generational model with the same selection pressure (Syswerda, 1991).

An alternative is to replace the oldest member of the population. Figure 3.7 compares kill-oldest with kill-worst. In all the graphs, the x-axis units are evaluations, to allow direct comparison between the different population models. Kill-oldest is comparable with a generational model of the same selection pressure. However, it is faster at the start of run, and slower to finish off, probably the opposite of what is desirable. As might be expected, loss of the worst is much more rapid as well (not shown). Slow finishing is the consequence of the sampling errors, like those of roulette wheel selection, that inevitably result from breeding one at once. With s=1.4, kill-oldest lost the best value an average of 2.5 times per run.

Figure 3.8 shows that, for growth in the presence of mutation, kill-oldest is closer to kill-worst than to a generational model with the same nominal selection pressure. One reason may be that the effects of sampling error are less significant in this task — in the take-over task, it is rather crucial if the best individual is lost. Another may be a manifestation of the claim sometimes made for steady-state reproduction that they can exploit good new individuals more rapidly, because they are immediately available for reproduction. In the simple take-over example, this has no effect, because the additional copies of better strings are exactly balanced by the increase in average fitness (De Jong and Sama, 1991). Even where progress is obtained because of genetic operators, the effects may not be that big, because although a new string *may* be selected immediately, it is more likely to have to wait a significant fraction of N evaluations at normal selection pressures. Because of the stochastic sampling effects, it may have to wait more than N evaluations, if it is allowed to live that long, or even fail to be selected at all. Clearly, the stronger the selection pressure, the more significant the effect becomes.

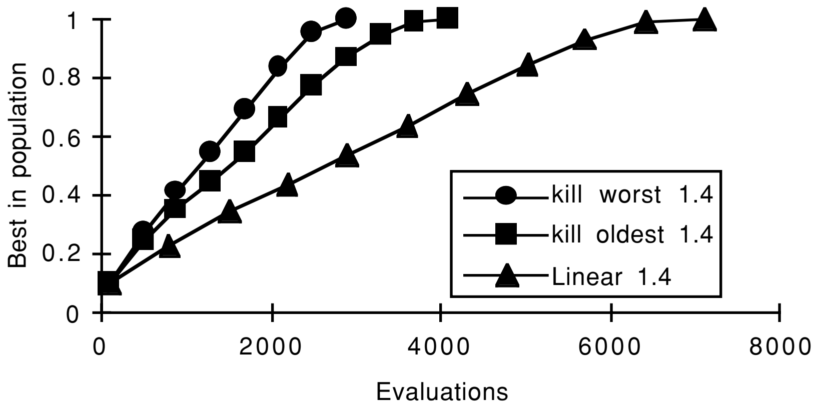

Figure 3.8. Growth rates in the presence of mutation for incremental and generational models, all using linear rank selection with s=1.4.

A third alternative is to delete worse members of the population preferentially, for instance by inverse ranking. Syswerda typically uses this method, and shows that it produces growth curves similar to kill-worst (Syswerda, 1991). However, he is using inverse exponential ranking with *s*=0.9, which is a very high selection pressure. A more typical usage might be to select for reproduction from the top with linear ranking, *s*=1.2, and kill from the bottom with the same selection rate. If we ignore for a moment the possibility that it will be superseded during its lifetime, the best string can expect *s*/(2-*s*) offspring before being deleted. However, part of this comes from its increased life-expectancy, and during any period of N evaluations, equivalent to one cycle of a generational model, it will only expect 1/(2-*s*) offspring: 1.25 for *s*=1.2. Added to this will be a factor caused by the preferential deletion of worse individuals, which will cause the average fitness to rise. The net result is a growth rate that rises faster than *s*. Figure 3.9 shows that a steady-state model, with selection from top and bottom with *s*=1.2, produces a near identical growth rate to a generational model with linear ranking, *s*=1.4. However, the steady-state model with *s*=1.4 is faster than generational with *s*=1.8, while it is slower for smaller values of s. The approximate equivalence at *s*=1.2/1.4 appears to be genuine, and is not affected by things such as the size of the mutation. For those wishing to compare the techniques on their own problems, equivalent values for lower selection pressures are 1.13/1.2 and 1.085/1.1.

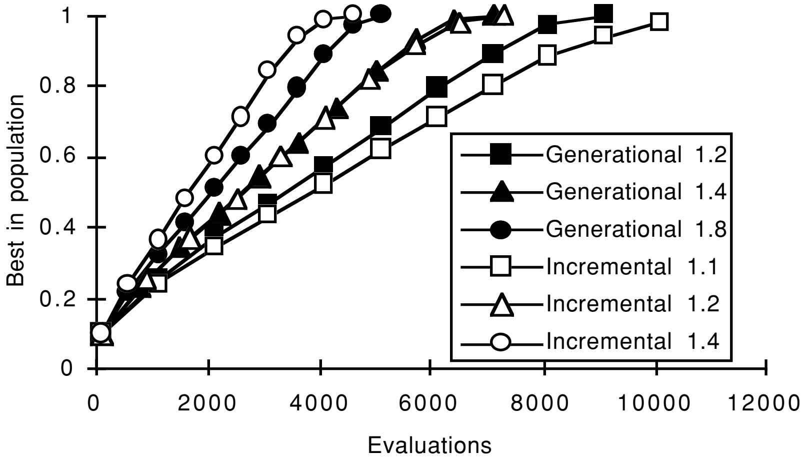

Figure 3.9. Growth rates for generational model, with linear ranking, and incremental model, with rank-based selection for reproduction, and inverse rank-based selection for deletion.

GA practitioners repeatedly claim significantly faster convergence using steady-state reproduction. In practice, this may often be the result of inadvertently high selection pressure acting on a fairly easy problem. However, another potentially crucial difference is that steady-state models are often run with the elimination of duplicates. Any new string is only admitted to the population if different from those already present. This will have a radical effect on the behaviour of the algorithm, which cannot be illustrated with the simple simulations used here, but is certainly worth trying on real problems.

A serious drawback with steady-state reproduction is the effect of noise in the evaluation function. This will be considered below, suffice it now to say that it does not do well.

3.11 Evolution strategy and evolutionary programming methods

The selection methods used in ES and EP algorithms are rather similar, producing high take-over rates. The approach differs from that typical of GAs, being more like selection of the fittest than fitness proportional reproduction. However, in some guises the effects can be very similar.

3.12 Evolution strategy approaches

There are two main methods of selection used in ESs, known as (m+l) and (m,l), where m is the number of parents and l is the number of offspring (for a review, see Hoffmeister and Bäck, 1992). The top m individuals form the next generation, selection being from parents and children in the (m+l) case, children only for (m,l). Typically, l is one to five times m (it obviously must be bigger than m for the (m,l) model). With these schemes, take-over by the best value is exponential. Thus for a (100,200) ES, it takes $\log_2(100) = 7$ generations, for (100+200), it takes $\log_3(100) = 5$ generations. The CHC genetic algorithm (Eshelman, 1991) uses an (m+m) selection procedure, combined with recombination operators designed to promote search.

3.13 Top-n selection

Some workers select the n best individuals, and give them each N/n offspring (Nolfi, 1990). This clearly has the potential for extremely rapid take-over — with n=10, the best value will take over in two generations. It differs from the ES (m,l) approach only in what is called the population, thus Top-n with n=50 and N=100 is equivalent to a (50,100) ES. Bäck and Hoffmeister (1991) term such selection algorithms as extinctive, in that some individuals are guaranteed to get no offspring. They show, as might be expected, that such harsh selection is appropriate for unimodal objective functions.

3.14 Evolutionary programming methods

The canonical EP selection algorithm is a stochastic version of a (m+l) ES. Each individual produces one offspring, each of the $2N$ individuals plays c others chosen at random (with replacement) in a tournament. The N with most wins forms the next generation. If the tournaments are deterministic, the result will converge to that of an $(N+N)$ ES as c increases. The size of c has little effect on the simple task used here: the best value always wins its competitions, and takes over the population in 7 generations.

As before, the selection may be softened by making the tournaments stochastic. Note a problem with terminology here: what is referred to above as a deterministic tournament, because the better individual always wins, Fogel refers to as a stochastic tournament (Fogel, 1994), because the opponents are picked at random. Here, a stochastic tournament is one in which the fitter individual may lose. One approach to this is to make the probability of the better string winning depend on the relative fitness of the two strings: $p_i = f_i/(f_i + f_j)$ (Fogel, 1988). This has the effect of reducing selection pressure to zero as the population becomes uniform, and produces a take-over curve remarkably similar to simple proportionate selection, Figure 3.10a. However, loss of the worst is more rapid, since poor strings initially get little chance to reproduce (not shown). Note, however, that since EP does not use recombination, there may be less requirement for diversity than in a GA.

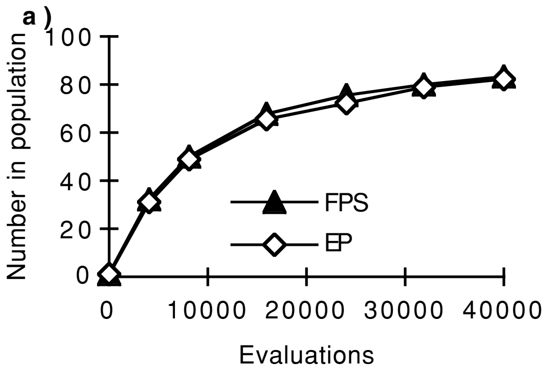

Figure 3.10a) Take-over rates for fitness proportional selection and stochastic evolutionary programming selection, with tournament size.

Figure 3.10b shows growth rates under mutation of a number of the selection methods described. Genitor produces the highest growth rate, but note that exponential ranking can be made to converge similarly fast, or much the most slowly, by varying the selection pressure. Linear ranking and linear scaling can also be set to give similarly slow growth rates (not shown). Despite the very rapid take-over given by the ES methods, growth rate is less spectacular, and actually very similar to that provided by simple fitness proportionate selection. This does relatively well because of the high number of offspring allocated to particularly fit individuals, not in general a good idea.

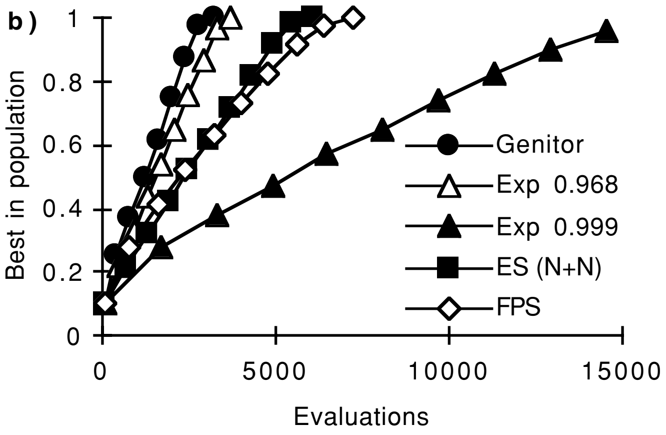

Figure 3.10b) Growth rates for Genitor (incremental, kill-worst), with s=1.2, exponential ranking, (100+100) ES and FPS.

3.15 The effects of noise

It is often claimed that Genetic Algorithms are relatively immune to the effects of noise in the evaluation function. However, it is to be expected that the various selection schemes would differ in their susceptibility. This was assessed by adding noise to the evaluation, and observing the effect on the growth in the presence of mutation. Another Gaussian random variable was added to the true value of each individual and used to allocate its number of offspring. The true value was then passed to any offspring, subject to the small mutation as before. In order to have a significant effect on the rate of convergence, it was found to be necessary to add noise with a standard deviation of 0.2: 10 times that of the mutation. This is not so much a signal-to-noise ratio as a noise-to-signal ratio, and gives some credence to the claimed noise immunity.

Figure 3.11a shows the effects of adding noise on two traditional GA methods, linear ranking and sigma scaling. Even with this level of noise, the time to convergence is less than halved. However, note that sigma scaling deteriorates rather less than the ranking method. The obvious conclusion is that sigma scaling is more noise tolerant, but this would be a mistake. The real reason for the result is quite subtle, but similar to that responsible for the growth rates in Figure 3.2b. The effect of the noise is to reduce the accuracy of the selection procedure. In the limit, if noise swamps the evaluation entirely, all individuals would expect one offspring. Here, the best individual can expect somewhat more than one, but less than it should get in the absence of noise. With ranking and s=1.8, it will therefore get somewhere between 1 and 1.8 offspring. Figure 3.11b shows the actual number of offspring allocated to the best individual, averaged over the 100 runs. Linear ranking scores about 1.2 at the start of run. Sigma scaling sets the baseline according to the spread of values in the population, which will be determined mostly by the noise. Lucky individuals can appear 2 or 3 standard deviations above the mean. Because there is not the upper limit imposed by ranking, the best individual averages higher, about 1.5 initially in this case. The faster growth rate is therefore effectively a case of premature convergence, but since the problem is so simple, there is only the correct solution for it to converge to.

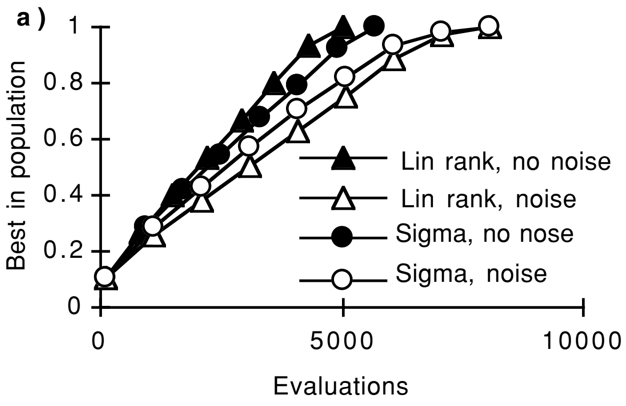

Figure 3.11a) Growth rates with and without noise for linear ranking with s=1.8 and sigma scaling with s=4.

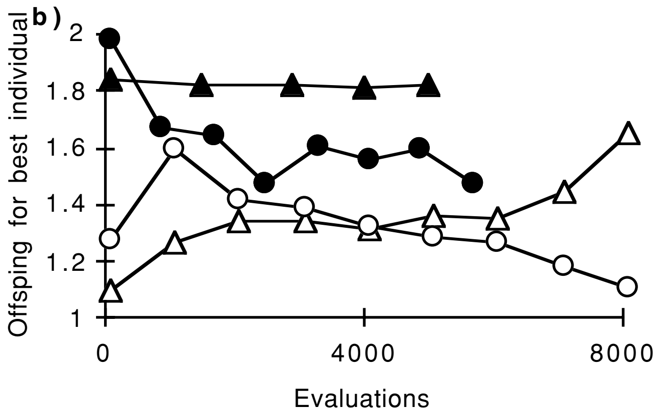

Figure 3.11b) Measured number of offspring for the best individual in the population during the same simulations.

Figure 3.12 shows the effects on steady-state reproduction. Because Genitor kills the apparent worst in the population, lucky individuals that got a much better evaluation than they merited will remain in the population. The effects on convergence rate are disastrous. Killing the oldest performs much better, echoing the findings of Fogarty (1993). Not shown is the effect on a model using inverse rank based deletion of worse individuals. As expected, it also deteriorates, so that steady state with *s*=1.2 from both ends converges in around 14,000 evaluations, compared with 10,000 for generational linear ranking, with *s*=1.4. As noted above, these two are almost identical in the absence of noise. Compared with kill-oldest, the deterioration is somewhat worse.

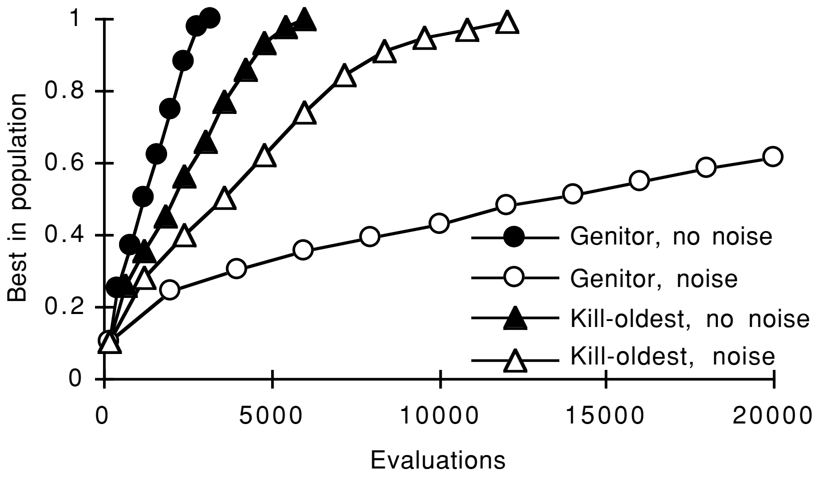

Figure 3.12. Effect of adding noise to growth rate of Genitor (kill-worst) and kill-oldest incremental models.

In theory, the noise sensitivity of steady-state models may be reduced by re-evaluating existing members of the population. One approach is to pick an individual at random for each new offspring produced. Since most of the problem appears to come from fortuitous individuals that are ranked higher than they should be, it might seem better to select high ranks preferentially, as for reproduction. The old and new evaluations should be averaged, to improve the estimate of the true value. When tried, the latter approach did perform slightly better, but the improvement given by either method was not nearly sufficient to discount the extra evaluations. They just took slightly less than twice as many. The most robust way of handling noise in incremental algorithms appears to be to kill the oldest.

Figure 3.13 tells a similar story for the ES selection methods. Without noise, (100+100) converges rapidly. Allowing the parents to pass to the next generation ensures that nothing is lost, giving rapid gains on the simple task. A (100+200) ES converges more rapidly in terms of generations, but requires more evaluations in total. Comparison with the (100,200) model illustrates the advantage of conserving the best parents. In the presence of noise, however, the conservative approach again fails. The (100,200) ES deteriorates by a similar amount to other generational techniques, (100+100) deteriorates dramatically. The deterministic version of EP selection performs very similarly to a (100+100) ES, depending on the number of tournaments held (not shown).

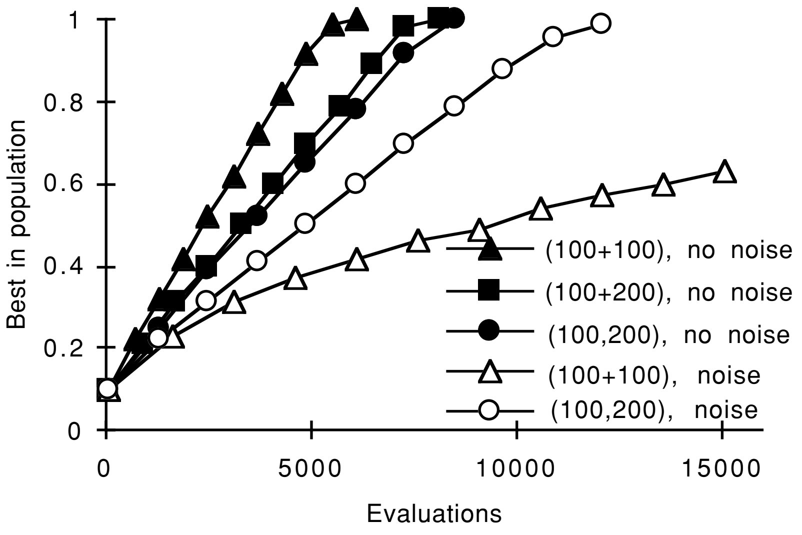

Figure 3.13. Effect of noise on growth rate of ES selection methods.

Conclusions

By now it should be apparent that there are fewer significant differences between the various selection schemes than might be thought. The decisions to be made include the following:

1. Whether to use direct fitness measures, with appropriate scaling, or rank-based selection. The latter provides better control over selection pressure, at the expense of the link between fitness and reproductive success.

2. Whether to use a generational or incremental model. The latter suffers in the presence of noise and also from the same kind of sampling errors as the roulette wheel. Its benefit is the ability to exclude duplicates, the advantage of which could not be illustrated here.

3. Whether to introduce a deliberate non-linearity between fitness and allocated offspring. Exponential rank selection benefits the worst members, at the cost of above-average individuals. Top-n selection goes to the opposite extreme, giving all the offspring to the top few. It may be appropriate to alter the balance during a run, keeping worse individuals initially, and moving towards harsher selection.

4. Whether to use fitness or rank proportional reproduction (GA) or selection of the fittest (ES/EP). It is probably relevant that the latter stress the importance of mutation as a search operator, while GAs rely on recombination.

A common technique that has not been mentioned yet is the elitist strategy (De Jong, 1975), which simply ensures that the best individual survives into the next generation. This is not simply to ward off errors in sampling, since the best string may be correctly selected, but be disrupted by recombination or mutation. It is a conservative strategy, often found to give an improvement in performance, but again suspect with noisy evaluations. It can also hinder progress, by anchoring the population to a local maximum. A softer alternative is to lose the best if no progress has been made for some number of generations (e.g. 5).

Selection is only one part of the algorithm, and decisions about which to use need to be made in parallel with decisions about recombination operators. For instance, Eshelman (1991) deliberately combines a conservative "survival of the fittest" selection method with disruptive recombination operators. It isn't possible to say which is best without defining evaluation criteria. Thus Goldberg and Deb preferred tournament selection to linear ranking because they considered time complexity, but not stochastic effects. Freisleben and Härtfelder (1993) used a meta-level GA to tune the parameters of another GA. It was able to alter parameters such as population size, mutation rates and the selection method. The choice was tournament, rank and two forms of FPS. Tournament was the clear winner, which is odd, because it is just rank with added noise.

How could adding noise help? Their task was learning the weights for a neural net simulation. This is plagued with symmetry problems — many different genetic string produce identical nets, because the order of hidden units is immaterial, while that of the genetic string is not. Crossover therefore does not work successfully. The GA has to decide which of the permutations to use. The most likely explanation for Freisleben and Härtfelder's result is that the noise of tournament selection allowed one permutation to get its nose ahead of the rest and take over the population, thus resolving the problem. Had the meta-GA had available recombination operators able to handle the permutations, it might well have chosen a different selection procedure. Hopefully this chapter willbe of some assistance in making such decisions.

Acknowledgements
This work was partly supported by grant no. GR/H93828 from the UK Science and Engineering Research Council.

References

Bäck, T. and Hoffmeister, F. Extended selection mechanisms in genetic algorithms. Pages 92-99 of Proceedings of the fourth international conference on Genetic Algorithms, Belew, R.K. and Booker, L. (Eds), Morgan Kaufmann. 1991.

Baker, J.E. Adaptive selection methods for Genetic Algorithms. Pages 101-111 of Proceedings of an international conference on Genetic Algorithms, Grefenstette, J.J. (ed), Lawrence Earlbaum. 1985.

Baker, J.E. Reducing bias and inefficiency in the selection algorithm. Pages 14-21 of Proceedings of the second international conference on Genetic Algorithms, Grefenstette, J.J. (ed), Lawrence Earlbaum. 1987.

De Jong K.A. An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, Department of Computer and Communication Studies, University of Michigan, 1975.

De Jong K.A. and Sarma J. Generation gaps revisited. In Foundations of Genetic Algorithms 2, Whitley, D. (Ed). Morgan Kaufmann. 1991.

Eshelman, L.J. The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In Foundations of Genetic Algorithms, Rawlins, G.J.E. (ed), Morgan Kaufmann. 1991.

Fogel, D.B. An evolutionary approach to the travelling salesman problem. Biological Cybernetics **60**, 139-144, 1988.

Fogel D.B. An introduction to simulated evolutionary optimization. IEEE Transactions on Neural Networks **5**, 3-14, 1994.

Fogarty, T.C. Reproduction, ranking, replacement and noisy evaluations: experimental results. Technical report, Faculty of Computer Studies and Mathematics, University of the West of England, 1993.

Freisleben, B. and Härtfelder, M. Optimization of genetic algorithms by genetic algorithms. In Artificial Neural Networks and Genetic Algorithms, Albrecht, R.F. and Reeves, C.R. and Steele, N.C. (Eds). 1993.

Goldberg, D.E. and Richardson, J. Genetic algorithms with sharing for multimodal function optimization. Pages 41-49 in Genetic algorithms and their applications: proceedings of the second international conference on Genetic Algorithms, Lawrence Earlbaum. 1987.

Goldberg, D.E, and Deb, K. A comparative analysis of selection schemes used in Genetic Algorithms. Foundations of Genetic Algorithms, Rawlins, G.J.E. (Ed), Morgan Kaufmann, 1991.

Gorges-Schleuter, M. ASPARAGOS: An asynchronous parallel genetic optimization strategy. Pages 422-427 of Proceedings of the third international conference on Genetic Algorithms, Schaffer, J.D. (Ed), Morgan Kaufmann, 1989.

Hoffmeister, F. and Bäck, T. Genetic algorithms and evolution strategies: similarities and differences. Technical report SYS-1/92, University of Dortmund, 1992.

Kuo, T. and Hwang, S.-Y. A genetic algorithm with disruptive selection. In proceedings of the fifth international conference on Genetic Algorithms, Forrest, S. (Ed), 1993.

Nolfi, S., Elman, J.L. and Parisi, D. Learning and Evolution in Neural Networks, Technical report CRL TR 9019, UCSD, July 1990.

Schaffer, J.D., Some effects of selection procedures on hyperplane sampling by genetic algorithms. Pages 89-103 of Genetic Algorithms and Simulated Annealing, Davis, L. (Ed), Pitman, London. 1987.

Syswerda, G. A study of reproduction in generational and steady-state genetic algorithms. In Foundations of Genetic Algorithms, Rawlins, G.J.E. (ed), Morgan Kaufmann. 1991.

Todd P.M. and Miller. G.F. On the sympatric origin of species: mercurial mating in the quicksilver model. Pages 547-554 of Proceedings of the fourth international conference on Genetic Algorithms, Belew, R.K. and Booker, L. (Eds), Morgan Kaufmann. 1991.

Whitley, D., and Knuth, J. GENITOR: a different genetic algorithm. Pages 118-130 of Proceedings of the Rocky Mountain Conference on Artificial Intelligence, Denver, Colorado. 1988.

Whitley, D. The Genitor algorithm and selection pressure: why rank-based allocation of trials is best. Pages 116-121 of Proceedings of the third international conference on Genetic Algorithms, Schaffer, J.D. (ed), Morgan Kaufmann. 1989.