Cranfield UNIVERSITY

Multi-Objective, Probabalistic Selection Evolutionary Algorithms (MOPSEA)

Department of Aerospace, Power & Sensors

Report No. DAPS/EJH/56/2000

Dr Evan J. Hughes

September 2000

Royal Military College of Science
Shrivenham

Abstract

Real engineering optimisation problems are often subject to parameters whose values are uncertain or have noisy objective functions. Techniques such as adding small amounts of noise in order to identify robust solutions are also used. The process used in evolutionary algorithms to decide which solutions are better than others do not account for these uncertainties and rely on the inherent robustness of the evolutionary approach in order to find solutions.

In this paper, the processes needed to provide probabilities of selection are reformulated to begin to account for the uncertainties and noise present in the system being optimised. Single and multi-objective systems are considered along with parameter constraints and objective limits for both rank-based and tournament selection.

The formulations are straightforward to programme and reasonably efficient to process. The techniques are ideally suited to interactive,constrained, uncertain, noisy, multi-objective design and can be effective in reducing the disturbances to the evolutionary algorithm caused by noise in the objective function.

Keywords

Evolutionary Algorithm, Ranking, Multi Objective, Noise, Uncertainty, Robustness

Contents

1. Introduction

The use of evolutionary algorithms (EA's) in engineering is now becoming acceptable and widespread. As the use of the algorithms matures and migrates from academia into industry, often the scale and characteristics of the problems being solved are changing. The objective function is often no longer a well defined analytical function but a complex, nonlinear and often uncertain model of a plant or system. Many model coefficients are derived by experiment and are therefore subject to experimental errors and the coefficients are often implemented in the models by approximating their behaviour with a polynomial function that best fits the measured data. In systems such as surfaces subject to aerodynamic forces, the aerodynamic coefficients are often measured in a wind tunnel, giving approximate data for only a limited operating envelope. In real systems, the true coefficients will not be the same as measured and are often time dependent or correlated with platform motion.

These errors in the modelling are unavoidable and inevitably propagate into the outputs of the objective functions, the results of which are used to classify the quality of the individual solutions to the problem. All optimisation algorithms attempt to find the problem solution that gives the most favourable output from the objective functions. With complex systems, evolutionary algorithms are a useful tool in that they can tolerate highly nonlinear and noisy system models and objective functions and still provide reasonable suggested solutions [1].

This robustness to errors has also been exploited by artificially adding *noise* to the objectives in an attempt to identify solutions that are robust to noise and uncertainty in the real system [2, 3, 4]. Noise is also often present when trying to optimise hardware systems such as in robotics. Noise or uncertainty in the objectives tend to slow evolution and reduce solution quality.

Attempts to reduce noise by repeating objective calculations and then averaging or combining results have been tried [5], but often with many realistic problems, the time to re-evaluate is prohibitive and often the number of samples used to average must be very small and therefore subject to considerable error. Most evolutionary algorithms to date have accepted these problems as the robustness of the algorithms allows small errors to be tolerated.

Therefore we may form two categories of problem:

1. **Noisy:** Two successive evaluations of the same chromosome information return two different sets of objectives.

2. **Uncertain:** When comparing two different chromosomes, errors in the modelling or noise in the data set under investigation may cause the objective values returned to classify the wrong solution as being superior.

This paper takes a fresh look at the problems of uncertain and noisy systems, both with single and multiple objectives, in order to provide a selection process

that is aware of the uncertainties and noise. The techniques discussed form a small step towards creating algorithms that can address the problems associated with the different categories of noisy or uncertain problems.

2. Problem Definition

As most engineering problems have multiple objectives that must be satisfied, the work concentrates on multi-objective evolutionary algorithms (MOEA). Carlos Coello Coello maintains an excellent database of publications relating to multi-objective optimisation [6]. Many of the publications tackling engineering problems (e.g. [7]) use techniques such as MOGA [8] and NSGA [9]. These methods use ranking techniques to address the problems of non-domination, then use sharing to spread the solutions across the objective surface. The use of ranking is widespread in EA's to prevent good solutions taking over the population in the early generations of the algorithm. Van Veldhuizen and Lamont [10] has studied the benefits / disadvantages of a number of techniques, including MOGA and NSGA, and begun to define metrics for assessing MOEA performance. These have been developed in the context of noise-free and certain problems and similar work is needed to address noisy and uncertain problems but is beyond the scope of this paper.

In all evolutionary algorithms, the key medium to evolution is being able to take two potential solutions to a problem, test them in the problem domain against some performance metrics, then given some values relating to the performance of each, decide which solution is better than the other. With noisy or uncertain problems, we find that given the results of the performance metrics, unless they are very different, we cannot say for certain which solution is better. Thus we must now refer to the probability that one solution is better than the other. This paper aims to review the processes needed in order to assign a probability of selection for each solution, given that we can no longer make a crisp decision about solution superiority.

The process of ranking both single and multiple objectives, parameter constraints, objective limits, and fitness sharing are re-formulated to account for the uncertainty in the optimisation problems. The paper also takes a brief look at possible ways of augmenting tournament selection to account for noise and uncertainty with constraints, limits, and multiple objectives.

3. Comparing two uncertain fitness measurements

3.1. Introduction

In a noise free situation, if we have two fitness values, A and B, and are trying to minimise, the lower value is always superior. However, if we know the fitness values are subject to noise, even if the measured fitness A is less than the measured fitness B, the mean of the distribution from which A is drawn may be greater than the mean of the distribution from which B is drawn. Therefore we would make the wrong decision. In the presence of noise, if we choose the simple case of *take the best measured objective*, we need to quantify the probability that we have made the wrong decision.

3.2. Analysis

3.2.1. Distributions with known mean

Figure 3.1 shows two known distributions with means μ_A and μ_B respectively, and a point of interest, x, generated at random from distribution A. If we generate a sample y from distribution B and it lies in the shaded area of B to the left of x, when we compare x and y, y will be less than x and therefore superior. We will therefore make the wrong decision. The probability of making a wrong decision given sample x may be found by integrating the shaded area and multiplying by the probability of x occurring. Therefore, for all possible values of x, the probability of the comparison of two samples giving the wrong decision is shown in (3.1), where $\text{pdf}_A(x)$ is the probability density function of fitness value A and $\text{cdf}_B(X <= x)$ is the cumulative distribution function for fitness value B.

$$P(\text{wrong decision}) = \int_{-\infty}^{\infty} \text{pdf}_A(x) \cdot \text{cdf}_B(X <= x) dx \tag{3.1}$$

In many real engineering problems, measurement noise is often Gaussian. The following derivations will be for Gaussian noise, although the analysis can be performed for any two distributions (e.g. a Gaussian and a Rayleigh). Equation 3.2 shows (3.1) expressed for two Gaussian distributions.

$$P(\text{wrong}) = \int_{-\infty}^{\infty} \left(\frac{1}{\sigma_a \sqrt{2\pi}} e^{-\frac{(x-\mu_a)^2}{2\sigma_a^2}} \int_{-\infty}^{x} \frac{1}{\sigma_b \sqrt{2\pi}} e^{-\frac{(y-\mu_b)^2}{2\sigma_b^2}} dy \right) dx \tag{3.2}$$

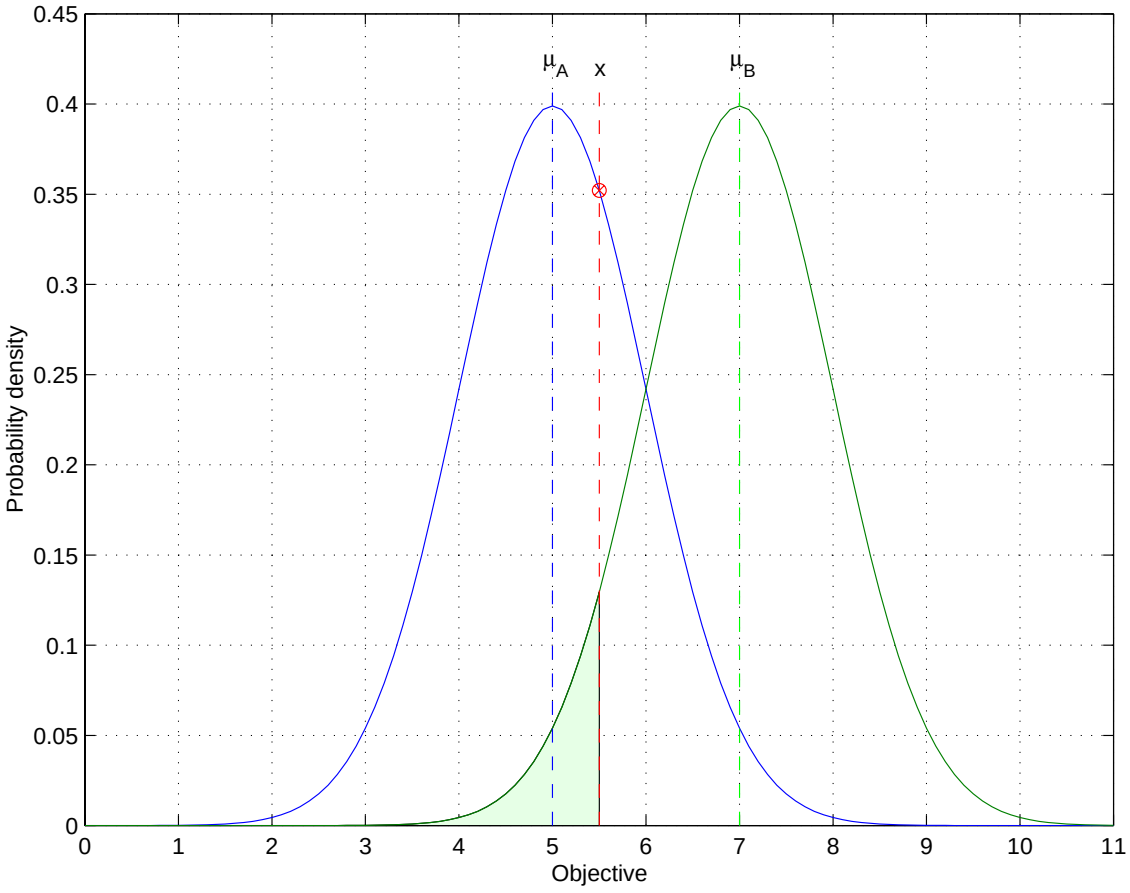

Figure 3.1: Probability that samples from distributions may be classified wrongly

Equation 3.3 has the parameter axis shifted to make distribution B centred around zero and then B is normalised, modifying distribution A accordingly. Equation 3.4 has been simplified with the replacements $m = \frac{(\mu_a - \mu_b)}{\sigma_b}$ and $s = \sigma_a / \sigma_b$.

$$P(\text{wrong}) = \int_{-\infty}^{\infty} \left(\frac{\sigma_b}{\sigma_a \sqrt{2\pi}} e^{-\frac{(x - \frac{(\mu_a - \mu_b)}{\sigma_b})^2 \sigma_b^2}{2\sigma_a^2}} \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right) dx \tag{3.3}$$

$$P(\text{wrong}) = \int_{-\infty}^{\infty} \left(\frac{1}{s\sqrt{2\pi}} e^{-\frac{(x-m)^2}{2s^2}} \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right) dx \tag{3.4}$$

3.2.2. Distributions with unknown mean

If we have a pair of samples from distributions with known characteristics and spread, but unknown means, we need to be able to calculate the probability that although sample A is less than sample B say, the mean of distribution B is less than the mean of distribution A. This will give us a probability of making the wrong decision. Figure 3.2 shows a scenario with two Gaussian distributions.

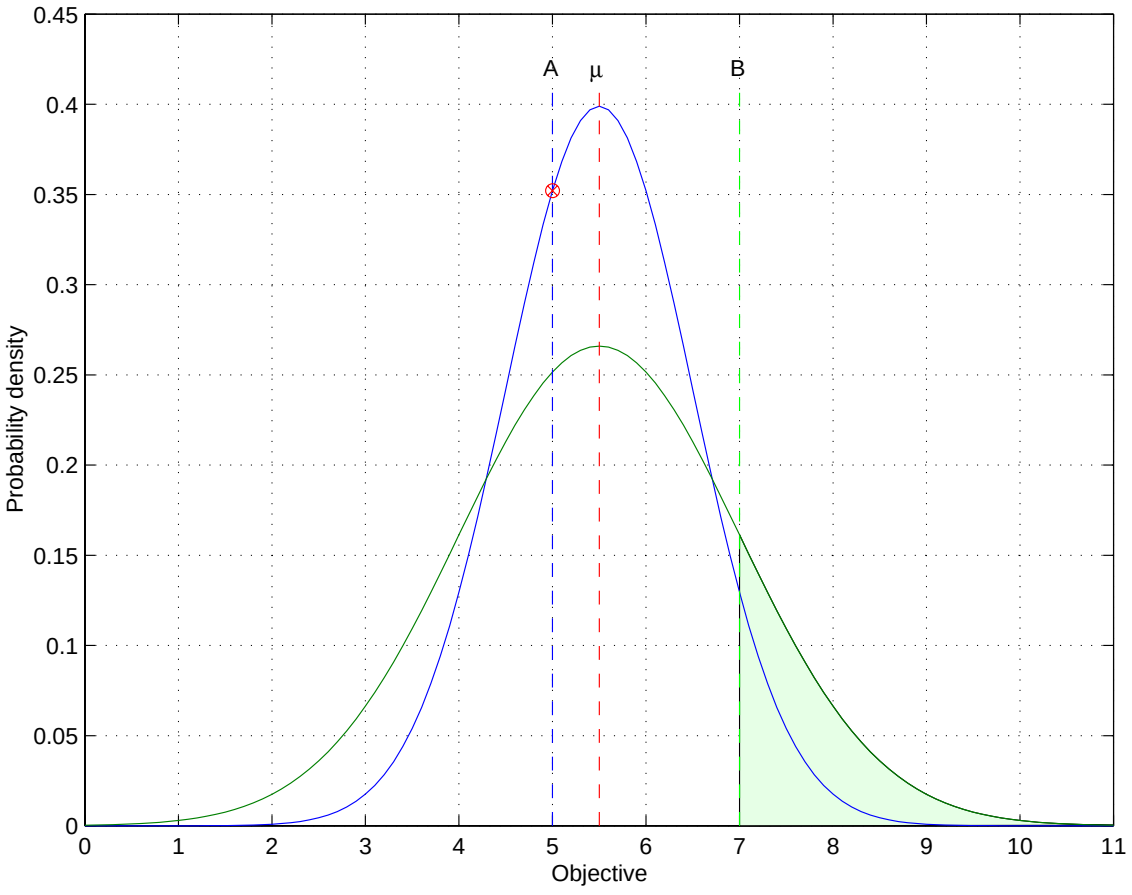

Figure 3.2: Choice between two noisy values

Here A and B are the measurements that were observed, and μ is an arbitrary point. The observed value A was less than B and is therefore superior. If the mean

of A was actually at point μ, the mean of B would have to be in any position to the left of μ for us to make the wrong decision. We can calculate the probability of the distributions being in this location as the probability of value A occurring, given μ_a, multiplied by the probability that μ_b is less than μ_a, shown as the shaded region on figure 3.2. This may be described mathematically as

$$P(\text{wrong decision}) = \int_{-\infty}^{\infty} \text{pdf}_A(A-\mu) \cdot \text{cdf}_B((X-\mu) > (B-\mu))d\mu \tag{3.5}$$

With Gaussian distributions, we may write this as

$$P(\text{wrong}) = \int_{-\infty}^{\infty} \left(\frac{1}{\sigma_a\sqrt{2\pi}} e^{-\frac{(A-\mu)^2}{2\sigma_a^2}} \int_{(B-\mu)}^{\infty} \frac{1}{\sigma_b\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma_b^2}} dy \right) dx \tag{3.6}$$

Equation 3.7 has the limits on the inner integration adjusted, as the Gaussian distribution is symmetrical $\text{pdf}(a) = \text{pdf}(-a)$ and $\text{cdf}(a,\infty) = \text{cdf}(-\infty,-a)$. The axis is shifted to make sample point $B = 0$ then distribution B is normalised, modifying distribution A accordingly. Equation 3.8 has been simplified with the replacements $m = \frac{(A-B)}{\sigma_b}$ and $s = \sigma_a/\sigma_b$.

$$P(\text{wrong}) = \int_{-\infty}^{\infty} \left(\frac{\sigma_b}{\sigma_a\sqrt{2\pi}} e^{-\frac{(\mu-\frac{A-B}{\sigma_b})^2\sigma_b^2}{2\sigma_a^2}} \int_{-\infty}^{\mu} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right) d\mu \tag{3.7}$$

$$P(\text{wrong}) = \int_{-\infty}^{\infty} \left(\frac{1}{s\sqrt{2\pi}} e^{-\frac{(\mu-m)^2}{2s^2}} \int_{-\infty}^{\mu} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right) d\mu \tag{3.8}$$

It is clear that (3.8) is now in the same form as (3.4) and that the subsequent analysis is equivalent, with $x = \mu$. This is only true as the Gaussian distribution is symmetrical. With other distributions, the probabilities when the mean is known may be different to when the mean is not known. We may now use the error function

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{t^2} dt \tag{3.9}$$

to give

$$\begin{aligned} P(\text{wrong}) &= \int_{-\infty}^{\infty} \left(\frac{1}{s\sqrt{2\pi}} e^{-\frac{(x-m)^2}{2s^2}} \frac{1+\text{erf}(\frac{x}{\sqrt{2}})}{2} \right) dx \\ &= \frac{1}{2} + \frac{1}{2s\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{(x-m)^2}{2s^2}} \text{erf}(\frac{x}{\sqrt{2}})\, dx \end{aligned} \tag{3.10}$$

Unfortunately (3.10) is difficult to integrate directly. An alternative approach is to recognise that the difference between two Gaussian distributions is also Gaussian but with a mean value that is the difference between the means of the two distributions and a variance which is a sum of the two variances (Cramer's Theorem), i.e.,

$$N(\mu_a, \sigma_a^2) - N(\mu_b, \sigma_b^2) = N(\mu_a - \mu_b, \sigma_a^2 + \sigma_b^2) \tag{3.11}$$

If A dominates B in a minimisation sense, then the area under the resulting curve from zero to infinity will give the probability that the decision that A dominates B is wrong. If we normalise B to give

$$N(\frac{\mu_a - \mu_b}{\sigma_b}, \frac{\sigma_a^2}{\sigma_b^2}) - N(0,1) = N(\frac{\mu_a - \mu_b}{\sigma_b}, \frac{\sigma_a^2}{\sigma_b^2} + 1) \quad (3.12)$$

$$= N(m, s^2 + 1) \quad (3.13)$$

Then the probability of being wrong is

$$P(\text{wrong}) = \frac{1}{\sqrt{2\pi\,(s^2+1)}} \int_0^\infty e^{-\frac{(x-m)^2}{2(s^2+1)}} dx \quad (3.14)$$

$$= \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{m}{\sqrt{(2+2s^2)}}\right) \quad (3.15)$$

Equation 3.15 can be shown to be equal to (3.10) by numerical integration.

3.3. Numerical approximation of probability

Figure 3.3 shows a plot of (3.10) with probability of error plotted with respect to m for different values of s. If $A = B$, the probability returned will be $P = 0.5$. As the ratio s increases, the curve flattens out. It must be remembered that the parameter m is normalised with respect to σ_b and therefore unless $\sigma_b = 1$, m will not represent the absolute difference between samples A and B.

We can use the above derivation in an evolutionary algorithm in a number of ways. If the objective function is very quick to calculate, we can evaluate each chromosome a number of times and estimate the mean and standard deviation of the objective value. We can then compare the objectives based on the estimates of their statistics using the equations in section 3.2.1. If we cannot afford to do multiple evaluations for each chromosome (often the case), we can choose a random chromosome before running the EA and perform multiple evaluations to estimate the noise standard deviation (and possibly noise distribution). This estimate may be used subsequently for all the comparisons of individual samples using the equations in section 3.2.2. If the noise statistics are known to be nonlinear, it may be advantageous to either re-estimate the statistics every few generations from an average chromosome, or even from the current population. When the same standard deviation is used for comparing two objective values, $\sigma_n = \sigma_a = \sigma_b$ therefore $s = 1$. Thus the probability is only determined by the value of m.

As the case of $s = 1$ is likely to be the most commonly used, we can tailor the equations specifically. The equations are calculated as the probability of being wrong in minimisation, this is the same as the probability of acceptance in maximisation. Thus the probability of sample A dominating sample B in maximisation ($P(A > B)$) is

$$P(A > B) = \frac{1}{2} + \frac{1}{2}\operatorname{erf}(\frac{m}{2})$$

$$= \frac{1}{2} + \frac{1}{2}\operatorname{erf}(\frac{A - B}{2\sigma_n}) \quad (3.16)$$

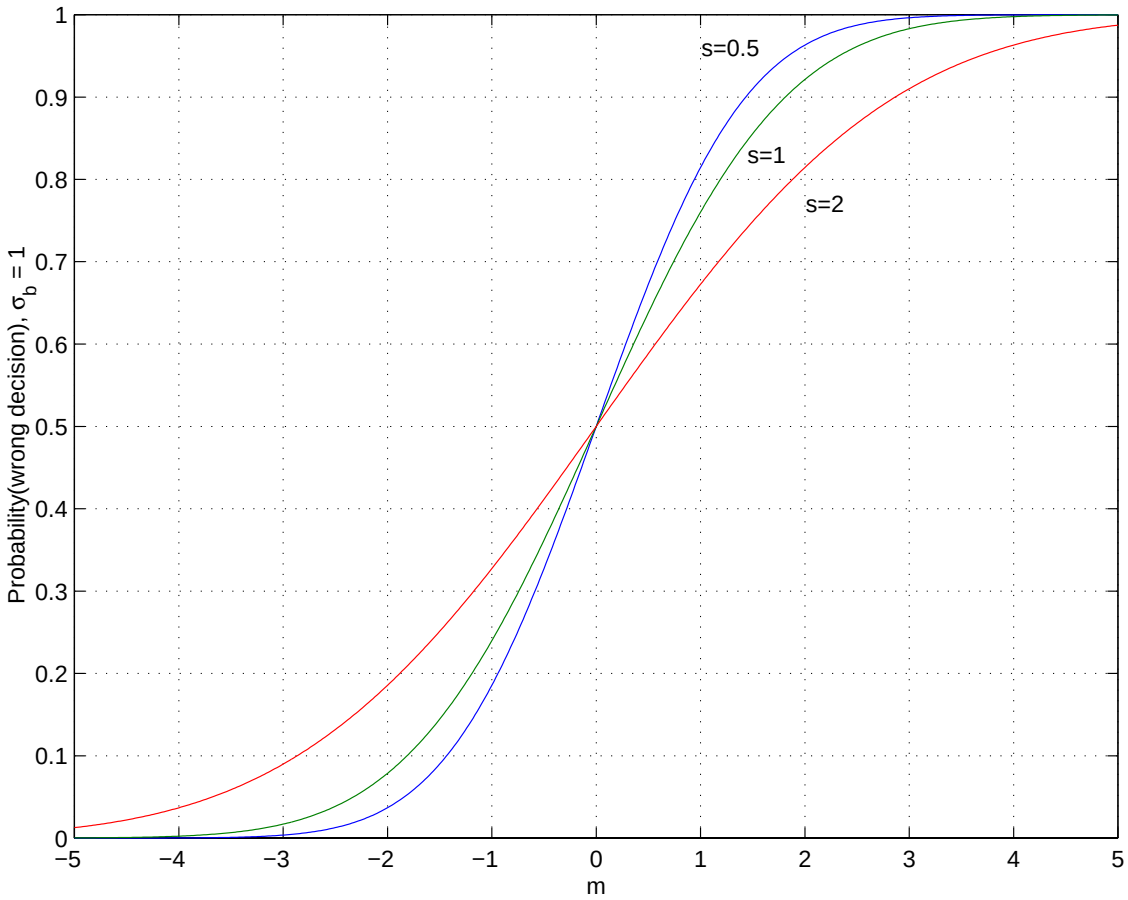

Figure 3.3: Probability of wrong decision against parameter m for different standard deviation ratios

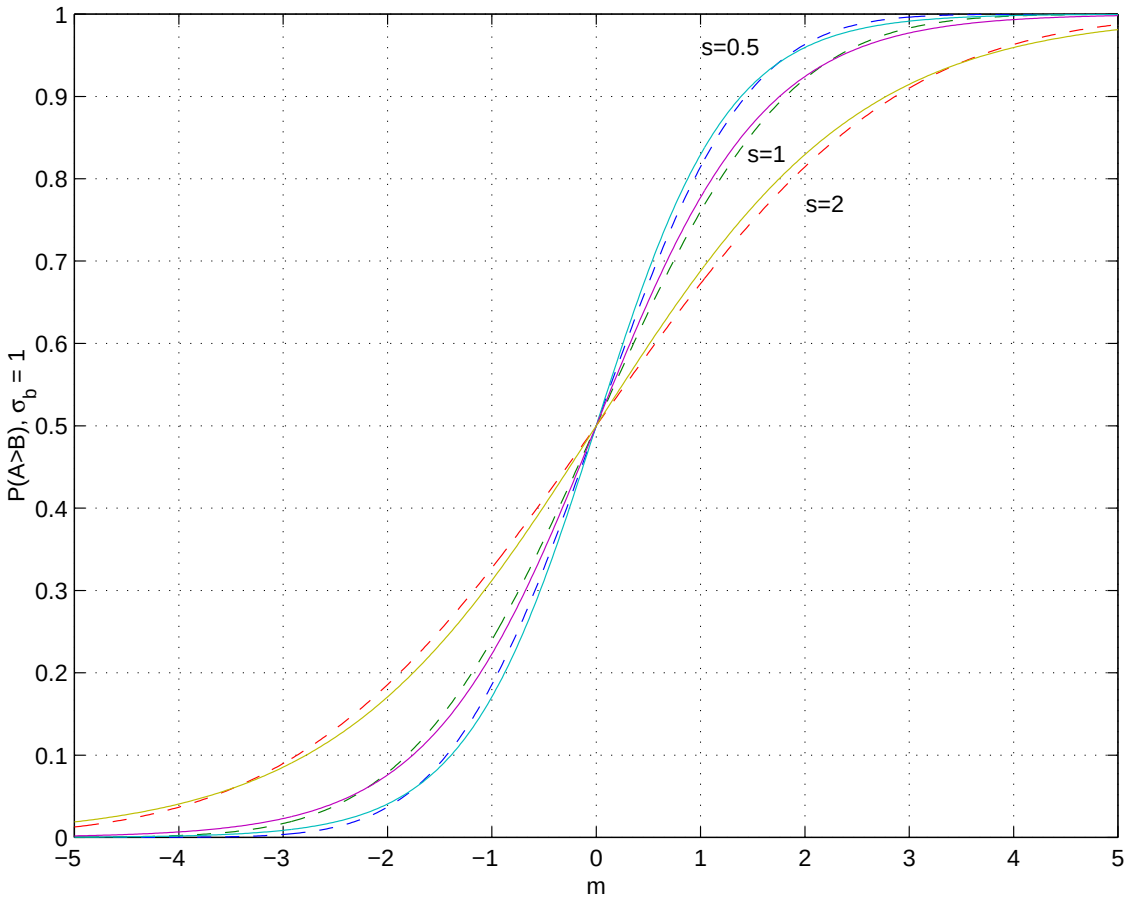

Figure 3.4: Approximation of $P(A > B)$ against m

Therefore if $A = 0$, $B = 5$ and $\sigma_n = 1$, $P(A > B) = 0$ as expected.

Unfortunately, the error function $\text{erf}(x)$ is not easy to calculate quickly. It can be approximated using Chebyshev fitting [11, Section 6.2] but even this is not very quick. Recognising that the curves in figure 3.3 are sigmoidal in shape, other standard sigmoidal curves have been fitted to give a good approximation to the curve, but allow the probability to be calculated quickly. Figure 3.4 shows the curve approximation (dashed curve is (3.15)) and (3.17) & (3.18) show the approximations. The results of the two different approximations are so similar to each other, they appear as a single line on the graph.

$$P(A > B) \approx \frac{1}{2}\left(1 + \tanh \frac{m}{0.8\sqrt{2 + 2s^2}}\right) \tag{3.17}$$

$$P(A > B) \approx \frac{1}{1 + e^{-\frac{2.5\, m}{\sqrt{2+2s^2}}}} \tag{3.18}$$

In the ranking process detailed in section 4, the calculation of the probabilities is $O(n^2)$ with respect to the number of objectives ranked n. Using (3.19) & (3.20) below, if $s = 1$ we can split the calculation of m allowing $\tanh(x/(1.6\sigma_n))$ or $e^{-1.25\, x/\sigma_n}$ to be calculated for each of the objectives to be ranked. This process is $O(n)$. The $O(n^2)$ comparison process where the probability is calculated only needs to perform very fast multiply and add instructions, drastically reducing the processing time for large n. Unfortunately, (3.19) & (3.20) assume that $\tanh(x)$ and e^x can be calculated to infinite precision. In practice, if $|x/(1.6\sigma_n)| > 17$ (for tanh) or $|1.25x/\sigma_n| > 15$ (for exponential) then (3.17) & (3.18) must be used. The exact limits though are dependent on the machine precision.

$$\tanh(a + b) = \frac{\tanh(a) - \tanh(b)}{1 - \tanh(a)\tanh(b)} \tag{3.19}$$

$$e^{(a-b)} = e^a / e^b \tag{3.20}$$

3.4. Multi-objective fitness functions and noisy domination

With multiple objectives, we no longer have only two possible outcomes from comparing two objectives A and B. We now have the possibility of the two objectives being non-dominated. We therefore can have $P(A < B)$, $P(A > B)$, and $P(A \equiv B)$. Figure 3.5 shows the effect graphically, with the point A in the centre of the figure ($[0.5, 0.5]$) representing one sample of the fitness. The shaded regions correspond to regions in which there is information to drive the evolutionary process. In the non-dominated regions, we have no way of deciding between the functions.

If we have two, two objective, independent fitness measurements with corresponding objective values A_1, A_2, B_1, and B_2, the probabilities $P(A < B)$, $P(A > B)$, and $P(A \equiv B)$ are simply

$$\begin{aligned} P(A > B) &= P(A_1 > B_1) \cdot P(A_2 > B_2) \\ P(A < B) &= P(A_1 < B_1) \cdot P(A_2 < B_2) \\ P(A \equiv B) &= 1 - P(A < B) - P(A > B) \end{aligned} \tag{3.21}$$

Therefore in general with k objectives

$$\begin{aligned} P(A > B) &= \prod_{j=1}^{k} P(A_j > B_j) \\ P(A < B) &= \prod_{j=1}^{k} P(A_j < B_j) \\ &= \prod_{j=1}^{k} (1 - P(A_j > B_j)) \\ P(A \equiv B) &= 1 - P(A < B) - P(A > B) \end{aligned} \tag{3.22}$$

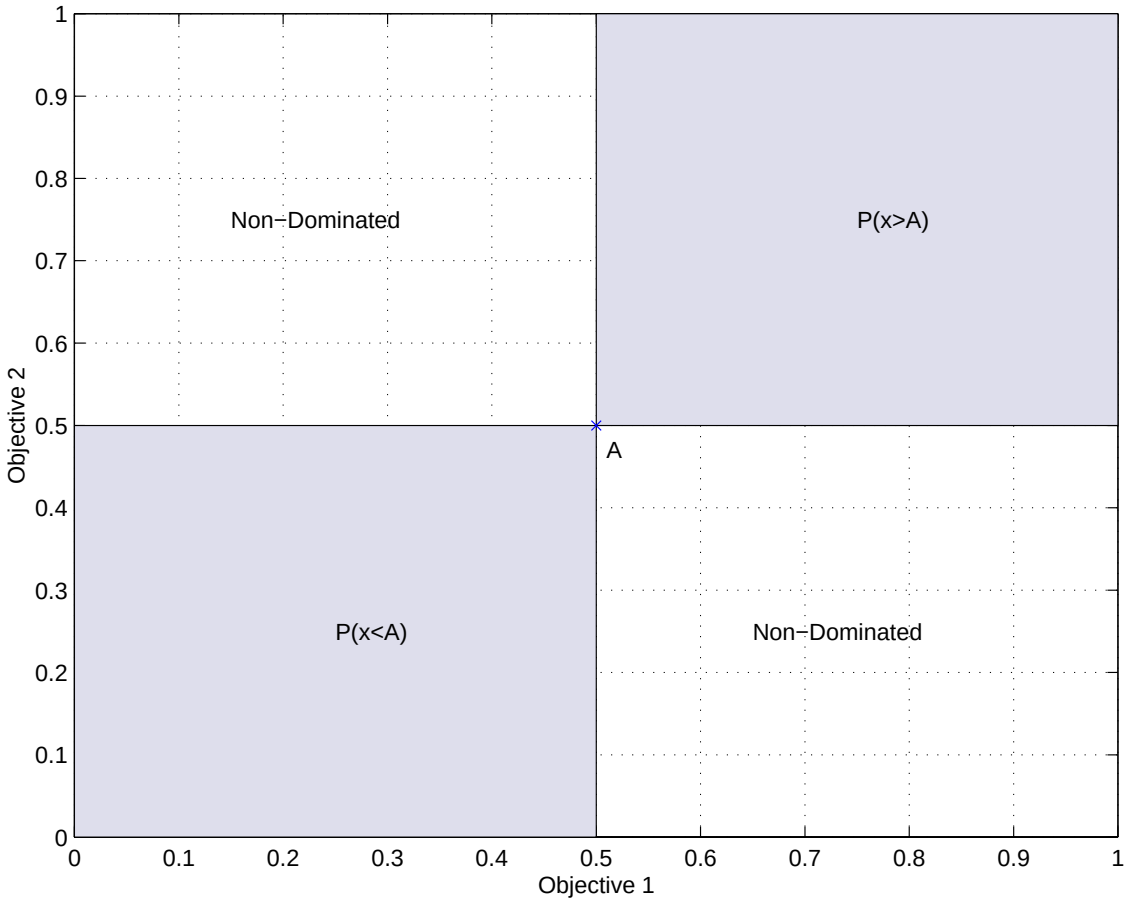

Figure 3.5: Noise free non-domination map

If all our fitness values lie within a bounded region, which is often the case, we may calculate the probability of being in a situation where one value dominates the other. Equation 3.23 shows the integral of the area (with reference to figure 3.5), this is relatively simple to evaluate by hand, and as expected gives a probability of $P = 0.5$ for a two dimensional objective.

$$P(x < A) + P(x > A) = \int_0^1 \int_0^1 x_1 x_2 + (1 - x_1)(1 - x_2) \; dx_1 dx_2 = \frac{1}{2} \tag{3.23}$$

Repeating the analysis for a 3 dimensional problem reveals that the probability that a point x will dominate, or be dominated by another point in the bounded area is $P = 0.25$. In general for k objectives

$$P(\text{domination}) = \frac{1}{2^{(k-1)}} \tag{3.24}$$

If we have a noisy function, the equivalent domination map of figure 3.5, in relation to point $[0.5, 0.5]$, is shown in figure 3.6. Here the change in probability is not distinct and we have to display a surface ($\sigma_n = 1/8$, $s = 1$).

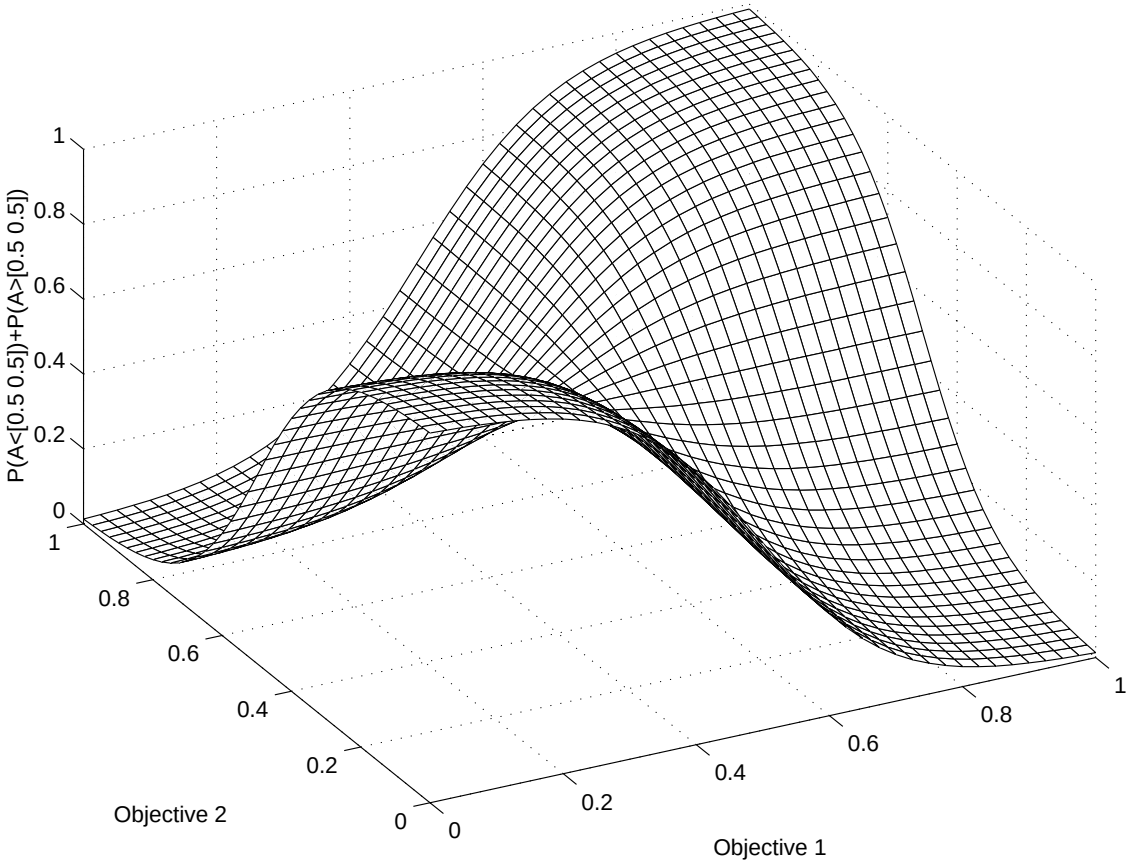

Figure 3.6: Uncertain non-domination map

We can however still approximate the mean probability of getting a dominated solution as shown in (3.25) (for 2D case, integrated using Mathematica). Integrating the three dimensional case is not as straight forward so it was verified numerically that (3.24) still holds, where h_1 and h_2 are the factors relating to the approximation constant $1.6\sigma_1$ and $1.6\sigma_2$ respectively and $[i, j]$ and $[x_1, x_2]$ are the coordinates of the two points of comparison.

$$\begin{aligned} P &= \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{(1+tanh((x_1-i)/h_1))(1+tanh((x_2-j)/h_2))}{4} \\ &\quad + \left(1-\frac{1+tanh((x_1-i)/h_1)}{2}\right)\cdot\left(1-\frac{1+tanh((x_2-j)/h_2)}{2}\right) didjdx_1dx_2 \\ &= \frac{1}{2} \end{aligned} \tag{3.25}$$

This shows that the regions where we can distinguish between good and bad solutions shrinks rapidly as we increase the number of objectives in the optimisation process. The larger the number of objectives, the larger the non-dominated region and therefore more solutions will be required to cover the Pareto surface.

4. Probabilistic Ranking and Selection

4.1. Introduction

Ranking is often employed to prevent a superior solution dominating the early populations in the evolutionary process. The conventional ranking process, however, does not take the uncertainty in the measured fitness values into account. The following sections provide a fresh view of the ranking process and develop theory for multi-objective ranking of uncertain fitness measurements.

4.2. Single objective ranking

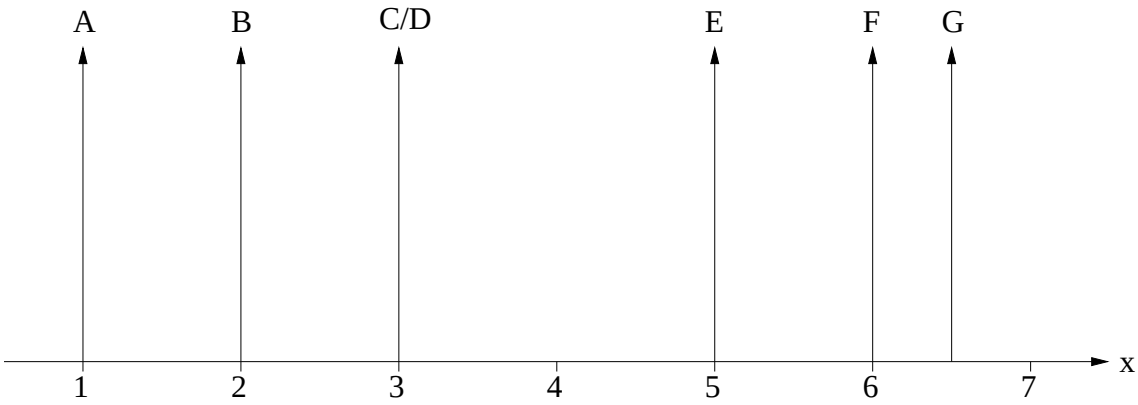

Figure 4.1: Fitness values to be ranked

Figure 4.1 shows seven fitness values to be ranked. If we are minimising, the best fitness value is the lowest. In the case shown, value A will get rank 0, and value G will be rank 6. Values C and D are equal and therefore should be assigned the same rank. We can assign rank values as shown in table 4.1

If we did not have a tie between C & D, we could use the linear selection equation, (4.1) to calculate probabilities of selection, based on the ranked fitness, where n is the number of fitness values and R_i is the rank of individual i. The sum of the rank values on the denominator will sum to $n(n-1)/2 = 21$ which is the sum of the arithmetic series zero to six, therefore the best individual will get a probability of selection of $2/n$ and the worst a probability of zero.

$$P(\text{select}_i) = \frac{(n-1) - R_i}{\sum_{j=1}^{n} R_j} = \frac{2((n-1) - R_i)}{n(n-1)} \tag{4.1}$$

Table 4.1: Ranks of example fitness values

Value	Rank
A	0
B	1
C	2
D	2
E	4
F	5
G	6

If we use the rank values in table 4.1 with both the tied fitness values being given the best 'untied' rank, we find that the sum of the ranks is no longer consistent, and in this case, $\sum_{j=1}^{n} R_j = 20$. Alternatively, as C & D are tied, it may be better to penalise them both a little and therefore take an average of the rank positions they could have shared, i.e., give them both a rank of 2.5. This would return the overall sum to be 21 and would be consistent, no matter how many fitness values share a rank. This is method most used for ranking a vector of data.

We can view the ranking process as counting the number of fitnesses that dominate the fitness of interest [8]. If a fitness equal to the current one is encountered, then it is half dominating, and half dominated by the current fitness. Therefore we can create the rank position numbers by this simple counting process. For example, E is dominated by *A, B, C & D* and therefore has a rank of 4. Value C is dominated by *A & B* but is tied with D and so gets a rank of 2.5.

Alternatively, we could consider the dominating / not dominating decision as being the *probability* that each fitness value dominates the value of interest. For example, if we consider fitness C, the probability that A dominates C is one. The probability that G dominates C is zero. The probability that D dominates C, from (3.10) with $m = 0$, is $P = 0.5$. Thus we can represent the rank position as the sum of probabilities of domination as shown in (4.2), where $P(F_j > F_i)$ is the probability that fitness value j dominates fitness value i.

$$R_i = \sum_{j=1}^{n} P(F_j > F_i) \Big|_{i \neq j} \tag{4.2}$$

In (4.2), we have to be sure not to compare fitness F_i with itself. If we did, we would get an extra probability of 0.5 added to the sum. We can therefore include F_i in the sum, but subtract the effect of comparing the fitness with itself. This is shown in (4.3).

$$R_i = \sum_{j=1}^{n} P(F_j > F_i) - 0.5 \tag{4.3}$$

As (4.3) is based on probability, if the fitness values are uncertain, we can use (3.10) or the approximation (3.17) & (3.18) to calculate the probability of domination. For example, if fitness values A to G have a standard deviation of $\sigma_n = 1$, the rank positions (using (3.17)) compared to the no noise case are shown in table 4.2.

Table 4.2: Ranks and probabilities with uncertainty of $\sigma_n = 0$ and $\sigma_n = 1$

Value	Rank ($\sigma_n = 0$)	Rank ($\sigma_n = 1$)
A	0	0.38
B	1	1.27
C	2.5	2.31
D	2.5	2.31
E	4	4.17
F	5	5.07
G	6	5.49

With $\sigma_n = 0$, we have conventional ranking and the probabilities will range from $2/n$ to zero. If $\sigma_n = \infty$, all of the fitness values will be assigned the same rank, and will have a probability of selection of $1/n$. Thus the standard deviation of the uncertainty has a similar effect to selective pressure in conventional selection processes [12].

4.3. Multi-objective ranking

With multiple objectives, we now have three possible outcomes from comparing the two fitness values: A dominates B, A is dominated by B, and A and B are non-dominated. If we apply the single objective ranking equation, we find that the total of the rank positions is no longer $n(n-1)/2$ as we now have to account for the non-domination. If we have no noise, for two fitness values where A dominates B, $P(A > B) = 1$, $P(A < B) = 0$, and $P(A \equiv B) = 0$ Therefore when we sum the probabilities of domination, the contribution from this pair will be 1. If the fitness values are non-dominated, the corresponding probabilities are $P(A > B) = 0$, $P(A < B) = 0$, and $P(A \equiv B) = 1$. We have now lost the value 1 from the probability of domination calculations, therefore reducing the sum of ranks total. This state will be the same when we compare A to B and also when we compare B to A, therefore if we sum the total probability of non-domination, this will give us twice what was lost from the probability of domination calculations.

If we consider the ranking case for a single dimension, if A and B are identical, we cannot choose between them and so add in 0.5 to the sum. With non-domination, we also have the situation where we cannot choose between objectives and should therefore add 0.5 to the sum as required. In the case of uncertain measurements, we can multiply the value of 0.5 by the probability of non-domination, and still subtract off 0.5 to allow for comparing the individual with itself, thereby maintaining the sum of the rank positions as $n(n-1)/2$. Thus we can add the non-domination term into (4.3). The rank calculation for multi-objective ranking is shown in (4.4), where n is the number of fitness measurements.

$$R_i = \sum_{j=1}^{n} P(F_j > F_i) + \frac{1}{2} \sum_{j=1}^{n} P(F_j \equiv F_i) - 0.5 \tag{4.4}$$

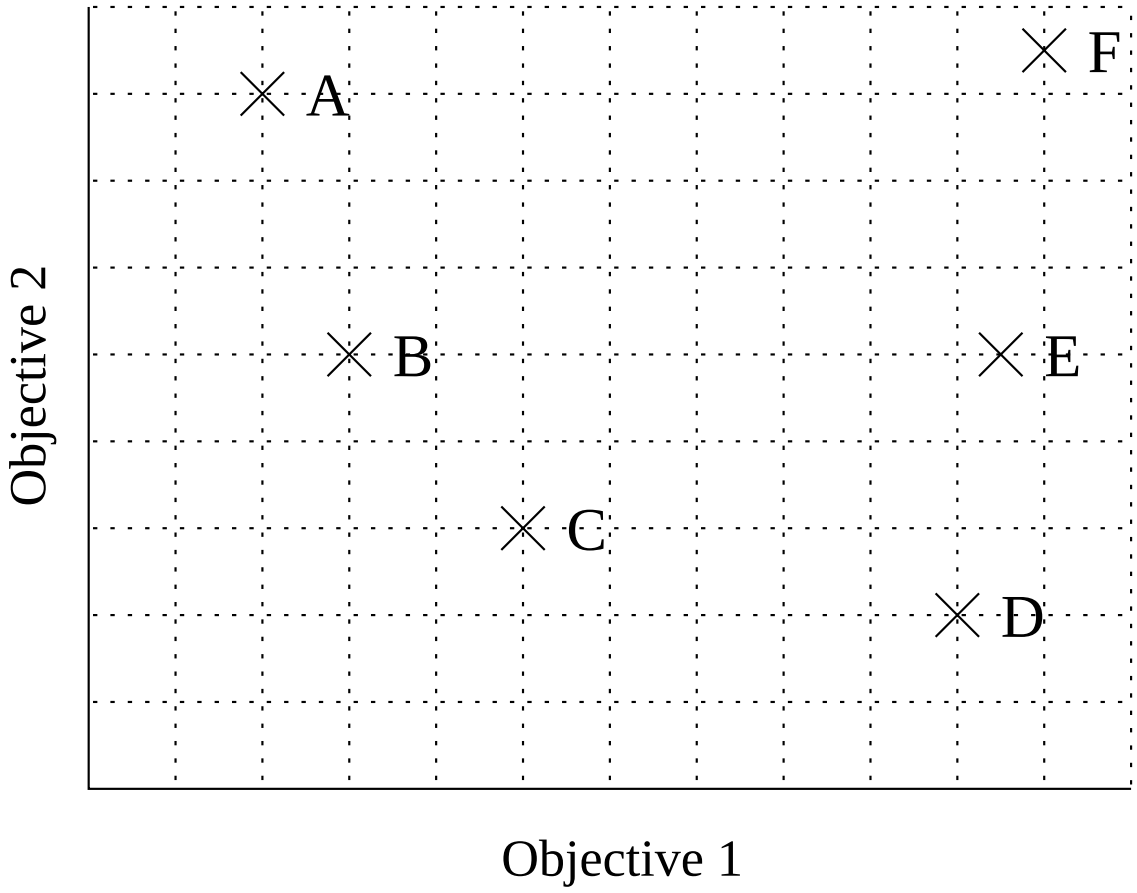

Figure 4.2: Multiple fitness values to be ranked

Table 4.3: Ranks, Probabilities and Non-domination counts with uncertainty of $\sigma_n = 0$ and $\sigma_n = 1$

Value	R ($\sigma_n = 0$)	R ($\sigma_n = 1$)
A	2	2.27
B	1.75	1.65
C	1.5	1.42
D	1.5	1.92
E	3.25	3.22
F	5.0	4.53

This *probabilistic ranking* equation allows chromosomes to be selected based on uncertain multi-objective fitness measurements. For the objectives shown in figure 4.2, we can calculate the rankings in order to minimise the fitness values. Table 4.3 shows ranks (R) for 1 standard deviation noise, and no noise.

In the example, we see that A is non-dominated with B, C, D, & E and therefore gets a rank of 2. Fitness B is non-dominated with A, C, & D but shares an objective value with E, thus being half dominating and half non-dominated with E, the rank of B is 1.5 from the three non-dominated points and 0.25 from E, giving a total of 1.75. We also see that each of the columns of table 4.3 sums to 15 ($= n(n-1)/2$) as expected. The ranking process is $O(n^2)$, as are many of the other ranking methods [8, 9].

In the general noisy scenario, we see that the proximity of other fitness values, even if only close on one objective, can influence how the rank is assigned. Measurements such as C which are relatively well spaced out on all objectives are ranked more highly than other fitness values that are uncertain. With no noise, the basic ranking by just counting how many points dominate each fitness measurement described by Fonseca and Flemming [8] is very similar, but does not allow for the non-dominated cases. The sum of the rank values will not be consistent if non-dominated solutions are present, causing a bias towards non-dominated solutions over other solutions. The ranking used by Srinivnas and Deb [9] is based on 'layers' of non-dominated solutions and has no consistency with regards to how many layers, or ranks, are produced, therefore making calculating selection probabilities awkward.

It is interesting to note that if we require an objective to be maximised, setting σ_n negative will cause the probabilities to be calculated for maximisation, setting σ_n negative has the same effect as negating the fitness values (the conventional way of converting from minimisation to maximisation). Therefore both minimisation and maximisation objectives may be handled easily by just setting the sign of the corresponding value of σ_n appropriately.

5. Designer Preference & Constraints

5.1. Introduction

For optimising real systems, allowing the designer to have interactive control over the evolutionary process is paramount. The designer will often want to confine the region of evolution in order to investigate an interesting region in detail and may also want to mark some objectives as being a higher priority. A simple approach to defining regions of interest is to allow the designer to specify a limit on each objective. Solutions with a corresponding value worse than the limit would be penalised. Problem parameters are also often constrained. With a single objective it is straight forward to add a *penalty function* that adds to the objective value if the parameters are out of bounds. In a multiple objective case, as the relative scalings between the objectives is not known, applying a penalty is less straight forward. The designer may also want to focus attention on certain solutions, in the case of a noisy function, the objective values will be time dependent. For a discussion of other Pareto preference techniques see [13].

5.2. Parameter Constraints and Objective Limits

The parameters defining potential problem solutions often have regions within which they must be constrained. Deb [14] provides a discussion on the different methods used to apply constraints in EA's. If possible, the constraints should be handled by the genotypic / phenotypic description to help prevent the production of fatal solutions. To describe the constraint, we can define a function $G(\chi_i)$ as being unity if the chromosome χ_i is wholly within a constrained region (assuming no noise or uncertainty) and zero if a constraint is violated. If the individual constraints are formulated as $g_j(\chi_i) \leq 0$, we can define an error metric as $\epsilon_{ji} = g_j(\chi_i)$ and use (3.16) with the constraint noise standard deviation σ_c to give

$$G_j(\chi_i) \;=\; \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\,(\frac{-\epsilon_{ji}}{2\sigma_c}) \;. \tag{5.1}$$

Multiple constraints may be combined either by forming the product (5.2) of the u constraints, or by taking the geometrical mean (5.3). It is prudent not to constrain the problem too highly in the early generations if possible to allow the evolutionary process to work with the greatest number of feasible solutions possible.

$$G \;=\; \prod_{j=1}^{u} G_j \tag{5.2}$$

$$G \;=\; \sqrt[u]{G_1 G_2 \cdots G_u} \tag{5.3}$$

Table 5.1: Required probabilities when constrained

$C(A)$	$C(B)$	$P_c(A > B)$	$P_c(A < B)$	$P_c(A \equiv B)$
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	$P(A > B)$	$P(A < B)$	$P(A \equiv B)$

Limits on the objectives may be applied by treating the k limits as forming a point Z in the k dimensional objective space. The probability of each individual dominating this point can be calculated ($P(F_i > Z)$) and this probability can then be multiplied with the parameter constraint value $G(\chi_i)$ to give $C(\chi_i)$.

We can apply the constraint easily to the previously developed ranking process by applying the logic of: if both chromosomes meet all constraints, the dominance probabilities are unchanged; If both violate the constraints, they are classed as being non-dominated; if one violates constraints and the other does not, the violating chromosome is classed as being dominated. Table 5.1 gives an alternative view of the logic.

Equation 5.4 shows the logic expressed in a form suitable for noisy systems.

$$\begin{aligned} P_c(A > B) &= P(A > B)C(A)C(B) + C(A)(1 - C(B)) \\ P_c(A < B) &= P(A < B)C(A)C(B) + (1 - C(A))C(B) \\ P_c(A \equiv B) &= 1 - P_c(A > B) - P_c(A < B) \end{aligned} \tag{5.4}$$

The probabilities after the constraints have been applied may be used directly in the ranking calculation shown in (4.4).

With this method of applying the constraints, the sum of the ranks is preserved, compared to an alternative technique where the reversed rank of individual i is multiplied by $C(\chi_i)$ to give $C(\chi_x)((n-1) - R_i)$, i.e., the constraints are applied to the ranked values. Thus a solution with good objective values that violates constraints will get a low probability of selection. With this alternative approach the sum of the ranks may not be $n(n-1)/2$ anymore. With either technique, by modifying the rank position, objective scaling differences are no longer a problem. This approach allows the designer to specify limits on the evolution interactively as the population evolves. Sharing may also be applied and the niche count used to reduce $C(\chi_i)$ accordingly, i.e. $C_s(\chi_i) = C(\chi_i)/s$, where s is the niche count for the individual. This is detailed in [15].

5.3. Preferred solutions and Priority

As the population evolves, the designer may see solutions that may be viable. If the individual in question is *marked* as a preferred solution, a form of elitism may be used to guarantee that all marked solutions survive and are copied into the next generation. Thus solutions that may have died off can be preserved, passing on a proportion of their genetic material. The marking process has no effect on the rank or probability of selection and if poor individuals are marked early in the

evolution, it is up to the designer to remove the mark as appropriate. In a noisy environment, solutions may move position and once marked, should be re-evaluated every generation, allowing the deviations of the objective values due to noise to be observed.

The noise standard deviation, σ_n, may be set differently for each objective. Any objective with $\sigma_n = \infty$ will have an effective selection pressure of zero in the ranking process. This will prevent a decision being made regarding the objective and will make the rankings more non-dominated. If $\sigma_n = 0$, any difference in the two objectives being compared will lead to a domination decision. In the ranking process therefore, objectives with lower σ_n will have a higher effective selective pressure and therefore σ_n could be used to express objective priority.

Unfortunately, the use of σ_n to control selective pressure and priority is not convenient as the values of σ_n needed are dependent on the scaling of each objective. A better way is to include selective pressure and priority explicitly within the ranking process. Equation 5.5 shows alternative methods of calculating the domination probabilities, with ν_j being the selective pressure for objective j and lying in the range [0,1] with 1 being maximum selective pressure and zero being no selection. The selective pressure of each objective can be controlled separately, when they are all set to zero, all the solutions have the same probability of selection. If one objective has a selective pressure of zero, it still has an effect on the ranking process, forcing the ranks to be more non-dominated.

$$\begin{aligned}
P_s(A > B) &= \prod_{j=1}^{k} \left(P_c(A_j > B_j)\nu_j + \frac{1}{2}(1 - \nu_j) \right) \\
P_s(A < B) &= \prod_{j=1}^{k} \left(P_c(A_j < B_j)\nu_j + \frac{1}{2}(1 - \nu_j) \right) \\
&= \prod_{j=1}^{k} \left((1 - P_c(A_j > B_j))\nu_j + \frac{1}{2}(1 - \nu_j) \right) \\
P_s(A \equiv B) &= 1 - P_s(A < B) - P_s(A > B)
\end{aligned} \tag{5.5}$$

For priority, we can use a similar technique to selective pressure, but here, if we have one objective with a priority of zero, it should play no part whatsoever in the ranking process. Equation 5.6 shows equations to allow the priorities ρ_j for each objective j to be integrated into the ranking process. The priorities ρ_j lie in the interval [0,1] with 1 being the highest priority and zero making the objective play no part in the ranking. In (5.6), the factor h will be zero if all of the priorities are zero. This will force all solutions to be non-dominated and so have an equal probability of selection.

$$
\begin{aligned}
h &= \left(1 - \prod_{j=1}^{k}(1-\rho_j)\right) \\
P_p(A > B) &= h \prod_{j=1}^{k} (P_c(A_j > B_j)\rho_j + (1-\rho_j)) \\
P_p(A < B) &= h \prod_{j=1}^{k} (P_c(A_j < B_j)\rho_j + (1-\rho_j))
\end{aligned} \tag{5.6}
$$

Equation 5.7 shows the final form with both selective pressure and priority taken into account. As both processes are performed as part of the ranking process, the consistency in the sum of the ranks is maintained.

$$
\begin{aligned}
P_{ps}(A > B) &= h \prod_{j=1}^{k} \left(\left(P_c(A_j > B_j)\nu_j + \frac{1}{2}(1-\nu_j)\right)\rho_j + (1-\rho_j)\right) \\
P_{ps}(A < B) &= h \prod_{j=1}^{k} \left(\left(P_c(A_j < B_j)\nu_j + \frac{1}{2}(1-\nu_j)\right)\rho_j + (1-\rho_j)\right) \\
P_{ps}(A \equiv B) &= 1 - P_{ps}(A < B) - P_{ps}(A > B)
\end{aligned} \tag{5.7}
$$

As the priority calculation is performed as part of the Pareto ranking process, the consistency in the sum of the ranks is maintained. Equation 5.6 can be used in place of (3.22) in calculating the domination probabilities. This elegant integrated approach to priority and constraint gives full control to the designer.

6. Fitness Sharing and Restrictive Breeding

6.1. Introduction

Due to the very nature of evolutionary algorithms, after many generations, the imbalances in the selection process leads to genetic drift and clusters of very similar individuals forming. Sharing allows multiple stable populations to form spread across the objective region. MOGA [8] and NSGA [9] both use sharing to help spread individuals across the Pareto front. In NSGA and some versions of MOGA, individuals are shared within each rank only.

In the case of uncertain objective measurements, the individual rank layers are no longer apparent and therefore the sharing needs to be applied irrespective of rank, therefore based on the whole population. Solutions from opposite sides of the Pareto front, once mated, can produce fatal solutions, therefore some restrictive breeding process can be beneficial.

6.2. Fitness sharing

The aim of sharing is to spread out the objectives to cover the non-dominated front evenly. Ideally we should apply the sharing to the objectives, but often the objectives are subject to unknown scaling parameters. The most often used scaling methods rely on measuring the Euclidean distance between two solutions. The scaling may have dramatic effects on the sharing process with parameters having a large scaling value having a disproportionate effect on the distance measured. Often the sharing is applied to the chromosomes rather than the objectives as the scaling parameters are easier to determine.

In this report, the sharing will be applied to the objective values rather than the chromosome in order to get an even spread of solutions in the objective space. To help remove the effect of scaling, a different share distance is set interactively by the designer for each objective. A share value is calculated for each pair of individuals, i and j, and each objective. The share value is unity if the individuals are identical and have zero uncertainty. Therefore with k objectives, there will be k share values. These are combined into a single value by taking the geometrical mean as shown in (6.1).

$$v(i,j) = \sqrt[k]{v_1 v_2 \dots v_k} \tag{6.1}$$

For one objective, we can use (3.11) to generate the distribution of the difference between a pair of individuals on one objective. A Gaussian shaped share function can be used to quantify the share value. In other methods such as MOGA and NSGA

linear sharing functions have been used but in the case of uncertain measurements, a Gaussian shape is simpler mathematically. The share function is defined as having a maximum of unity, regardless of the spread, i.e.

$$F(\text{Share}) = e^{\frac{-x^2}{2\sigma_s}} \tag{6.2}$$

Figure 6.1 shows the result of sharing measurements $A = 1$ and $B = 2$ with both measurements having an uncertainty of $\sigma_n = 0.1$ with a share function with a spread of $\sigma_s = 1$. The figure shows the spread of the functions normalised with respect to the error difference function.

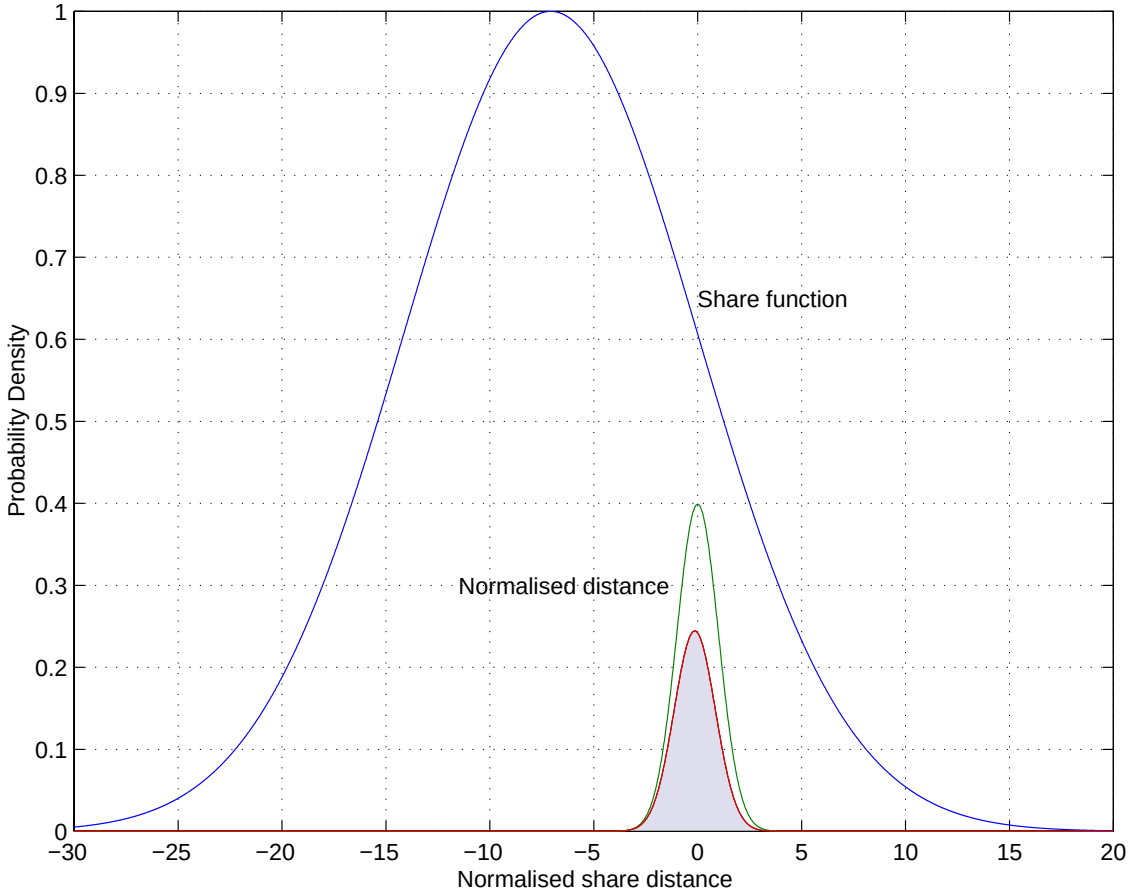

Figure 6.1: Sharing uncertain fitness values

The share value v is defined as the area under the product of the two curves. After spread normalisation, the sharing function retains a peak value of unity.

$$v = \int_{-\infty}^{\infty} e^{-\frac{x^2}{2\sigma_s^2}} \frac{1}{\sqrt{2\pi}\sigma_d} e^{-\frac{(x-\mu_d)^2}{2\sigma_d^2}} \tag{6.3}$$

Where $\sigma_d^2 = \sigma_{nA}^2 + \sigma_{nB}^2$ and $\mu_d = A - B$. We can normalise the Gaussian describing the difference between the two objective values to be zero mean and unity variance

to give

$$v = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{(x+\frac{\mu_d}{\sigma_d})^2 \sigma_d^2}{2\sigma_s^2}} e^{-\frac{x^2}{2}} \tag{6.4}$$

$$v = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{(x+d)^2}{2h^2}} e^{-\frac{x^2}{2}} \tag{6.5}$$

$$= \frac{h_d}{\sqrt{1+h_d}} e^{-\frac{d^2}{2(1+h_d^2)}} \tag{6.6}$$

Where $d = \mu_d/\sigma_d$ and $h_d = \sigma_s/\sigma_d$. If the noise spread of A and B is the same, $\sigma_d = \sqrt{2}\sigma_n$ therefore for this common case, we can use the previous definition of $m = (A - B)/\sigma_n$ and define $h_n = \sigma_s/\sigma_n$ to give

$$v = \frac{h_n}{\sqrt{2+h_n}} e^{-\frac{m^2}{2(2+h_n^2)}} \tag{6.7}$$

When the share values, v, for each objective are combined using the geometrical mean, computational savings can be made by noting that the equation for v is an exponential form and therefore the multiplication of the individual share values is equivalent to summing the exponents. The n^{th} root may be taken by dividing the sum of the exponents by n.

The effects of sharing are applied by sharing each solution with every other solution, except for itself, and then summing the share values to give a share count. The share count has unity added to it to allow for the individual sharing perfectly with itself. As in the calculation of rankings, if we sum over all individuals, including the one of interest, we need to subtract off the share value that occurs at $m = 0$ to account for self sharing. We can therefore define the share count for individual i as

$$s_i = \sum_{j=1}^{n} v(i,j) - \frac{h_n}{\sqrt{2+h_n^2}} + 1 \tag{6.8}$$

We can then use the share count in (6.8) to derate the constraint values $C(\chi_i)$ to give a shared constraint of $C_s(\chi_i)$ as shown in (6.9). The shared constraint values may be used in (5.4) to produce the constrained and shared rank values.

$$C_s(\chi_i) = \frac{C(\chi_i)}{s_i} \tag{6.9}$$

In low noise applications, as the sharing is applied to the whole population, solutions which are dominated by a point of interest will also have an effect on that points share count. This can be reduced by utilising the probability of non-domination to influence the effect of individual comparisons. Increments only need to be made to the share count by points that are non-dominated. A modification to the share count calculation that accounts for non-domination is shown in (6.10).

$$s_i = \sum_{j=1}^{n} P(i \equiv j) v(i,j) - \left(1 - \frac{1}{2^{k-1}}\right) \frac{h_n}{\sqrt{2+h_n^2}} + 1 \tag{6.10}$$

6.3. Restrictive breeding

It has been established [8] that with certain problems, the objective surface is such that if the genetic material from distant regions of the Pareto surface are mated, often fatal solutions are created that perform far worse than either of the parents. The share distance v may be used as a basis for reproduction, with preference being given to mates with a higher v, therefore similar objective values. A similar measure may be applied to the chromosomes to identify solutions that are close in the genotypic domain.

6.4. Steady state evolutionary algorithms

With steady state EA's, only a few individuals are evaluated at a time and then re-inserted into the population. With a large population, it is expensive to re-evaluate the whole of the Pareto ranking calculations each time. Using the ranking methods derived previously, if the probability of domination data is stored in an array, to remove an individual, only the appropriate row and column needs to be deleted from the array. To insert an individual, an extra row and column must be added, comparing the new individual to every other. The sums of the rows (or columns as appropriate) can then be calculated to yield the rank positions. This approach only requires the final sum to be re-evaluated for the whole array each time.

7. Probabilistic Tournament Selection

7.1. Introduction

Evolutionary techniques such as Differential Evolution [16] use tournament selection to drive the evolutionary process. An offspring is created and compared to a parent in the population. If the offspring is better, it replaces the parent, otherwise it is discarded. With multiple objectives and no noise, comparing two individuals will give one of three results: $A < B$, $A > B$ or non-domination ($A \equiv B$). If the individuals are non-dominated, they are equivalent and therefore there is no evolutionary drive to select one over the other. It is not clear which individual should be retained or how the non-domination should be managed. In a noisy scenario, if the individuals are well spaced on each objective, then they will be truly non-dominated. However, if they are 'close' on at least one objective, it may be uncertain whether the individuals are truly non-dominated, or if one may sometimes dominate the other. This allows us to quantify how much better one close, but non-dominated, individual is to another. Figure 7.1 shows two individuals that are close on one objective but well separated on the other.

7.2. Tournament selection algorithm

For a single objective we only have two possible domination outcomes, therefore $P(A < B) + (A > B) = 1$. We can also apply the constraint details developed in section 5.2. To select an individual:

1. Generate probability that A dominates B, $P(A > B)$, and probability that B dominates A, $P(A < B)$ using (3.17) or (3.18). Also generate constraint parameters $C(A)$ and $C(B)$.

2. Generate random number, R, and accept A if $R < \frac{P(A>B)C(A)}{P(A>B)C(A)+P(A<B)C(B)}$, else take B.

For multiple objectives, we also have the case of non-domination, therefore $P(A > B) + P(A < B) + P(A \equiv B) = 1$, and so $P(A \equiv B) = 1 - P(A < B) - (A > B)$. To select an individual we can use:

1. Generate probability that A dominates B, $P(A > B)$, and probability that B dominates A, $P(A < B)$ using (3.17) or (3.18) and (3.22). Also generate constraint parameters $C(A)$ and $C(B)$.

2. Generate random number, R, and accept A if $R < \frac{P(A>B)C(A)}{P(A>B)C(A)+P(A<B)C(B)}$, else take B.

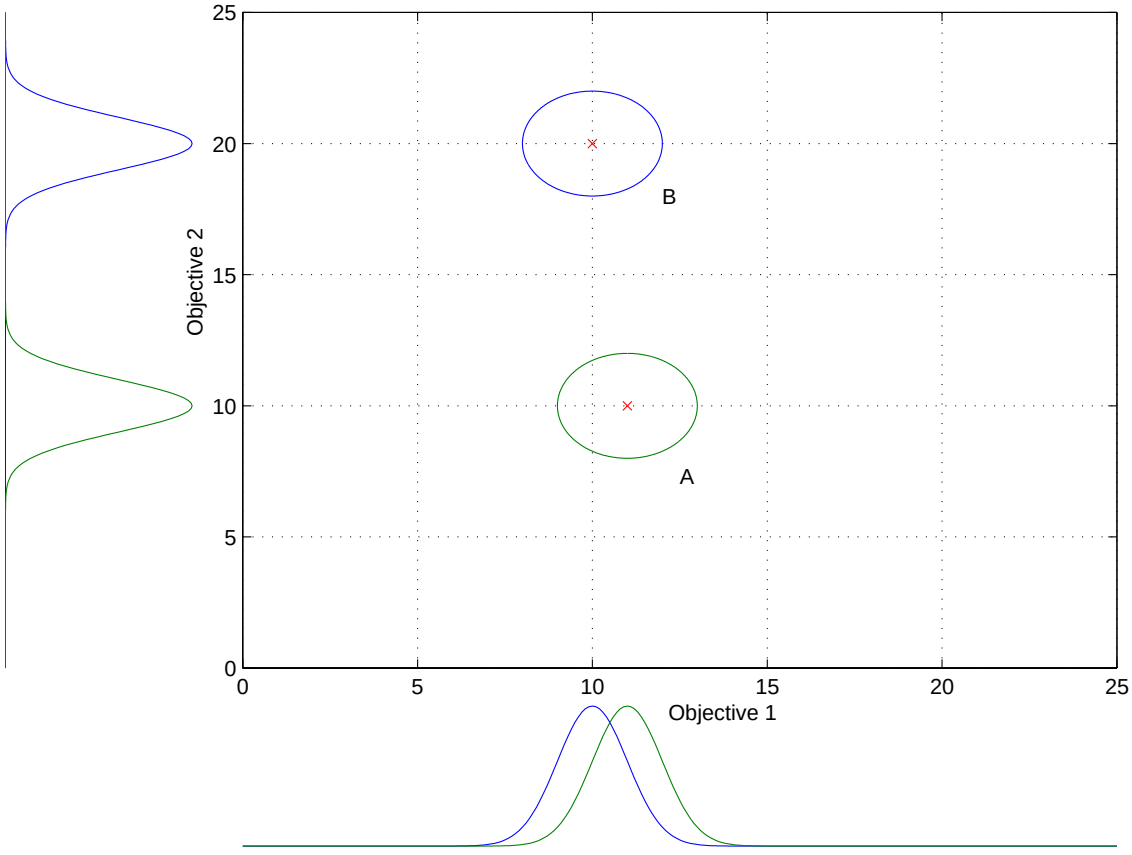

Figure 7.1: Tournament selection in uncertain case

3. Generate a second random number R_2 and classify solutions as non-dominated if $R_2 < (1 - P(A > B) - P(A < B))$. If solutions are non-dominated, add them both to a 'history reservoir'. At the end of the generation, if the history reservoir is too full, rank all the solutions using (4.4) and select the best to give the required reservoir size. The constraints for the non-dominated solutions are then handled in the ranking process by (4.4).

Care needs to be taken to prevent the denominator of the selection equations in step 2 of each algorithm from becoming zero. If the denominator is zero, the probability of selecting A or B should be 0.5.

This approach allows decisions to be made with some account for noise or uncertainty. The algorithm suggested for multiple objectives is only one possible way of maintaining the non-dominated solutions. In this algorithm, the history reservoir is not used in the selection of individuals for the population and is only used to maintain a record of the non-dominated surface found. Elements such as restrictive breeding or re-combination with members of the history population may help with certain problems and the algorithms can be adjusted accordingly.

8. Experiment Results

8.1. Introduction

Noise and uncertainty can be split into two broad categories relating to noise that occurs within the process (Type A) and measurement noise (Type B):

1. **Type A Noise:** Noise is applied to the chromosome *before* the objective function is calculated, i.e. $\mathcal{O} = F(\chi + N)$.

2. **Type B Noise:** Noise is applied to the objective function *after* calculation, i.e. $\mathcal{O} = F(\chi) + N$.

Both types of noise are of interest and often the observed noise will be a combination of type A and B. Experiments have been devised to assess each of the main sections of the work developed in this paper in the presence of noise: ranking, tournament selection, sharing, and constraints.

8.2. Test Objectives

A range of test objectives were developed for the trials. Table 8.1 lists the objective functions used, with either type A or type B noise as appropriate.

8.3. Results

8.3.1. Rank Positions

Trials have been performed to assess how the noise effects the assigned rank positions within a population of chromosomes. For the following results, 100 two-parameter chromosomes were generated uniformly distributed in the range [0,1] for assessing the rank performance.

Scaled versions of the objective functions MOP1, MOP2, and MOP3, defined by Van Veldhuizen and Lamont [10] and given in (8.1, 8.2, & 8.3), were used to provide input data to the ranking processes, with either type A or B noise applied as appropriate. The data were ranked and the assigned rank postion for each chromosome recorded. The process was repeated 1000 times with different values chosen for the applied noise each time. For each chromosome, the standard deviation of the rank position was calculated. The mean standard deviation of the 100 chromosome rank positions was then generated and plotted.

The ranking algorithms from NSGA and MOGA were generated for comparison with the new multi-objective probabilistic selection evolutionary algorithm (MOPSEA) ranking process developed in this paper. With a different set of 100

Table 8.1: Objective Functions

Objective	Definition	Input
MOP1 [10]	$\mathcal{O}_1 = \frac{x^2+y^2}{5000}$ $\mathcal{O}_2 = \frac{(x-2)^2+(y-2)^2}{5408}$ (8.1)	$x = 100\,\chi(1) - 50$ $y = 100\,\chi(2) - 50$
MOP2 [10]	$\mathcal{O}_1 = 1 - \exp\left(-\sum_{i=1}^{n}\left(x_i - \frac{1}{\sqrt{n}}\right)^2\right)$ $\mathcal{O}_2 = 1 - \exp\left(-\sum_{i=1}^{n}\left(x_i + \frac{1}{\sqrt{n}}\right)^2\right)$ (8.2)	$x_i = 4\,\chi(i) - 2$
MOP3 [10]	$\mathcal{O}_1 = \frac{\frac{(x^2+y^2)}{2} + sin(x^2+y^2)}{8.249}$ $\mathcal{O}_2 = \frac{\frac{(3x-2y+4)^2}{8} + \frac{(x-y+1)^2}{27}}{46.940}$ $\mathcal{O}_3 = \frac{\frac{1}{x^2+y^2+1} - 1.1e^{-x^2-y^2} + 0.1}{0.296}$ (8.3)	$x = 6\,\chi(1) - 3$ $y = 6\,\chi(2) - 3$
CON1	$\mathcal{O}_1 = 1.0$ $\mathcal{O}_2 = 1.0$ $0.01 \geq (x-0.5)^2 + (y-0.5)^2$ (8.4)	$x = \chi(1)$ $y = \chi(2)$
CON2 [10]	$\mathcal{O}_1 = x$ $\mathcal{O}_2 = y$ $0 \geq -(x)^2 - (y)^2 + 1 + 0.1\cos\left(16\arctan\left(\frac{x}{y}\right)\right)$ $0.5 \geq (x-0.5)^2 + (y-0.5)^2$ (8.5)	$x = \pi\,\chi(1)$ $y = \pi\,\chi(2)$
EJH1	$a = 3\pi\sqrt[4]{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$ (8.6) $b = 3\pi\sqrt[4]{\frac{1}{n}\sum_{i=1}^{n}(1-x_i)^2}$ (8.7) $\mathcal{O}_1 = 1 - \sqrt[4]{\left\|\frac{\sin(a)}{a}\right\|}$ $\mathcal{O}_2 = 1 - \sqrt[4]{\left\|\frac{\sin(b)}{b}\right\|}$ (8.8)	$x_i = \chi(i)$

initial chromosomes, a slightly different set of graphs will result. The differences have been found to be small however.

For example, in (8.3), $\chi(1)$ and $\chi(2)$ are the two parameters of the input chromosome in the range [0,1]. The parameters x and y are scaled to lie within [-3,3] as defined by Van Veldhuizen and Lamont. The three objective functions are then calculated and scaled to give each of the objectives in the range [0,1]. Noise was then applied either to the input chromosome χ for type A noise, or to the output objectives $\mathcal{O}$ for type B noise. The applied noise was Gaussian with a standard deviation of σ.

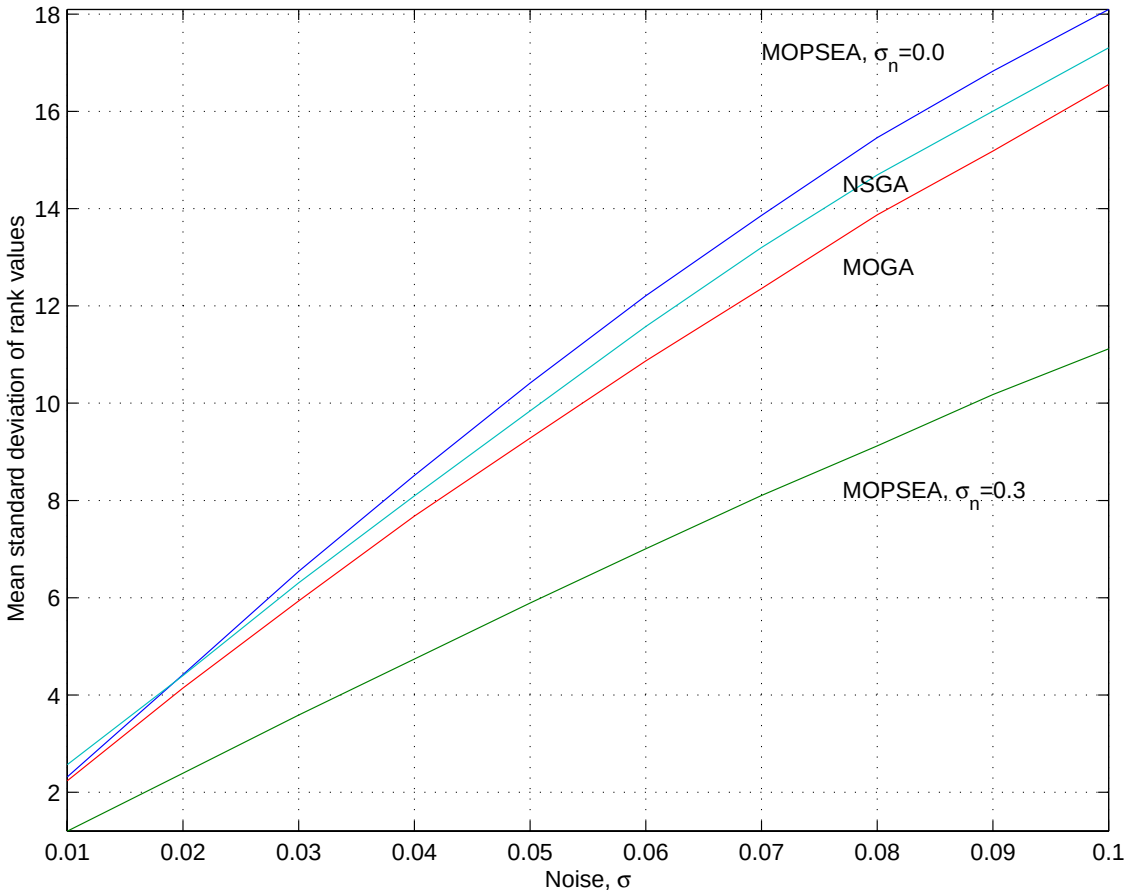

Figure 8.1: Applied noise with respect to mean standard deviation of rank position for MOP1, type A noise. Performance of MOGA and NSGA ranking compared to MOPSEA with $\sigma_n = 0$ & $\sigma_n = 0.3$

From figures 8.1 to 8.8, it is clear that both MOGA and MOPSEA outperform the NSGA ranking process in the presence of noise for this objective function. As the uncertainty parameter σ_n is increased, it is clear that MOPSEA can out perform both alternative algorithms. The specific performance of each algorithm is dependent on the objective function though, and each set of 100 points to be ranked.

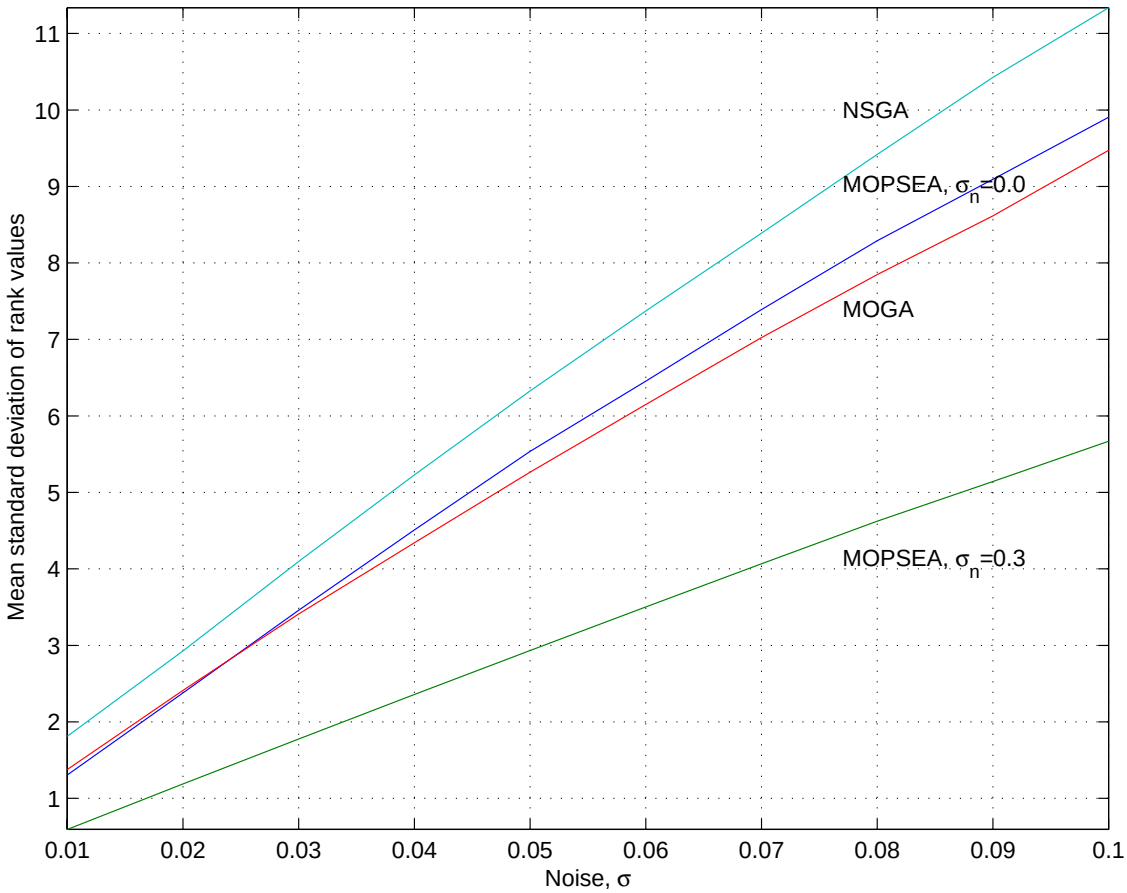

Figure 8.2: Applied noise with respect to mean standard deviation of rank position for MOP1, type B noise. Performance of MOGA and NSGA ranking compared to MOPSEA with $\sigma_n = 0$ & $\sigma_n = 0.3$

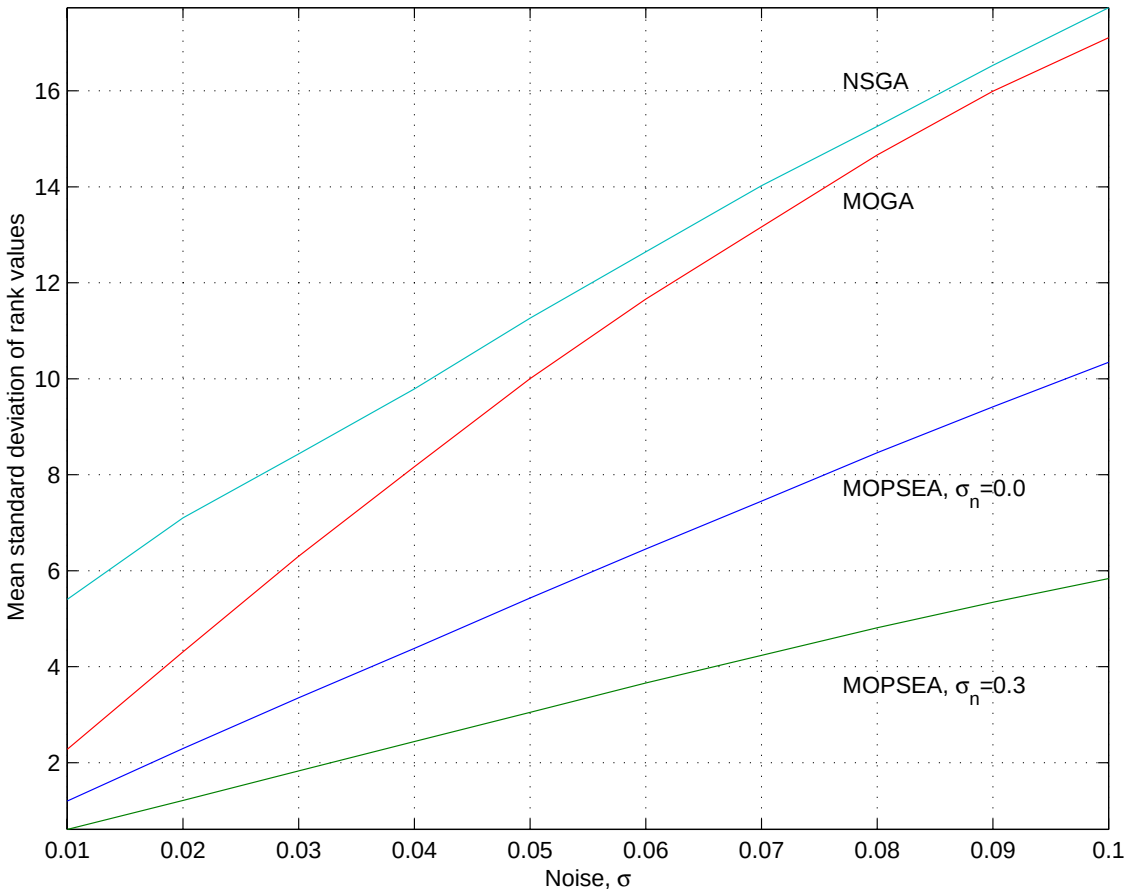

Figure 8.3: Applied noise with respect to mean standard deviation of rank position for MOP2, type A noise. Performance of MOGA and NSGA ranking compared to MOPSEA with $\sigma_n = 0$ & $\sigma_n = 0.3$

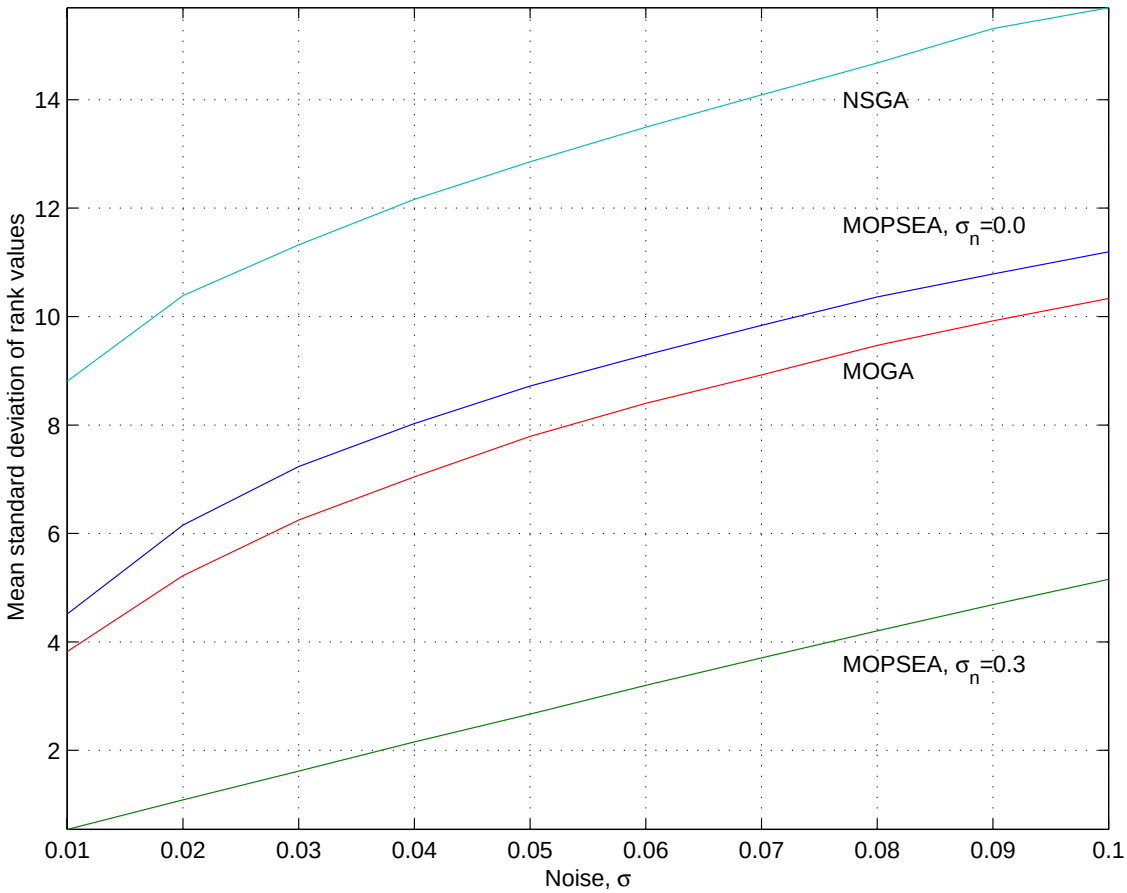

Figure 8.4: Applied noise with respect to mean standard deviation of rank position for MOP2, type B noise. Performance of MOGA and NSGA ranking compared to MOPSEA with $\sigma_n = 0$ & $\sigma_n = 0.3$

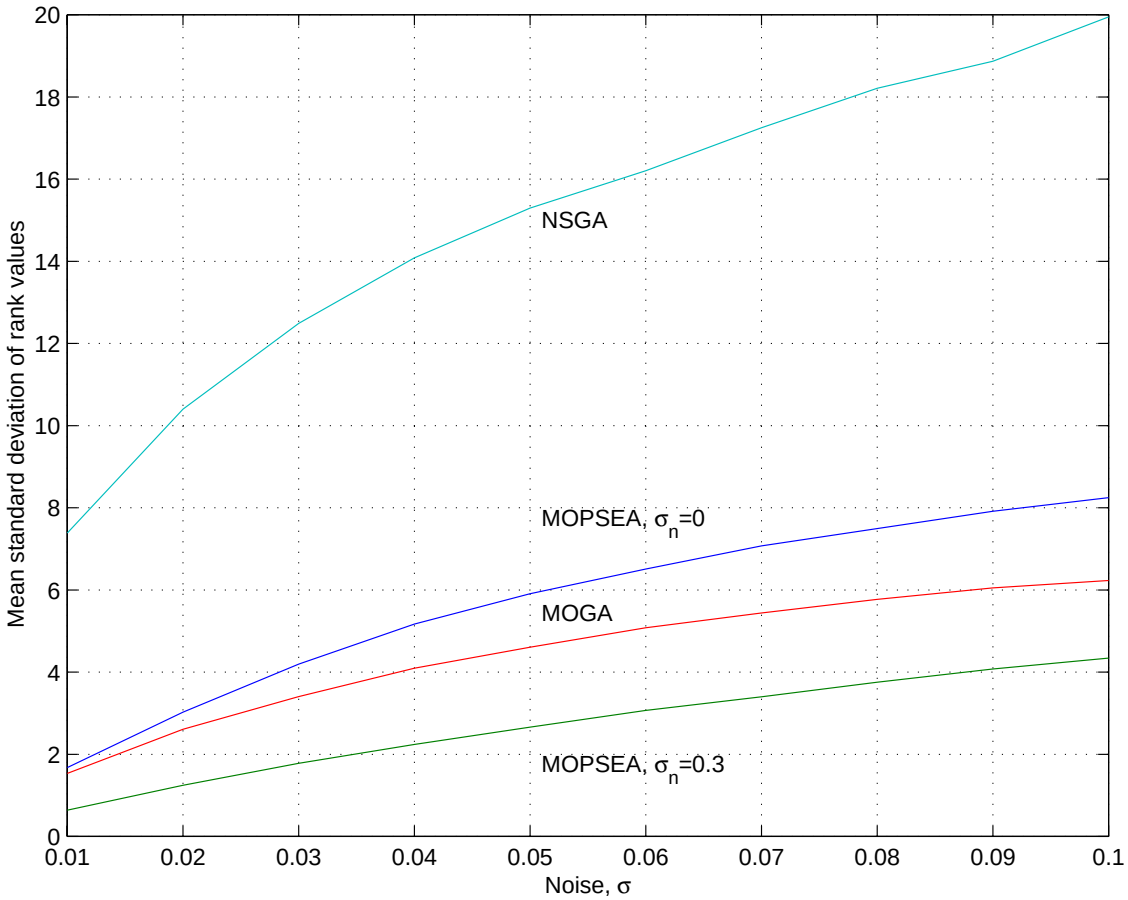

Figure 8.5: Applied noise with respect to mean standard deviation of rank position for MOP3, type A noise. Performance of MOGA and NSGA ranking compared to MOPSEA with $\sigma_n = 0$ & $\sigma_n = 0.3$

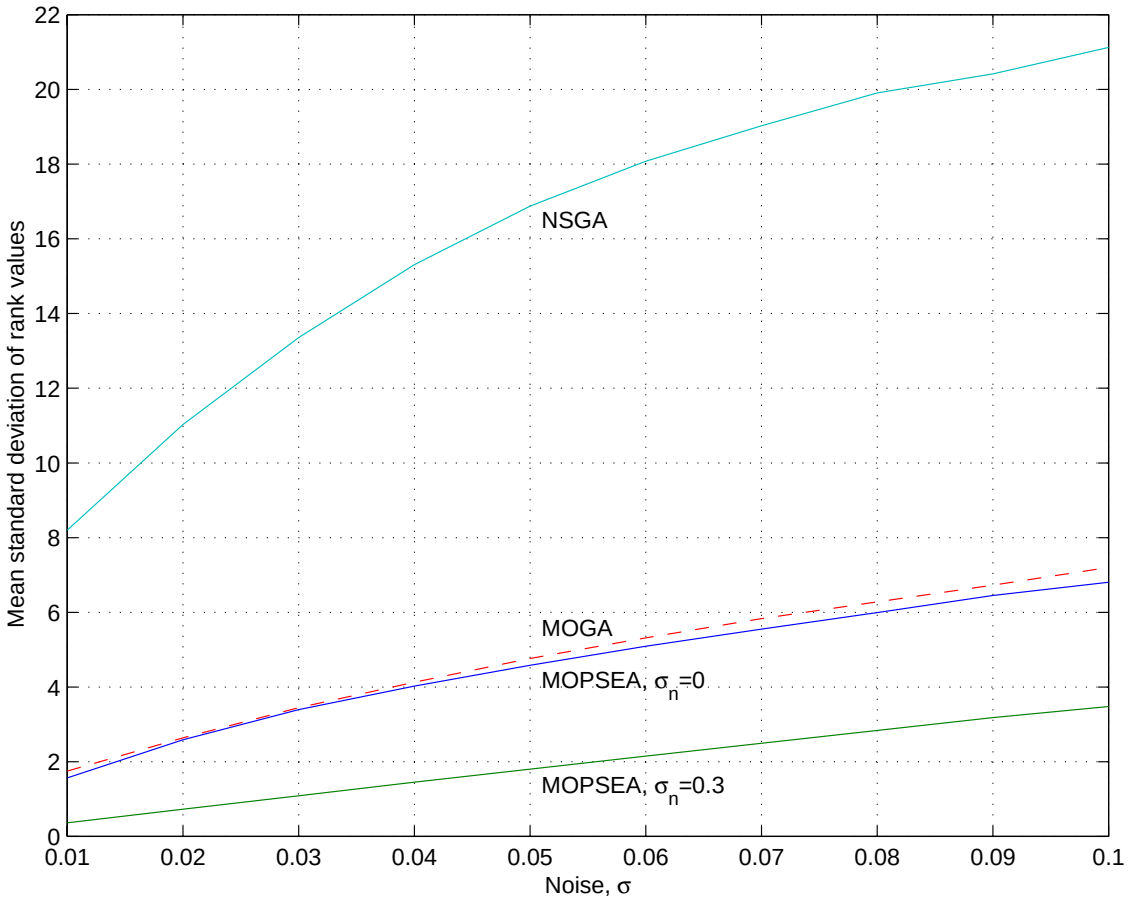

Figure 8.6: Applied noise with respect to mean standard deviation of rank position for MOP3, type B noise. Performance of MOGA and NSGA ranking compared to MOPSEA with $\sigma_n = 0$ & $\sigma_n = 0.3$

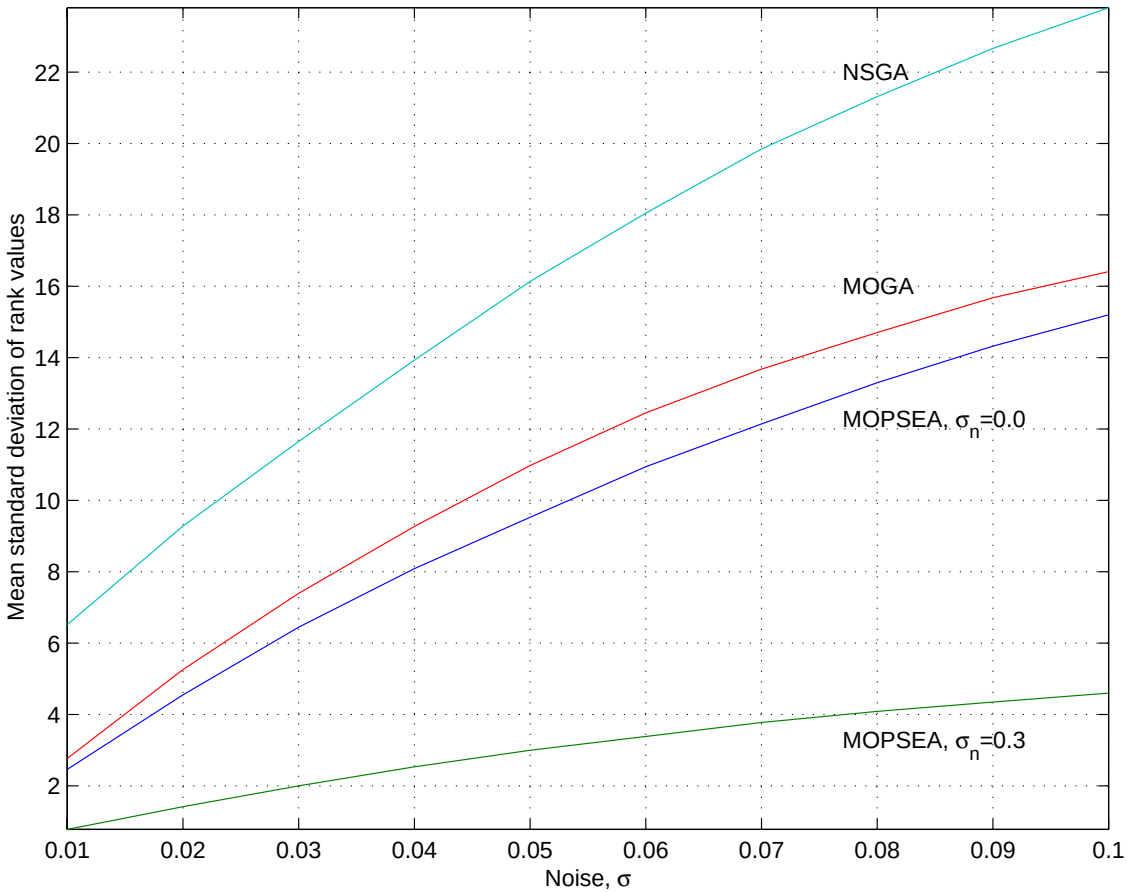

Figure 8.7: Applied noise with respect to mean standard deviation of rank position for EJH1, type A noise. Performance of MOGA and NSGA ranking compared to MOPSEA with $\sigma_n = 0$ & $\sigma_n = 0.3$

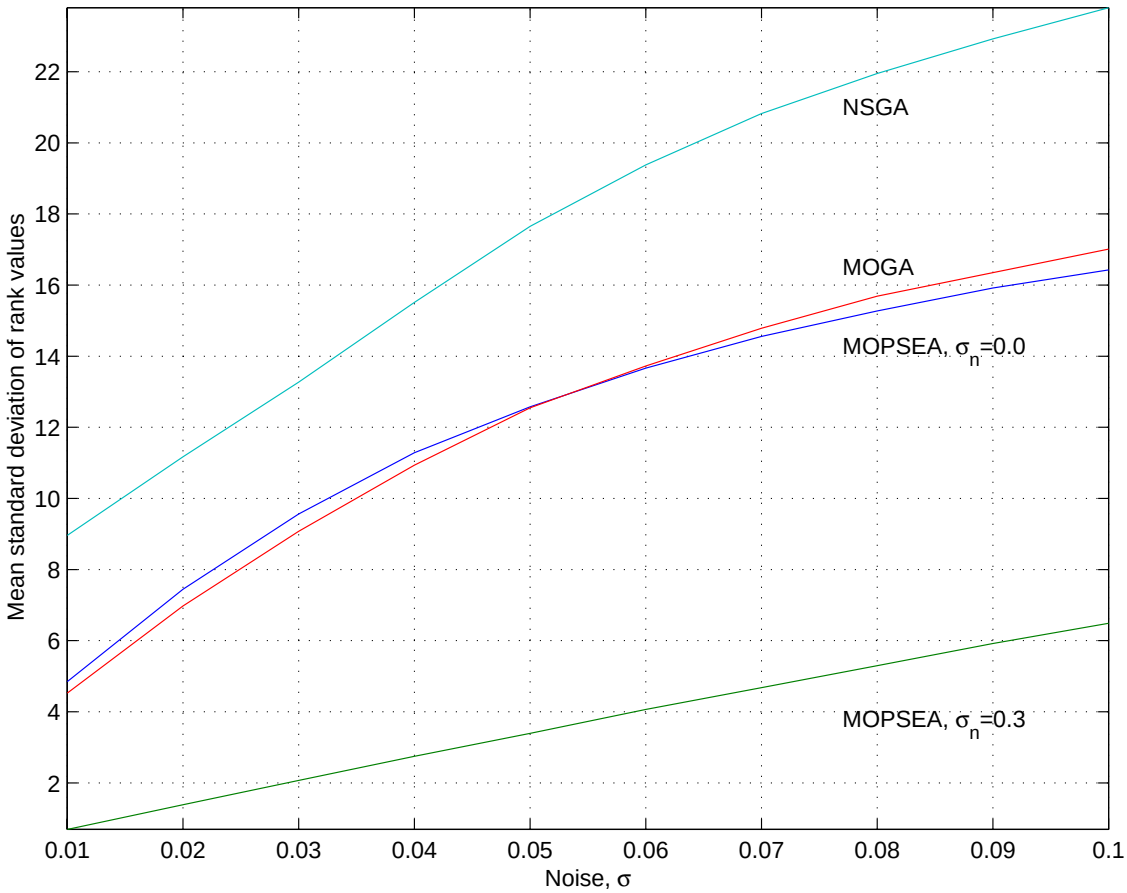

Figure 8.8: Applied noise with respect to mean standard deviation of rank position for EJH1, type B noise. Performance of MOGA and NSGA ranking compared to MOPSEA with $\sigma_n = 0$ & $\sigma_n = 0.3$

8.3.2. Limits, Priority, and Constraints

The following graphs demonstrate the effectiveness of the ranking equations under the conditions of:

1. Limits on objectives.
2. Objective priority.
3. Parameter constraints.

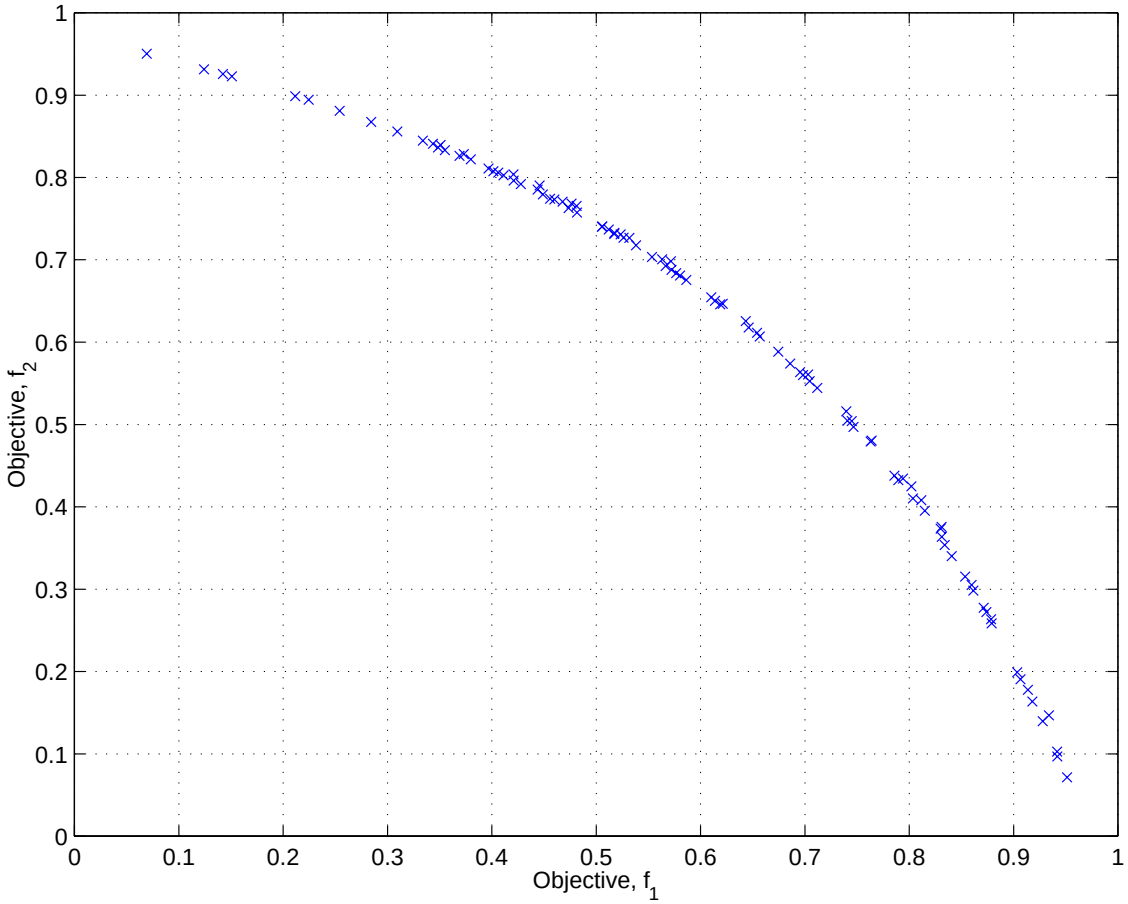

Figure 8.9: MOP2 objective space. No noise, No limits, No prefe rence

Both types of noise are of interest and often the observed noise will be a combination of type A and B. A range of test objectives were developed for the trials. Table 8.1 lists the objective functions used, with either type A or type B noise as appropriate.

The evolutionary algorithm used was a simple structure with selection, crossover, and mutation. A population of 100 individuals was used with chromosomes consisting of two real-valued genes with values lying in the range [0,1]. Stochastic universal sampling was used to select individuals for breeding and then intermediate crossover at a rate of 70% and uniformly distributed mutation at a rate of 10% were applied to generate new individuals. The best 70% were inserted back into the population. The plots shown were all taken after 50 generations.

8.3.3. Objective Limits

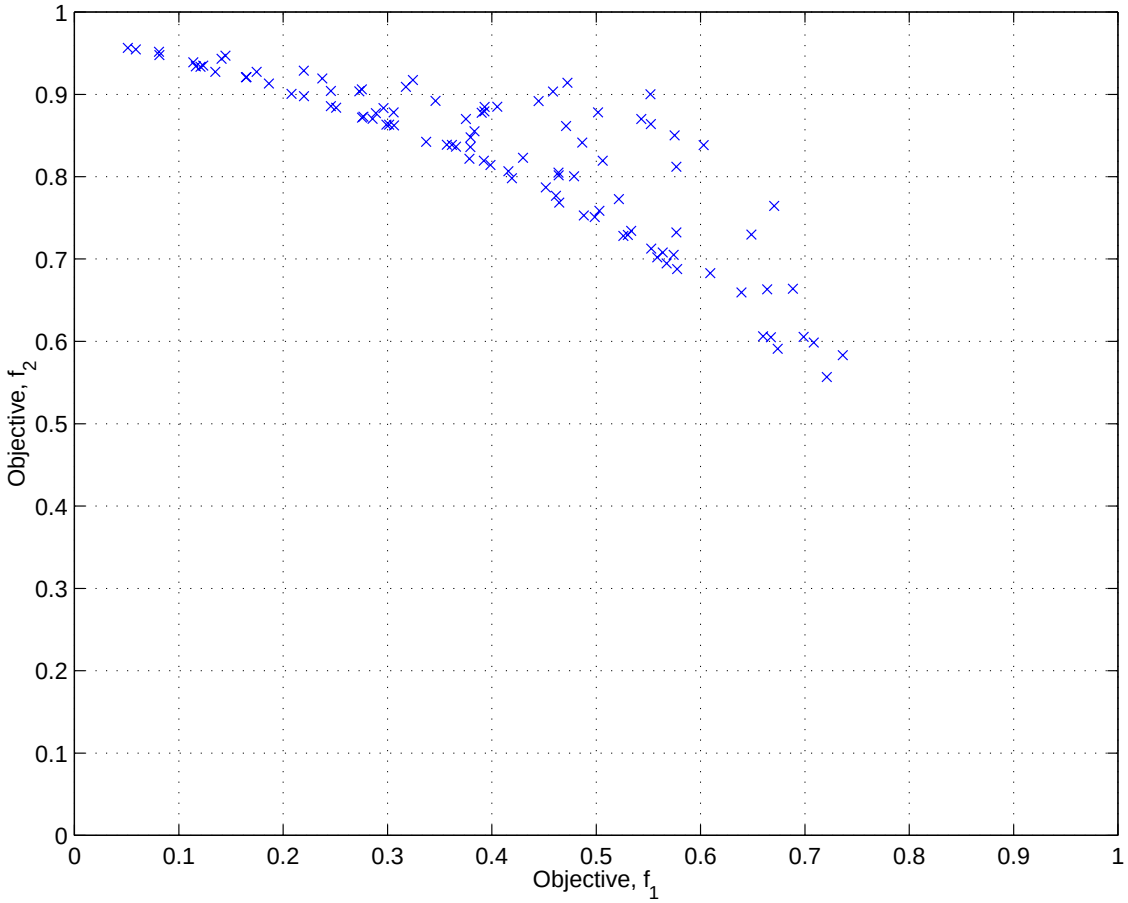

Figure 8.10: MOP2 objective space. Noise $\sigma = 0.1$, Type A, $\sigma_n = 0.1$, Limits [0.7 0.9], No preference

Figure 8.9 shows the objective MOP2 without noise, constraints, preferences, or sharing applied. Figure 8.12 shows the effect of applying the limits [0.7 0.9] to objectives f_1 and f_2 respectively with $\sigma_n = 0$ used to give a rapid transition from constrained to unconstrained.

Figure 8.10 shows how the ranking process copes with significant type A noise. The chromosome values have been perturbed, leading to a smaller region in the chromosome space being responsible for the spread of objective values seen. Despite the noise, the search is still focussed in the required region.

Figure 8.11 shows the effect of type B noise. Here the objective value itself has been perturbed. Again the chromosomes occupy a smaller region, with the perturbed objective values still being focussed. By focusing the objectives, the spread of chromosomes is also reduced. With noisy objectives, if the objectives are constrained too much, there may be no single chromosome that will always have a perturbed solution within the constrained region. This can cause problems with genetic drift and loss of diversity within the population.

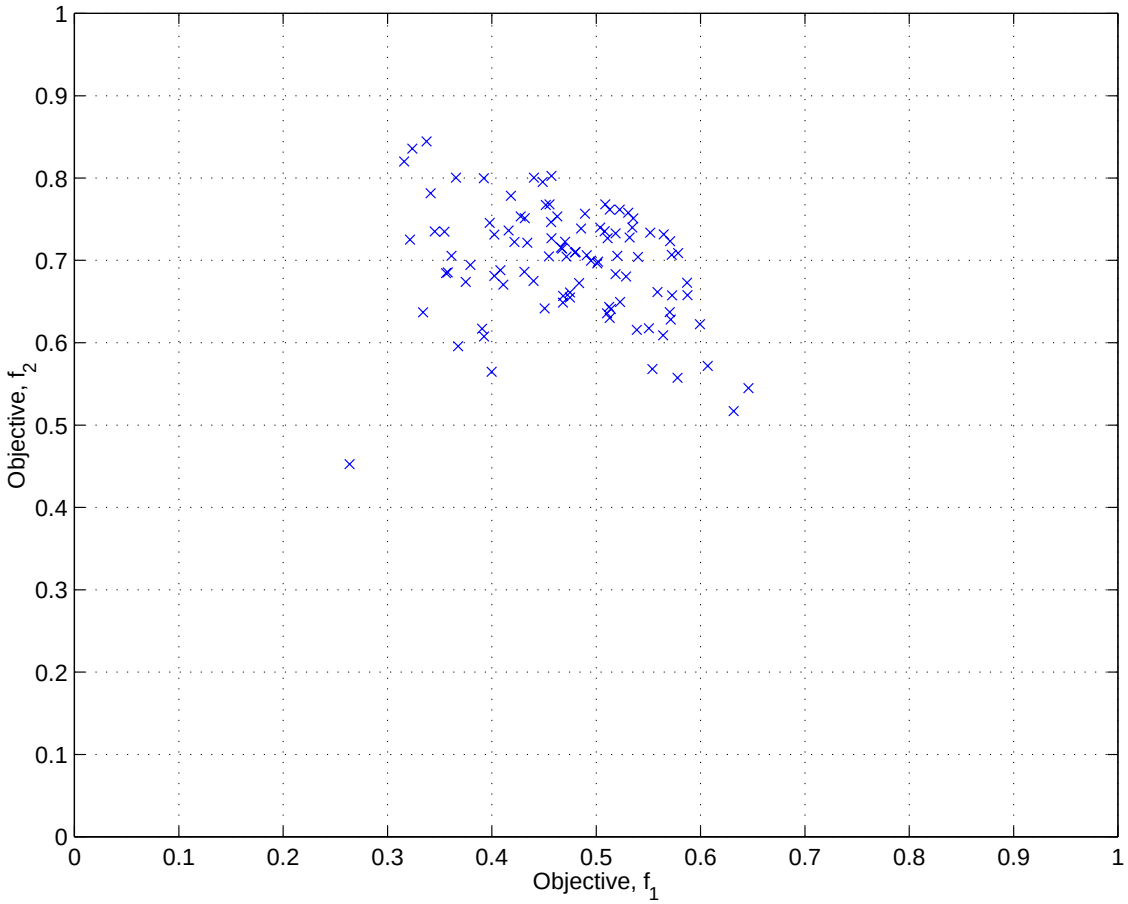

Figure 8.11: MOP2 objective space. Noise $\sigma = 0.1$, Type B, $\sigma_n = 0.1$, Limits [0.7 0.9], No preference

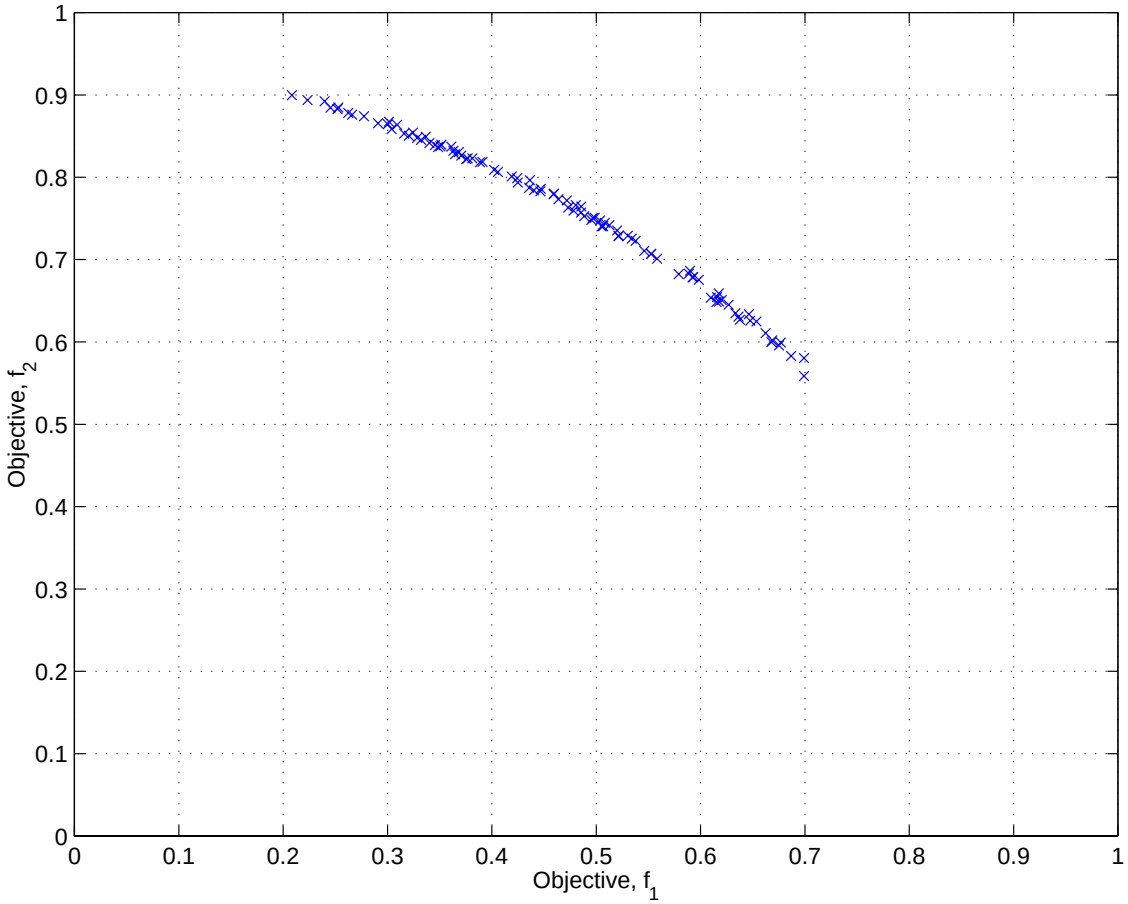

Figure 8.12: MOP2 objective space. No noise, Limits [0.7 0.9], No preference

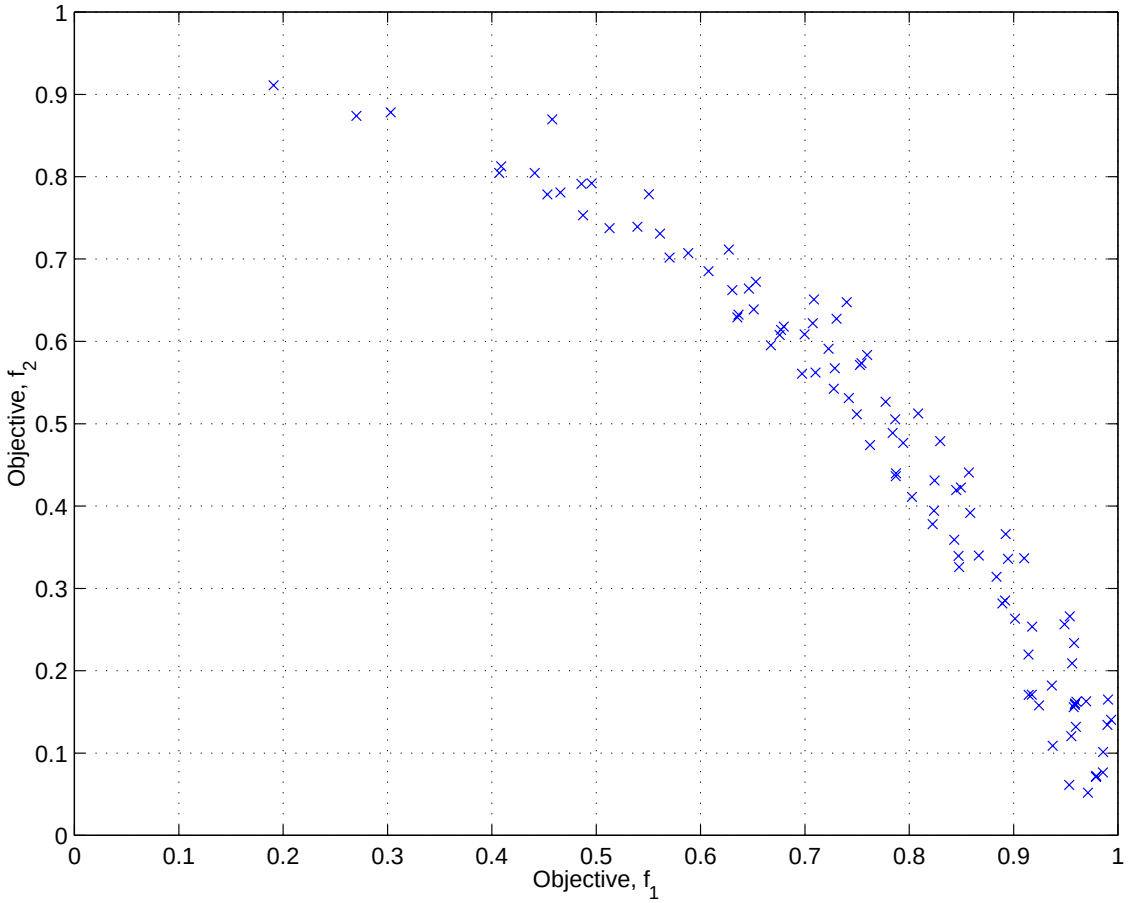

Figure 8.13: MOP2 objective space. No noise, No limits, Preference [0.9 1.0], Sharing 0.005 on objectives

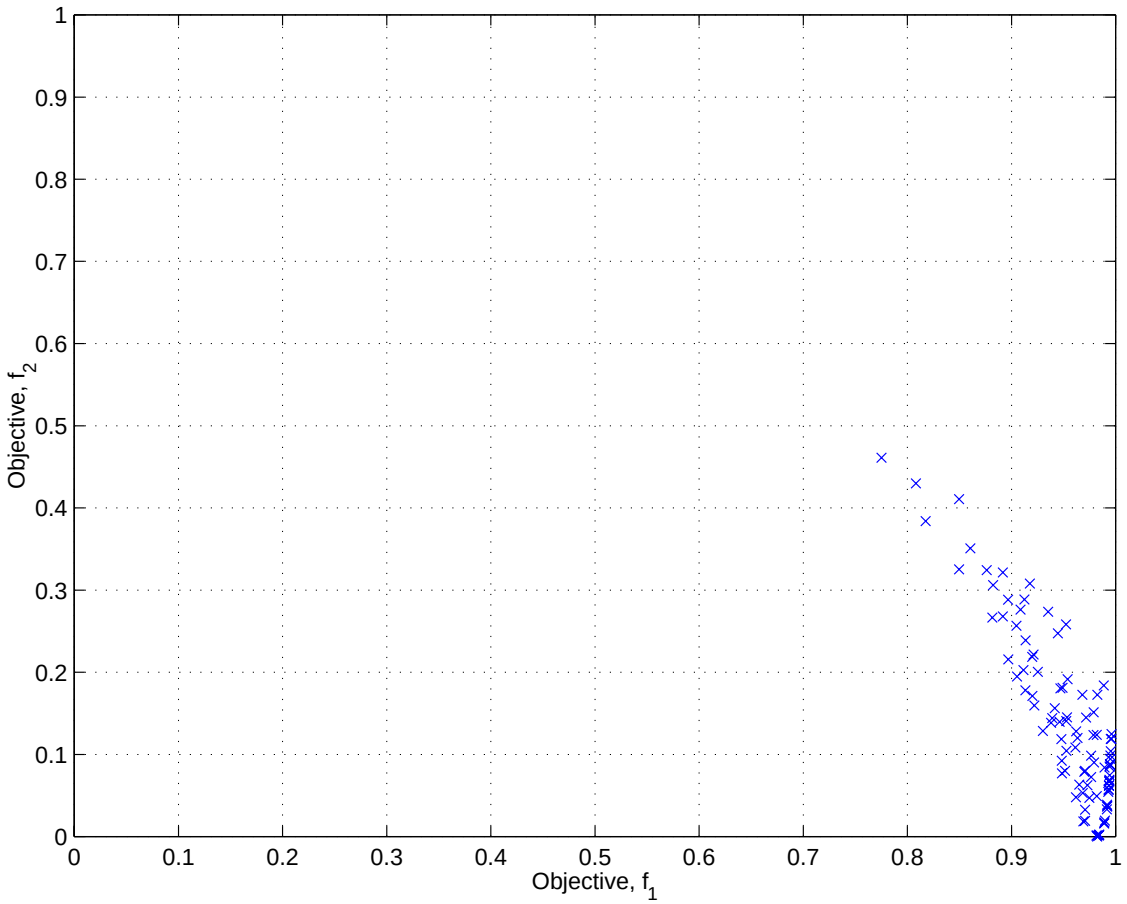

Figure 8.14: MOP2 objective space. No noise, No limits, Preference [0.5 1.0], Sharing 0.005 on objectives

8.4. Objective Preference

Often with multiple objectives, not all the objectives are of equal interest to the designer. For example, in a cost / performance tradeoff, if very high volumes are to be manufactured, the cost is often paramount.

Figure 8.13 shows the function MOP2 without noise or objective limits but with the priorities [0.9 1.0] specified for objectives f_1 and f_2 respectively. As the objectives are minimised, preferred solutions are better on f_2 and therefore will be worse on f_1 and so will tend to lie towards the bottom right of the plot. A small amount of sharing has been applied to reduce genetic drift. It is clear that f_2 is dominating the resul ts.

Figure 8.14 shows the effect of a preference vector [0.5 1.0]. Here f_1 is penalised further. If the vector [0 1.0] was used, only f_2 would have any influence on the ranking process.

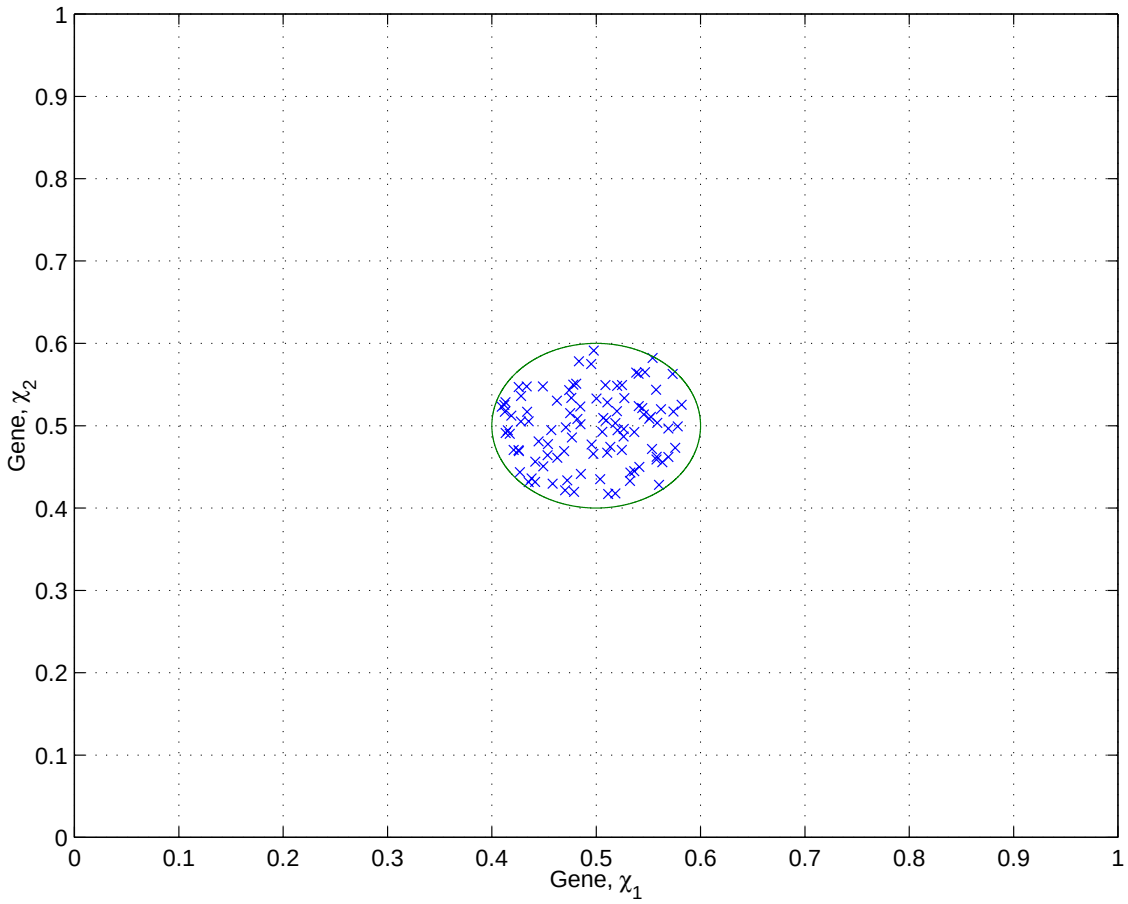

Figure 8.15: CONS1 chromosome space

8.5. Chromosome Constraints

Figure 8.15 shows the chromosome locations for the problem CONS1, along with the boundary of the constrained region. Both objective values always equal unity

and therefore only the constraints have any effect. It is clear that the algorithm quickly converges to the constrained region. In problems of this type, it would be advisable to use sharing on the chromosome positions to try to reduce the effects of genetic drift.

Figure 8.16 shows the non-dominated boundary of the CONS2 function. The discontinuous objective surface can be seen clearly. The handling of the constraints as part of the ranking process treats solutions that do not satisfy constraints as non-dominated, and so share rank positions. This reduces the rank value and effective selective pressure of the individuals, allowing the solutions that satisfy the constraints to dominate.

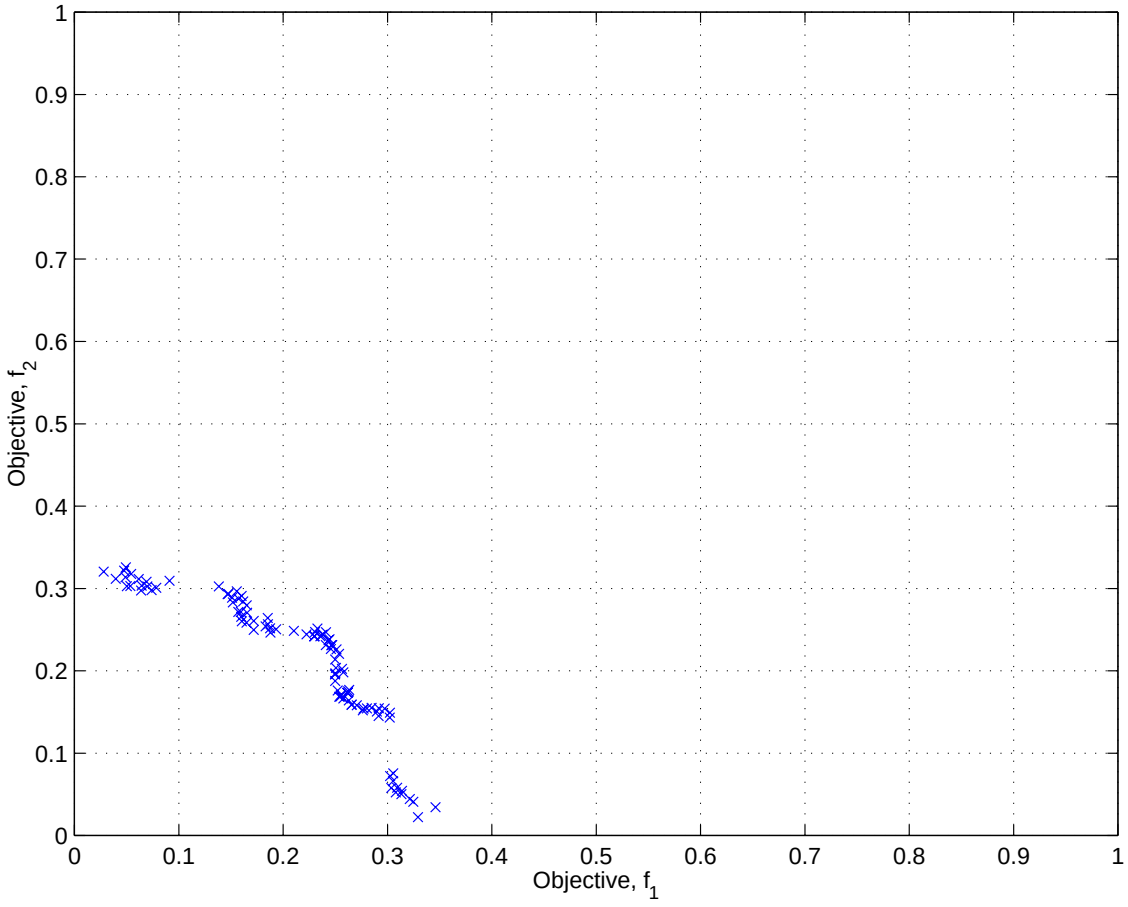

Figure 8.16: CONS2 Objective space

9. Conclusions

The results have shown that the modified ranking process can reduce the disturbances in the rank positions caused by noisy objectives. Unlike conventional ranking processes, the rank values and therefore the corresponding selection probabilities take some account of the noise and uncertainty in the system. The theory developed in this paper forms an important first step towards addressing directly noise and uncertainty in multi-objective problems. The simplicity of the ranking and selection equations may also provide a route to further theoretical research into the operation and performance of evolutionary algorithms.

The integrated ranking, constraint, and priority equations that have been developed form a first step towards evolutionary algorithms that can address the problems of noisy objective functions directly. By integrating the constraints and priorities into the ranking, the rank values maintain their consistency and allow selection probabilities to be calculated easily.

The new ranking, constraint, and preference equations are simple functions, unlike many existing ranking processes that are based on logical decisions and are difficult to manipulate mathematically. This may help in the analysis of algorithm operation in the future. By reducing the effects of the noise on the rank positions, the evolutionary process is more stable and with the inclusion of constraints and preferences, allows the designer full interactive control over the evolutionary process.

9.1. Acknowlegements

The author would like to acknowledge the use of the Department of Aerospace, Power, and Sensors DEC Alpha Beowulf cluster for the production of the results.

Bibliography

[1] T. W. Then and Edwin K. Chong. Genetic algorithms in noisy environment. In *IEEE International Symposium on Intelligent Control*, pages 225–230, Columbus, Ohio, 16-18 August 1994.

[2] Adrian Thompson. Evolutionary techniques for fault tolerance. In *Control '96, UKACC International Conference on*, volume 1, pages 693–698. IEE, 2-5 September 1996. Conf. Publ. No. 427.

[3] T. Bäck and U. Hammel. Evolution strategies applied to perturbed objective functions. In *IEEE World Congress on Computational Intelligence*, volume 1, pages 40–45. IEEE, 1994.

[4] Shigeyoshi Tsutsui and Ashish Ghosh. Genetic algorithms with a robust searching scheme. *IEEE Transactions on Evolutionary Computation*, 1(3):201–8, September 1997.

[5] Kumar Chellapilla and David B. Fogel. Anaconda defeats hoyle 6-0: A case study competing an evolved checkers program against commercially available software. In *Congress on Evolution Computation - CEC2000*, volume 1, pages 857–863, San Diego, CA, 16-19 July 2000. IEEE.

[6] Carlos A. Coello Coello. List of references on evolutionary multiobjective optimization. http://www.lania.mx/∼ccoello/EMOO/EMOObib.html. Last accessed 3 July 2000.

[7] Anna L. Blumel, Evan J. Hughes, and Brian A. White. Fuzzy autopilot design using a multiobjective evolutionary algorithm. In *Congress on Evolution Computation - CEC2000*, volume 1, pages 54–61, San Diego, CA, 16-19 July 2000. IEEE.

[8] Carlos M. Fonseca and Peter J. Flemming. Multiobjective genetic algorithms made easy: Selection, sharing and mating restriction. In *GALESIA 95*, pages 45–52, 12-14 September 1995. IEE Conf. Pub. No. 414.

[9] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995.

[10] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical Report TR-98-03, Air Force Institute of Technology, 1 Dec 1998.

[11] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *NUMERICAL RECIPES in C. The Art Of Scientific Computing*. Cambridge University Press, second edition, 1993.

[12] J. E. Baker. Adaptive selection methods for genetic algorithms. In *Proc. 1st Int. conf. on* Genetic Algorithms, pages 101–111, 1985.

[13] Dragan Cvetković and Ian C. Parmee. Genetic algorithm-based multi-objective optimisation and conceptual engineering design. In *Congress on Evolutionary Computation - CEC99*, volume 1, pages 29–36, Washington D.C., USA, July 1999. IEEE.

[14] Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311–338, June 2000.

[15] Evan J. Hughes. Multi-objective probabilistic selection evolutionary algorithm. Technical Report DAPS/EJH/56/2000, Dept. Aerospace, Power, & Sensors, Cranfield University, RMCS, Shrivenham, UK, SN6 8LA, September 2000.

[16] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI, Berkeley, March 1995. ftp.icsi.berkeley.edu.

A. Matlab Mex-File

```
 /*

noisy non-dominated ranking
Matlab Mex-file.  Compiled on Sun Ultra 10, Matlab 5.3

[prob(select), rank, sharecnt, q] = NoisyNondom(obj, sigmaobj,
                                    sigmashare, objlimits, constraints,
    priority, selective_pressure)

input:

obj         - p x k : objective values to minimise
sigmaobj    - 1 x k : standard dev. of objective noise
sigmashare  - 1 x k : share distance for objectives
objlimits   - 1 x k : objective constraint vector
constraints - p x 1 : constraint vector - noise processing done externally
priority    - 1 x k : [1 1 .. 1] gives all same pri.
sel_press   - 1 x k : [1 1 .. 1] for maximum selective pressure

p is population size, k is number of objectives.

output:

prob(select) will sum to unity, best has largest probability.

rank is 'rank fitness', smallest the best, minimising all objectives
for maximisation, use negative std dev in sigmaobj.

sharecnt is the share count for each individual.

q is the final constraint value, after objective constraints,
   parameter constraints and sharing.

time is O(n^2)   for tanh operations

E.J.Hughes 28 July 2000

*/
```

```
#include "mex.h"
#include <math.h>
#include <stdlib.h>

#ifndef PI
#define PI 3.141592653589793
#endif

#define SQ2 0.7071067811865475

#define obj(i,j) obji[(i)+(popsize*(j))]
#define t(i,j)    tmp[(i)+(popsize*(j))]

void NoisyNondom(double *err, double *nerr, double *qerr, double *errx,
 double *obji, double *st, double *shr, double *q,
                 double *c, double *pr, double *sp, int popsize, int nobj)
{
  double *tmp, *tmpn, *tmpq, *pq, *stmp, *prtmp, *prtmpb, *sptmp, *sptmpb;
  double tt,st1,st2,tn,sumq,numdiv,tx, shfact, prtot;
  int i,j,k;


  tmp = (double *)mxCalloc(popsize*nobj,sizeof(double));
  tmpn = (double *)mxCalloc(popsize,sizeof(double));
  pq = (double *)mxCalloc(popsize,sizeof(double));
  tmpq = (double *)mxCalloc(nobj,sizeof(double));
  stmp = (double *)mxCalloc(nobj,sizeof(double));
  prtmp = (double *)mxCalloc(nobj,sizeof(double));
  prtmpb = (double *)mxCalloc(nobj,sizeof(double));
  sptmp = (double *)mxCalloc(nobj,sizeof(double));
  sptmpb = (double *)mxCalloc(nobj,sizeof(double));

  /* precalc sharing factors */

  shfact=0;
  for(k=0;k<nobj;++k)
    {
      st1=shr[k]/(st[k]+1.0e-10)/sqrt(2.0);
      shfact+=log(st1)-log(st1*st1+1)/2.0;/* log[s/sqrt(s^2+1)] */
      stmp[k]=1.0/4.0/(st1*st1+1);  /* Gaussian sharing */
    }

  /* precalc effects of sigma
       - beware sigma == 0, not trapped properly*/

  /* process fitness values */
```

```
  for (i=0;i<popsize;++i)
    for (k=0;k<nobj;++k)
      t(i,k)=obj(i,k)/(st[k]+1.0e-10);

  if(q)
  /* process restriction point values */
    for (k=0;k<nobj;++k)
      tmpq[k]=q[k]/(st[k]+1.0e-10)/1.6;

  /* zero elements */

  for (i=0;i<popsize;++i)
    tmpn[i]=pq[i]=err[i]=0.0;

  if(q)
  /* calc probability of dominating the restriction point Q */
    for (i=0;i<popsize;++i)
      {
st1=1.0;
for (k=0;k<nobj;++k)
  st1*=(1.0+tanh(tmpq[k]-t(i,k)/1.6))/2.0;  /* probability */
pq[i]=st1;
      }
  else
    for (i=0;i<popsize;++i)
      pq[i]=1.0;

  if(c)
    for (i=0;i<popsize;++i)   /* apply constraints */
      pq[i]*=c[i];

  if(pr)
    for (i=0;i<nobj;++i)   /* set priority */
      prtmp[i]=pr[i];
  else
    for (i=0;i<nobj;++i)   /* set default priority */
      prtmp[i]=1.0;

  prtot=1.0;
  for (i=0;i<nobj;++i)   /* pre-calc for priority */
    {
      prtmpb[i]=1.0-prtmp[i];
      prtot*=prtmpb[i];
    }

  prtot = 1.0-prtot;
```

```
  if(sp)
    for (i=0;i<nobj;++i)   /* set selective pressure */
      sptmp[i]=sp[i];
  else
    for (i=0;i<nobj;++i)   /* set default SP */
      sptmp[i]=1.0;

  for (i=0;i<nobj;++i)   /* pre-calc for SP */
      sptmpb[i]=(1.0-sptmp[i])/2.0;


  /* Calculate share count */
  for (i=0;i<(popsize-1);++i)
    for (j=i+1;j<popsize;++j)
      {
tn=shfact;
for (k=0;k<nobj;++k)
  {
    tx=t(i,k)-t(j,k);   /* calculated k */
    tn-=tx*tx*stmp[k];  /* sharing */
  }
tn=exp(tn/(double)nobj);
tmpn[i]+=tn;  /* share info */
tmpn[j]+=tn;
      }

    for (i=0;i<popsize;++i)
     pq[i]/=(tmpn[i]+1);    /* add 1 to account for self, and apply sharing */


  /* calc probability table */
  /* sum probability so if on Pareto, will calc how many we dominate */
  /* do not count self comparison */

  for (i=0;i<(popsize-1);++i)
    for (j=i+1;j<popsize;++j)
      {
st1=prtot; /* for priority */
st2=prtot; /* for priority */
for (k=0;k<nobj;++k)
  {
    tt=(1.0+tanh((t(i,k)-t(j,k))/1.6))/2.0;  /* probability A_i>B_i*/
    /*      st1*=tt;
    st2*=1.0-tt; */
    /* do priority & SP */
    st1*=(tt*sptmp[k]+sptmpb[k])*prtmp[k]+prtmpb[k];
    st2*=((1.0-tt)*sptmp[k]+sptmpb[k])*prtmp[k]+prtmpb[k];
```

```
  }
st1=st1*pq[i]*pq[j]+pq[j]*(1-pq[i]);  /* apply constraints */
st2=st2*pq[i]*pq[j]+(1-pq[j])*pq[i];
tx=(1-st1-st2)/2.0;      /* non-domination */
err[i]+=st1+tx;
err[j]+=st2+tx;
      }

    for (i=0;i<popsize;++i)
     ++tmpn[i];           /* add 1 to account for self */


  if(errx)                    /* save raw info */
    for (i=0;i<popsize;++i)
      errx[i]=err[i];

  if(nerr)                    /* save raw info */
    for (i=0;i<popsize;++i)
      nerr[i]=tmpn[i];

  if(qerr)                    /* save raw info */
    for (i=0;i<popsize;++i)
      qerr[i]=pq[i];

/*     correct to make probabilities that sum to unity  */

  tt=(double)(popsize-1);
  sumq=2.0/(double)(popsize*(popsize-1));
  for (i=0;i<popsize;++i)
    err[i]=(tt-err[i])*sumq;  /*normalise into prob of selection*/


  mxFree(sptmpb);
  mxFree(sptmp);
  mxFree(prtmpb);
  mxFree(prtmp);
  mxFree(stmp);
  mxFree(tmpq);
  mxFree(pq);
  mxFree(tmpn);
  mxFree(tmp);

}


/* gateway function */
```

```
void mexFunction(int nlhs,mxArray *plhs[], int nrhs, const mxArray *prhs[])

{
  double *errx, *nerr, *qerr, *c, *q, *pr, *sp;
  double freq,ap;
  int m,n,popsize,nobj;

  if (nrhs > 7 || nrhs <3) {
    mexErrMsgTxt("NoisyNondom requires three to seven input arguments.");
  }
  if (nlhs >4) {
    mexErrMsgTxt("NoisyNondom requires one to four output arguments.");
  }

  m = mxGetM(prhs[0]);
  n = mxGetN(prhs[0]);

  if ((m<2)||(n <1))
    mexErrMsgTxt("NoisyNondom requires that obj be a pop x nobj matrix");

  popsize = m;
  nobj=n;

  m = mxGetM(prhs[1]);
  n = mxGetN(prhs[1]);

  if ((m!=1)||(n !=nobj))
    mexErrMsgTxt("NoisyNondom requires that sigma be 1 x nobj");

  m = mxGetM(prhs[2]);
  n = mxGetN(prhs[2]);

  if ((m!=1)||(n !=nobj))
    mexErrMsgTxt("NoisyNondom requires that sigma share be 1 x nobj");

  if(nrhs>=4)
    {
      m = mxGetM(prhs[3]);
      n = mxGetN(prhs[3]);

      if ((m!=1)||(n !=nobj))
mexErrMsgTxt("NoisyNondom requires that Q be 1 x nobj");

      q=mxGetPr(prhs[3]);
    }
  else
```

```
    q=NULL;

  if(nrhs>=5)
    {
      m = mxGetM(prhs[4]);
      n = mxGetN(prhs[4]);

      if ((m!=popsize)||(n !=1))
mexErrMsgTxt("NoisyNondom requires that C be npop x 1");

      c=mxGetPr(prhs[4]);
    }
  else
    c=NULL;

  if(nrhs>=6)
    {
      m = mxGetM(prhs[5]);
      n = mxGetN(prhs[5]);

      if ((m!=1)||(n !=nobj))
mexErrMsgTxt("NoisyNondom requires that priority be 1 x nobj");

      pr=mxGetPr(prhs[5]);
    }
  else
    pr=NULL;

  if(nrhs>=7)
    {
      m = mxGetM(prhs[6]);
      n = mxGetN(prhs[6]);

      if ((m!=1)||(n !=nobj))
mexErrMsgTxt("NoisyNondom requires that SP be 1 x nobj");

      sp=mxGetPr(prhs[6]);
    }
  else
    sp=NULL;

  plhs[0] = mxCreateDoubleMatrix(popsize,1,mxREAL);

  errx=NULL;
  nerr=NULL;
  qerr=NULL;
  if(nlhs>=2)
```

```
    {
      plhs[1] = mxCreateDoubleMatrix(popsize,1,mxREAL);
      errx=mxGetPr(plhs[1]);
    }

  if(nlhs>=3)
    {
      plhs[2] = mxCreateDoubleMatrix(popsize,1,mxREAL);
      nerr=mxGetPr(plhs[2]);
    }

  if(nlhs==4)
    {
      plhs[3] = mxCreateDoubleMatrix(popsize,1,mxREAL);
      qerr=mxGetPr(plhs[3]);
    }

  NoisyNondom(mxGetPr(plhs[0]), nerr, qerr, errx, mxGetPr(prhs[0]),
      mxGetPr(prhs[1]), mxGetPr(prhs[2]), q, c, pr, sp, popsize, nobj);

} /* mexFunction */
```