Chapter 5

Parallel GAs for Multiobjective Functions

Jon Rowe, **Kevin Vinsen**, and **Nick Marvin**
Artificial Intelligence Group
Department of Computer and Information Sciences
De Montfort University
Milton Keynes MK7 6HP
United Kingdom
E-mail: jrowe@dmu.ac.uk

Abstract. Multiobjective functions require the simultaneous optimisation of a problem along a number of dimensions. Several methods for adapting genetic algorithms for this purpose have been proposed, including the use of Pareto-ranking as means of comparing solutions across many objectives, and sharing. It is important to generate as many non-dominated solutions as possible, and the GA technique of sharing has been previously used for this purpose. In this paper, parallel diffusion GAs are used to provide a natural niching mechanism that has considerable computational advantages. The diffusion GA is compared with a sharing GA for multiobjective functions and some results presented. A particular multiobjective function of interest is derived from a sensitivity analysis scenario.
Keywords: parallel processing, multiobjective optimization

The complete title of this article: "Parallel Genetic Algorithms for Multiobjective Functions". The article is available via anonymous ftp at site ftp.uwasa.fi directory cs/2NWGA as file Rowe.ps.Z.

5.1 Introduction and overview

Genetic algorithms (GAs) are often used to try to find optimal or near optimal solutions to problems. A multiobjective problem is one in which there is more than one measure of fitness to be optimised. It is unlikely that a solution which optimises one of the objectives will be optimal for any of the others. Compromise solutions are therefore sought such that no other solutions are better in any one objective while remaining no worse in the others. Such solutions is called non-dominated.

Section 5.2 of this paper introduces multiobjective function optimisation more formally and gives some examples and applications. The notions of dominance and Pareto-ranking are also defined. The problem of multiobjective optimisation is to find the whole set of non-dominated solutions.

There have been a number of methods proposed to adapt genetic algorithms to multiobjective problems [190]. These are summarised in Section 5.3. They fall into two categories. Firstly, there are methods that try to combine all the different objectives into one. Secondly, there are methods based on Pareto-ranking. The adaptation of sharing in GAs is also presented.

The parallel genetic algorithm known as a diffusion GA has a natural niching behaviour [219, 386]. It works by having a distributed population in which members can only mate with their immediate neighbours. Geographical niches form because of this spatial isolation. This makes it an ideal method for multiobjective optimisation. It is described in Section 5.4.

An interesting area of application is in sensitivity analysis. Rather than trying to find an optimal solution, we try to criticise a chosen solution by finding small perturbations to environmental parameters to which it is sensitive. The problem of finding the most sensitive perturbations can be formulated as a multiobjective function, as described in Section 5.5.

Section 5.6 uses an example function from an artificial sensitivity analysis problem to compare the behaviour of a diffusion GA and a sharing GA. In addition to providing computational advantages, the diffusion GA seems (at least on this function) to find a better set of non-dominated solutions than the sharing GA.

5.2 Optimising multiobjective functions

Let S be the search space under consideration. That is, it is the set of all possible solutions to our problem. Suppose that we have a number of objective functions $f_1, \ldots, f_z$ such that $\forall i \in \{1, \ldots, z\}, f_i : S \to \mathcal{R}$. The problem is to find an $x \in S$ that maximises every $f_i(x)$. This is known as a multiobjective optimisation problem. (Naturally, the problem can always be reposed in terms of minimisation or as a mixture of minimising and maximising the various $f_1, \ldots, f_z$).

Examples of this kind of problem can be found in a number of domains, including engineering design, financial investment, planning and interior decorating. Indeed, any situation in which a number of constraints have to be met simultaneously, can be posed as a multiobjective optimisation problem. A novel application described in this paper is in couching sensitivity analysis as a multiobjective problem.

In all but the simplest cases, it will not be possible to find a solution to a multiobjective optimisation problem. Successfully maximising one of the component functions will typically degrade performance in another. In practice, we seek solutions such that no one component function can be improved without sacrificing another. Such solutions are called Pareto-optimal.

More formally, we say of two solutions $x, y \in S$ that x dominates y if $\exists i \in \{1, \ldots, z\}$ such that $f_i(x) > f_i(y)$ and that $\forall j \in \{1, \ldots, z\}, f_j(x) \geq f_j(y)$. In other words, x dominates y if x is better than y for at least one objective function, and is no worse on any of the others. A solution is Pareto-optimal if it is not dominated by any other solution. Figure 5.1 illustrates a situation in which x and y are non-dominated, and z is dominated by x.

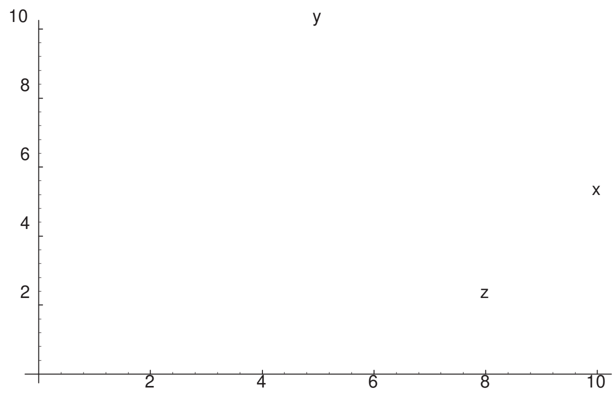

Figure 5.1: x and y are not dominated. x dominates z.

Ideally, we would like to find the set of all Pareto-optimal solutions. From amongst these, a final decision will be reached using various preference criteria (presumably qualitative).

5.3 GA methods

A number of adaptations to GAs have been proposed to deal with multiobjective functions [190, 413]. The simplest method is to combine the various objective functions into a single fitness function. This might be done using a weighted sum method, by defining:

$$f(x) = \sum_{i=1}^{z} w_i f_i(x) \tag{5.1}$$

for various weights $w_i \in \mathcal{R}$. The obvious drawback here is that the weights have to be pre-specified. Part of the problem here is that it is not fully understood what the effects of varying the weights will have on a GAs behaviour.

If the optimal values for each individual objective function is known to be $y_i, i \in \{1, \ldots, z\}$ then a distance method can be used:

$$f(x) = \left[\sum_{i=1}^{z} |f_i(x) - y_i|^r \right]^{\frac{1}{r}} \tag{5.2}$$

Apart from the problem of knowing beforehand the individual optimal values, this technique does not guarantee Pareto-optimality even if Equation 5.2 is successfully minimised.

An alternative method is to adapt the selection mechanism, as in the VEGA program [516]. Here, different fractions of the population are selected according to each of the separate objectives. These subpopulations are then merged and shuffled. Crossover and mutation are then applied as is usual. The expected number of offspring of a parent is then the sum of the expected number according to each objective. The overall effective fitness is therefore a linear sum of the objective values, weighted according to the population distribution in that generation. This results in different non-dominated individuals getting different effective fitnesses, which is not what we want if we are looking for Pareto-optimality.

Other methods have used the idea of Pareto-ranking [212, 535]. The Pareto-rank of an individual is defined to be the number of members of the population by which it is dominated. The idea is then to seek individuals with minimum Pareto-rank.

One way of trying to get as many non-dominated solutions as possible is to use sharing [217, 191]. This makes it unfavourable for a GA to generate individuals which are too similar. The end result is a population divided into niches of different solutions. To do this, you need to define what is meant by a niche. We used a method that combined a Hamming distance measure (that is, genotypic difference) with a measure of fitness distance (phenotypic difference). If $d_1(x_i, x_j)$ is the Hamming distance between any two population members x_i and x_j, and $d_2(x_i, x_j)$ is a measure of fitness distance then a sharing function can be defined as follows:

$$s(x_i, x_j) = \begin{cases} 1 - \frac{d_1(x_i,x_j)}{\sigma_1} & \text{if } d_1(x_i, x_j) < \sigma_1, d_2(x_i, x_j) \geq \sigma_2 \\ 1 - \frac{d_2(x_i,x_j)}{\sigma_2} & \text{if } d_1(x_i, x_j) \geq \sigma_1, d_2(x_i, x_j) < \sigma_2 \\ 1 - \frac{d_1(x_i,x_j) d_2(x_i,x_j)}{\sigma_1 \sigma_2} & \text{if } d_1(x_i, x_j) < \sigma_1, d_2(x_i, x_j) < \sigma_2 \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

where σ_1 and σ_2 are *niche radii*, that is, the biggest distances (for genotype and phenotype respectively) at which we will consider two members to be in the same niche. We then scale the fitness of each population member by first summing all the share values of each member with respect to every other, and then dividing the fitness by that total. That is:

$$f'(x_i) = \frac{f(x_i)}{\sum_{j=1}^{M} s(x_i, x_j)} \tag{5.4}$$

where M is the population size. In this way, members which are similar to a number of others in the population (in the sense defined by the metric) will have their fitness reduced. This leads to a population of distinct individuals.

The sharing method can be applied to multiobjective problems by choosing a suitable phenotypic distance metric. Unfortunately, the Pareto-rank is not suitable, as two population members with the same Pareto-rank may have different phenotypes. We therefore chose the following function:

$$d_2(x_i, x_j) = \sum_{k=1}^{z} |f_k(x_i) - f_k(x_j)| \tag{5.5}$$

which sums the fitness distances along each dimension.

5.4 Parallel (diffusion) GAs

The principle feature of a diffusion genetic algorithm is that the population is spatially distributed. Typically, each member of the population resides at its own vertex on a square lattice. The next generation is found in one time step as follows. Each member chooses an immediate neighbour at random and crosses over to produce an offspring. The offspring replaces the original member if it is fitter. Mutation can be applied before or after replacement. A typical algorithm is as follows:

1. Generate a random population $x = (x_1, \ldots, x_M)$ and place on lattice
2. For each member x_i:
   (a) choose a random neighbour x_j of x_i
   (b) cross x_j and x_i to produce an offspring y_i
3. For each member x_i:
   (a) if y_i is better than x_i then let $x_i = y_i$
   (b) apply mutation to x_i
4. Go to 2.

Note that all members of the population act simultaneously. Obviously, this is ideally suited for implementation on certain parallel processor architectures.

Distributing the population in this way, so that information is only exchanged locally (via crossover), has a number of effects. One is that allelic diversity is maintained much more than in an ordinary (panmictic) GA [158]. This gives rise to a second effect: niching. Different areas of the population will tend to converge to different optima. It therefore seems interesting to use diffusion GAs on multiobjective functions in the hope that many different non-dominated solutions will be found within the single population.

Another property of the diffusion GA strategy is that it depends purely on a local comparison between population members. For a Pareto ranking method this is ideal. Rather than calculate the Pareto rank of every member by comparing it to every other member, we need only see if the generated offspring dominates the original parent. Thus the overhead of calculating the Pareto rank is only linear in the population size (as opposed to quadratic in the panmictic case).

5.5 The sensitivity analysis problem

In many situations, in addition to finding an optimal solution, one is interested in the robustness of the chosen solution. Indeed, it may be preferable to have a robust near-optimal solution, than an optimal solution which is unstable. Examples come from such fields as control systems, in which we require a control regime which is robust in the face of small system changes. Another example is in financial management. Given an investment strategy, it is desirable that the expected profits (for example) will not vary drastically with small changes in, say, interest rates.

A first approach to this kind of sensitivity analysis would be to look at single-stuck-at faults (SSAs). In these, one assumes that a single fault has occurred and remains constant. The effects of each possible fault can then be measured. In the financial investment example, this might correspond to looking at this different factors that could vary in turn (budget, interest rates, etc.) and consider the effects of each of these on the chosen investment strategy.

The consideration of SSAs will find the sensitivity of the chosen solution to each factor individually, and will do so in a time linear in the number of factors to be considered. However, one might also be interested in the effects of combinations of factors. For example, a given investment strategy might be perfectly robust with respect to every factor considered individually, but a particular combination of two or more factors (for example, a change in budget and a change in market share) might severely affect it. But to search all possible combinations is clearly exponential in the number of factors to be considered.

To address this problem, we first convert it into a multiobjective problem, with two objective functions. The first is the absolute value of the degree to which the proposed changes will affect the solution's quality. This is to be maximised. The second objective is simply the number of factors that are being considered. This is to be minimised (or, equivalently, the number of factors not considered is to be maximised).

More formally, we suppose that there is a real-valued function $Q(s,p)$ that measures the overall quality of a solution, which depends on a set of choices, or *strategy* s and a set of parameters p. Using some method (possibly a genetic algorithm), a particular strategy has been chosen. We wish to find out how robust this strategy is and, in particular, to which parameters it is most sensitive. Let C be the set of changes to the parameter values under consideration. Then any $c \subseteq C$ can be interpreted as a function that maps one set of parameter values, p, on to a modified set $c(p)$. Our two objective functions are then

$$f_1(c) = |Q(s,p) - Q(s,c(p))| \tag{5.6}$$

$$f_2(c) = \#c \tag{5.7}$$

where $\#$ indicates the cardinality of a set. The set of Pareto-optimal solutions then comprises the maximum values obtainable for for each possible number of faults. So we should find the most sensitive single fault, the most sensitive pair, and so on.

We can represent $c \subseteq C$ naturally as a binary string, indicating whether or not each particular parameter change is to be considered. When more than one possible change to a parameter is considered, a higher string cardinality can be used.

In our tests, we have used functions of the following form:

$$Q(s,p) = \sum_{i=1}^{n} p_i s_i (1 - s_i) \tag{5.8}$$

where $0 < s_i < 1$, $3 < p_i < 4$ and $s = (s_1, \ldots, s_n)$, $p = (p_1, \ldots, p_n)$ were chosen randomly. Note that this is the sum of logistic functions each within the chaotic regime. It should, therefore, provide some interesting sensitivities. We consider changes to the parameters as increases or decreases of 10 %. A set of changes was represented as a string from $\{-1, 0, +1\}^n$, where -1 represents a decrease in value, $+1$ and increase and 0 no change.

5.6 Experimental results

We ran a number of variants of the diffusion and sharing genetic algorithms on problems of the type already described. A number of factors are of interest. Firstly, we want to find the *largest* values of f_1 for each cardinality. Second, we want all possible cardinalities to be investigated. Thirdly we are interested in how long it takes to do this, both in terms of the number of function evaluations

6	4	8	9	9	6	8	8	6	6
5	7	9	10	8	8	8	8	8	8
8	7	8	7	7	8	9	11	9	8
7	7	7	6	11	7	9	7	9	9
7	10	9	6	10	11	10	8	7	7
8	8	8	5	10	13	9	8	10	9
10	8	8	5	4	5	6	9	8	8
7	8	7	7	6	6	6	7	7	8
8	7	9	5	6	8	7	7	7	9
9	9	11	9	4	10	8	9	8	9

Figure 5.2: Diffusion population after 150 generations (number of 1s indicated) on a 10×10 torus topology. Mutation was applied before replacement.

9	7	12	15	13	9	8	13	12	10
6	10	11	9	9	12	10	13	12	4
13	9	12	9	11	12	12	11	9	6
10	10	10	9	11	8	10	10	8	11
9	8	8	10	9	10	9	9	8	8
10	12	8	10	10	8	9	7	8	8
10	9	11	8	10	12	11	10	8	7
10	10	7	7	9	13	9	9	8	9
8	9	7	9	9	12	13	7	11	8
6	12	9	8	11	10	11	7	11	8

Figure 5.3: Diffusion population after 150 generations (number of 1s indicated) on a 10×10 torus topology. Mutation was applied after replacement.

and also clock time. The diffusion GA should have a noticeably faster clock time due to its more efficient way of using dominance. It only compares offspring with parents rather than calculating the Pareto-rank for the whole population.

Our results indicate that not only is the diffusion GA faster in clock time, but it also requires fewer generations to converge. This effect is strongest when uniform crossover is used. We also see better solutions coming from the diffusion GA both in terms of f_1 values and in the spread of cardinalities.

Typical final populations for the diffusion GA are shown in Figure 5.2 (mutation applied before replacement) and Figure 5.3 (mutation applied after replacement). The effect of mutating after replacement is a much wider coverage of the possible cardinalities (maximum 20), although the niches are less stable. We have only indicated the cardinality of each population member to illustrate where the niches are.

To compare these two diffusion methods with a panmictic sharing GA, we ran

Method	time[s]
Panmictic GA with Sharing	23.45
Diffusion GA (mutation before replacement)	6.86
Diffusion GA (mutation after replacement)	9.29

Table 5.1: Clock times for 150 generations.

each 20 times (at 150 generations each) and recorded the best solutions found for each possible cardinality. The results are displayed in Figure 5.4. The results clearly show that the diffusion GA is finding better solutions in general. This is in part due to the fact that the diffusion GA, especially with mutation after replacement, finds a better spread of results in each run.

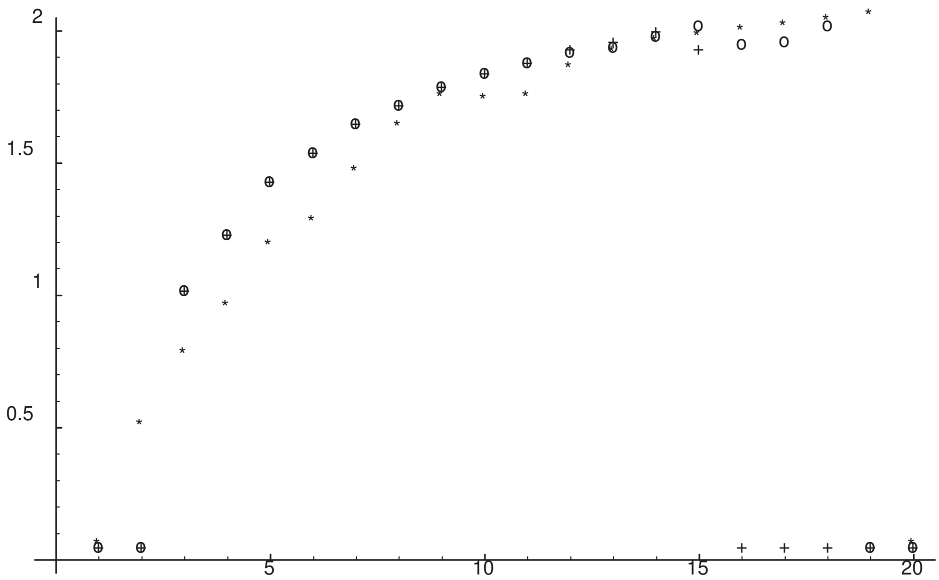

Figure 5.4: The best solutions found for each possible number of 1s in 20 runs. * sharing, + mutuation before, o mutation after

It is also of interest that the diffusion GA had typically found its best solutions within 100 generations. The sharing algorithm needed 150 generations to settle down. Moreover, the clock time for the diffusion GA was much better, as expected. The average run times are displayed in Table 5.1.

5.7 Conclusions

We have investigated the use of parallel diffusion GAs in multiobjective optimisation. They have a number of advantages. Firstly, they have a natural niching behaviour which allows multiple Pareto-optimal solutions to evolve. Secondly, they are much faster than methods requiring the whole population to be Pareto-ranked. Thirdly, they seem to produce a better spread and higher quality of results (at least for the class of problems investigated here).

One potential disadvantage comes from the fact that some areas of the population are wasted. If two competing solutions within the same niche (for example, with the same cardinality) are physically separated in the population, they might never come to compete, and both stabilise in the final population.

We have also presented a method of adapting sensitivity analysis problems to multiobjective ones, allowing this class of problem to be tackled by GAs. A recent paper [549] has conjectured that, if the original decision had been taken by a GA, then it would naturally be robust. It would prove an interesting future project to test this conjecture using our sensitivity analysis approach.

5.8 Bibliography

See chapter bibliography at the end of this proceedings.