Genetic Algorithm Optimization of Multi-Peak Problems: Studies in Convergence and Robustness

Article *in* Artificial Intelligence in Engineering · January 1995

DOI: 10.1016/0954-1810(95)95751-Q · Source: DBLP

1 author:

Andy J. Keane
University of Southampton
293 PUBLICATIONS **6,028** CITATIONS

ELSEVIER

Artificial Intelligence in Engineering **9** (1995) 75–83

0954-1810/95/$09.50

Genetic algorithm optimization of multi-peak problems: studies in convergence and robustness

A. J. Keane

Department of Engineering Science, University of Oxford, Parks Road, Oxford, UK, OX1 3PJ

Engineering design studies can often be cast in terms of optimization problems. However, for such an approach to be worthwhile, designers must be content that the optimization techniques employed are fast, accurate and robust. This paper describes recent studies of convergence and robustness problems found when applying genetic algorithms (GAs) to the constrained, multi-peak optimization problems often found in design. It poses a two-dimensional test problem which exhibits a number of features designed to cause difficulties with standard GAs and other optimizers. The application of the GA to this problem is then posed as a further, essentially recursive problem, where the control parameters of the GA must be chosen to give good performance on the test problem over a number of optimization attempts. This overarching problem is dealt with both by the GA and also by the technique of simulated annealing. It is shown that, with the appropriate choice of control parameters, sophisticated niche forming techniques can significantly improve the speed and performance of the GA for the original problem when combined with the simple rejection strategy commonly employed for handling constraints. More importantly, however, it also shows that more sophisticated multi-pass, constraint penalty functions, culled from the literature of classical optimization theory, can render such methods redundant, yielding good performance with traditional GA methods.

Key words: design optimization, genetic algorithm, constraint, cluster.

1 INTRODUCTION

When an engineer is faced with the problem of producing a new design the first stage in the process commonly consists of considering the number of competing options. Usually, there is a requirement to explore the effects on the design of changes in a number of key parameters, often using computerised design facilities. When more than two or three parameters are to be considered, their interactions can be very hard to predict and the design task becomes difficult. Such studies can often be cast in terms of an optimization problem, where some measure of merit or objective function is maximized or minimized by altering the design parameters while meeting various constraints. Methods for solving the general optimization problem have been studied for many years and there is a considerable literature (see for example Ref. 1). None the less, design often involves awkward problems that have highly non-linear relationships and many constraints that traditional methods find hard to cope with.

Genetic algorithms have received much attention over the last ten years as providing a mechanism for dealing with such difficult optimization problems. They are known to be good at handling cases where the objective function of the problem is characterized by a number of sharp peaks, which lead classical, slope driven methods to terminate at false optima. A comprehensive survey of this work is provided in Ref. 2. However, there is relatively less work in the literature dealing with the constraint surfaces that are commonly found in engineering design problems. Moreover, it is often the case that such constraints define the true global optimum solution to a design problem. They typically arise from material limitations, such as maximum stress or temperature levels. The calculation of these quantities is often complex and the designer has little a priori knowledge of how they will limit the choice of the free variables in the problem under consideration. This paper is concerned with the application of optimizers, and the GA in particular, to such constrained, multi-peaked function optimization.

Genetic algorithms, along with many other techniques, have a number of difficulties when dealing with this kind of task; the two most severe appear to be:

(1) convergence to sub-optimal solutions;
(2) robustness of the convergence for repeated attempts using different random number sequences within the routines.

In an attempt to overcome these problems, a programme of work has been carried out to choose the best set of optimizer control parameters for a given test problem. The choice of control parameters forms one of the major difficulties in using optimization software: ideally an optimizer should require few such controls and, more importantly, the results, in terms of accuracy and speed, should be relatively insensitive to this choice. It is usually the case that, in any given problem, one particular optimizer can be tuned to give startlingly good performance; it is much more difficult to choose a method and select its control parameters so as to get good results at the first attempt. Since optimization is only really difficult where computation times for achieving an optimum are long, no *real* problem allows the luxury of such fine tuning. None the less, researchers working on optimizers must consider tuning aspects so they can assess competing methods. When an optimizer has many inter-related parameters, this forms an optimization task of its own, i.e. to choose a set of control parameters that will, on average, give good results in the minimum time when applied to a particular problem. This is the task considered in this work. Here the choice has been carried out using two methods to optimize the GA parameters for the underlying problem: the GA itself and simulated annealing (SA).[3]

2 THE GENETIC ALGORITHM

The GA used here is fairly typical of those discussed in Ref. 2. Such methods work by maintaining a pool or population of competing designs which are combined to find improved solutions. In their basic form, each member of the population is represented by a binary string that encodes the variables characterizing the design. The search progresses by manipulating the strings in the pool to provide new generations of designs, hopefully with better properties on average than their predecessors. The processes that are used to seek these improved designs are set up to mimic those of natural selection: hence the method's name. The most commonly used operations are currently:

(1) Selection according to fitness, i.e. the most promising designs are given a bigger share of the next generation.
(2) Crossover, where portions of two good designs, chosen at random, are used to form a new design, i.e. two parents 'breed' an 'offspring'.
(3) Inversion, where the genetic encoding of a design is modified so that subsequent crossover operations affect different aspects of the design.
(4) Mutation, where small but random changes are arbitrarily introduced into a design.

In addition, the number of generations and their sizes must be chosen, as must a method for dealing with constraints (usually by application of a penalty function).

The algorithm used here works with 16 bit binary encoding (although parameters that are selected from a number of fixed possibilities use only the minimum necessary number of bits). It uses an elitist survival strategy which ensures that the best of each generation always enters the next generation, and has optional niche forming to prevent dominance by a few moderately successful designs preventing wide-ranging searches. Two penalty functions are available. The main parameters used to control the method may be summarized as:

N_{gen}, the number of generations allowed (default 10);
N_{pop}, the population size or number of trials used per generation, which is therefore inversely related to the number of generations given a fixed number of trials in total (default 100);
P[best], the proportion of the population that survive to the next generation (default 0·8);
P[cross], the proportion of surviving population that are allowed to breed (default 0·8);
P[invert], the proportion of the surviving population that have their genetic material re-ordered (default 0·5);
P[mutation], the proportion of the new generation's genetic material that is randomly changed (default 0·005);
A proportionality flag, which selects whether the new generation is biased in favour of the most successful members of the previoius generation or alternatively if all P[best] survivors are propagated equally (default TRUE);
The penalty function choice.

When using the GA to explore large design spaces with many variables, it has also been found that the method must be prevented from being dominated by a few moderately good designs that prevent further innovation. A number of methods have been proposed to deal with this problem; that used here is based on MacQueen's adaptive KMEAN algorithm,[4] which has recently been applied with some success to multi-peak problems.[5] This algorithm subdivides the population into clusters that have similar properties. The members of each cluster are then penalized according to how many members the cluster has and how far it lies from the cluster centre. It also, optionally, restricts the crossover process that forms the heart of the GA, so that large successful clusters mix solely with themselves. This aids convergence of the method, since radical new

ideas are prevented from contaminating such sub-pools. The version of the algorithm used here is controlled by:

- D_{min}, the minimum non-dimensional Euclidean distance between cluster centres, with clusters closer than this being collapsed (default 0·1);
- D_{max}, the maximum non-dimensional Euclidean radius of a cluster, beyond which clusters subdivide (default 0·2);
- N_{clust}, the initial number of clusters into which a generation is divided (default 25);
- N_{breed}, the minimum number of members in a cluster before exclusive inbreeding within the cluster takes place (default 5);
- α, the penalizing index for cluster members, which determines how severely members sharing an overcrowded niche will suffer, with small numbers giving a less penalty (default 0·5), i.e. the objective functions of members of a cluster of m solutions are scaled by $m[1-(E/2D_{max})^{\alpha}]$, where E is the Euclidean distance of the member from its cluster centre (which is always less than D_{max}).

In this form the GA has 13 control parameters that the user may alter before applying the method.

3 PENALTY FUNCTIONS

As has already been mentioned, in most work on GAs, constraints are dealt with by the use of penalty functions: such functions are used to distort the objective function in order to force the search towards feasibility, when a constraint is, or is about to be, violated. Typically, a one pass external function is used so that, whenever a design violates a constraint, a large penalty is applied, effectively ruling it out of further consideration. This is very severe and sometimes the search stalls, particularly if it must follow along a constraint line to reach the optimum. Despite this disadvantage, the strategy is often quite successful, and requires a minimum of computer time when it works.

Other penalty functions soften this severe distortion and to try to achieve less risk of premature stalling of the search. One such is the Fiacco and McCormick function which was originally conceived for use with multiple passes of classical, slope driven optimizers. In this case the function is set up to fit naturally within the GA by being applied with increasing severity to each new generation. It modifies the objective function to be maximized, f, as follows:

$$f' = f - \sum_i \frac{(\phi_i^{\mathrm{v}})^2}{p^{\max(1,n/N_{\mathrm{pop}}-2)}} - \sum_i \frac{p^{2.\max(1,n/N_{\mathrm{pop}}-2)}}{\phi_i^{\mathrm{s}}} \tag{1}$$

Here ϕ_i^{v} are the violated constraint conditions and ϕ_i^{s} those that are satisfied (in both cases being normalized so that violated constraints are negative and satisfied ones positive). p is a penalty quantity, less than one (default 0·5), chosen to suit the problem in hand, n the current function evaluation number, and N_{pop} the number of members in each generation. Thus, to begin with, when n/N_{pop} is small, violated constraints are only slightly penalized along with satisfied ones that are near the constraint boundaries, effectively warning the optimizer of the presence of boundaries while allowing their exploration. As the optimization proceeds and the number of evaluations increases, n/N_{pop} grows, causing $p^{\max(1,n/N_{pop}-2)}$ to become exponentially small, severely penalizing the violated constraints and removing the effect of the term involving the satisfied ones, causing the optimizer to chose feasible solutions where f' reverts to f. Notice that here the division n/N_{pop} is carried out in integer arithmetic, causing this quantity to be constant for members of the same generation. By also subtracting two, but preventing the result from being less than unity, the penalty is deliberately held low during the first three generations of the method, giving the GA chance to get started before the penalizing function takes effect (see Siddall[1] for further details of the original usage of this function). The use of an evolving constraint penalty fits naturally within the general scheme of the GA yet does not seem to have been reported before, perhaps because more applications of the GA to date have been to unconstrained problems. Certainly, the Fiacco and McCormick function has been widely discussed in the literature of classical optimization methods.

4 A BUMPY EQUATION

To simulate a multi-peak optimization problem the following simple objective function can be defined:

$$f(x,y) = \frac{\sin^2(x-y)\sin^2(x+y)}{\sqrt{x^2+2y^2}} \tag{2}$$

This function produces a series of peaks that get smaller with distance from the origin and are nearly symmetrical about $y = x$. The optimization problem is then defined as finding x, y in the range $0 \leq x, y \leq 10$, starting from the point (5, 5) to maximize the function $f(x,y)$ subject to $x + y \leq 15$ and $xy \geq 0{\cdot}75$ (see Fig. 1). This problem has a number of features that are designed to make optimization difficult:

(1) The surface is nearly, but not quite, symmetrical in $x = y$, so that peaks always occur in pairs but with one always bigger than its sibling.
(2) The true maximum is 0·365 at (1·593, 0·471), which is defined by a constraint boundary.
(3) There is another similar peak of height 0·274 at (0·475, 1·578).

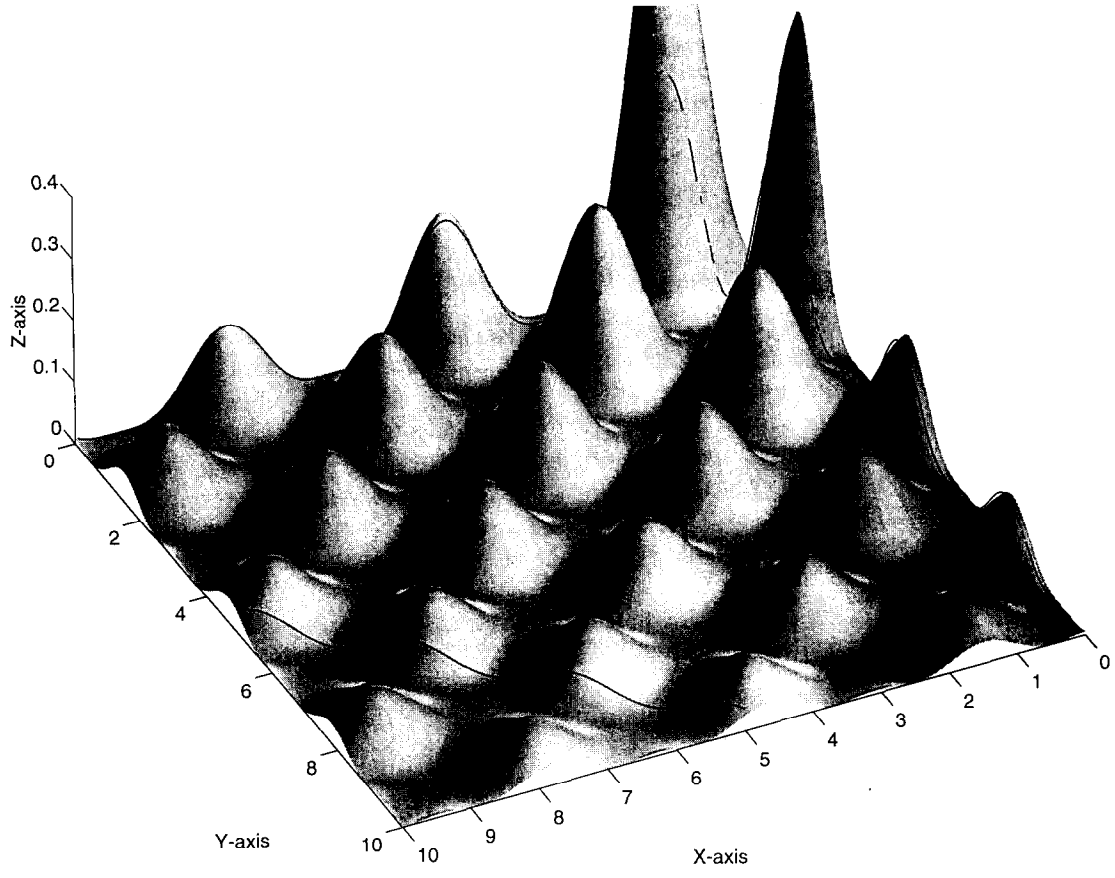

Fig. 1. The surface generated by eqn (2) with the constraints marked.

(4) The major maximum within the boundaries has a height of 0·263 at (3·087, 1·517), which is thus quite competitive with solutions on the slopes of the true, constrained maximal peak (where the slope is much steeper and the base area much smaller), causing many solutions to be trapped on this sub-optimal peak.
(5) The starting point lies on the line $x = y$ where the function is always zero, so that explorations carried out by making small equal changes in x and y show the function as invariant (affecting many heuristic methods).

The problem is, however, dependent on only two variables, which aids display of the function and speed of calculation, but somewhat limits the generality of the results. None the less, it is an improvement over the one-dimensional functions used for many studies in this field.

5 INITIAL OPTIMIZATION

To gain some initial insights into this problem, four different optimization methods were deployed using the appropriate default control parameters. These were:

(1) The GA.
(2) Simulated annealing (SA), using ten temperatures. The simulated annealing approach is based on the kinetics of freezing crystals, where it is observed that minimum energy states are reached for sufficiently slow cooling. Essentially, the method makes random small changes to the design and these are accepted if they improve it and occasionally even if they worsen it. The likelihood of changes that worsen the solution being accepted is controlled by a Boltzman probability function which is dependent on the so-called annealing temperature. The method is often cited as being an alternative to the GA.
(3) Repeated sequences of linear approximation followed by simplex solutions (see Siddall's program, APPROX[1]). APPROX uses small explorations in the vicinity of the current design to establish the local slope. The problem is then assumed to be linear near this point and classical linear programming methods are used to find the best point in this sub-space. The process is then repeated around the new point and so on, with the size of the linear region being reduced at each new base point. This method is very rapid for smoothly varying problems with few variables but tends to find local rather than global solutions.
(4) The Hooke and Jeeves heuristic search followed by a local random search (see Siddall's program

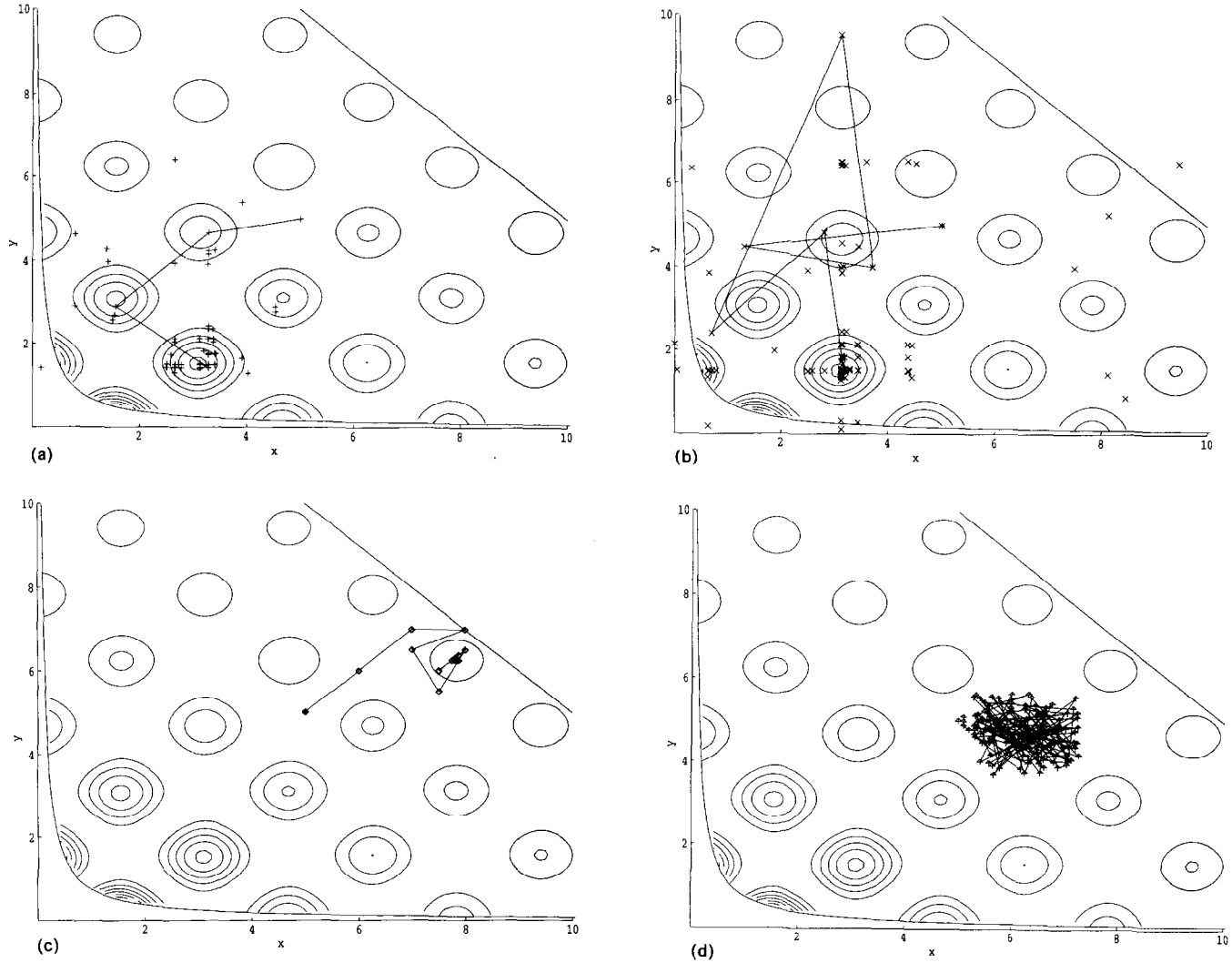

Fig. 2. The optimization paths taken by the four methods mentioned in section 5: (a) GA; (b) SA; (c) APPROX; and (d) SEEK.

SEEK[1]). SEEK is a classical heuristic search that wanders around the design space following a preprogrammed pattern of moves, seeking the steepest ascent to the maximum. When used with a one pass penalty function it also uses random point generation to test its final solution for better points in the vicinity of the current design. When such an improvement is found this is used to restart the search.

In all cases 1000 function evaluations was set as the suggested maximum and a simple one pass external penalty function used to deal with constraint violations. As has already been noted, this penalty function causes all solutions that do not meet the constraint requirements to be considered as totally unworkable and consequently assigns them a very low, -10^{20}, objective function value. The resulting optimization traces are shown in Fig. 2. Here, only the best generation/annealing temperatures of the GA and SA are shown, together with all points from their final generation/annealing temperature. The linear approximation and Hooke and Jeeves methods used many fewer than 1000 points since they rapidly converged to the wrong peaks. Figure 3 shows all the points evaluated by the GA with these default settings and this demonstrates the considerable coverage achieved by the method.

6 THE RECURSIVE PROBLEM

Given the previously stated optimization task and GA, the recursive problem was set as selecting the best values for the 13 control parameters of the GA. To allow for the effects of different random number sequences on the behaviour of the algorithm the results were averaged over five different runs of the task, with slight weighting in favour of combinations that worked with the minimum number of function evaluations. Thus, the new objective function, here called Measure(5), becomes

$$\frac{1}{5}\sum_{i=1}^{5}\frac{f_i^{\max}(x,y)}{(N_{\text{gen}}N_{\text{pop}}/1000)^{0.15}} \tag{3}$$

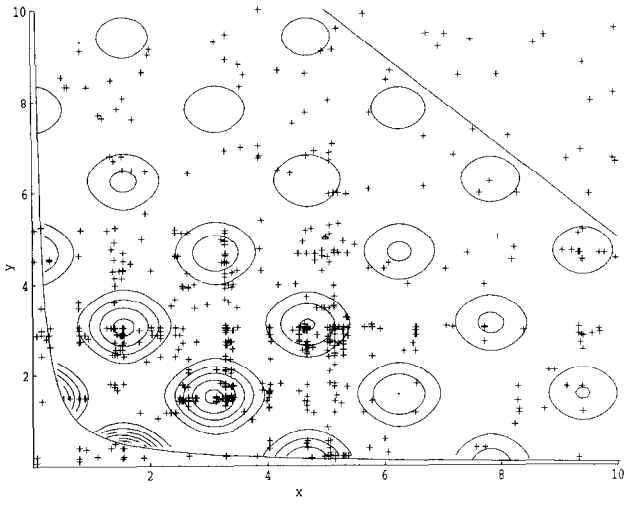

Fig. 3. All the points evaluated by the GA with default parameters.

where i indicates the run number. Thus, sequences using less than 1000 function evaluations (the default of 10 generations, each of 100 members) benefit while those using more are penalized. The index of 0·15 biases accurate solutions in favour of very rapid ones, e.g. if only 500 evaluations are used the average solution value needs to be at least $(500/1000)^{0.15} = 90\%$ as good to be preferred, despite being twice as fast as the default. This new problem is much more demanding than the original for a number of reasons:

(1) It has 13 rather than 2 variables.
(2) Each function evaluation takes, obviously, around 1000 times longer to compute.
(3) It is much more dramatically non-linear, with certain parameter choices acting as on/off switches.
(4) Being essentially abstract, there is little obvious pattern to its behaviour.

It is, however, an unconstrained problem, which obviates the need to consider penalty functions.

6.1 Niche forming parameters

Initially, the focus was placed on the subset of parameters controlling the niche forming behaviour as it was expected that these would be crucial to performance on multi-peak problems of the sort considered here. This gives a problem with seven variables, including turning the niche mechanism off and changing N_{gen} or N_{pop}. Both the GA and SA were used to optimize this recursive problem, in each case with 500 function evaluations (i.e. 25 000 runs of the original optimization problem leading to typically 25 000 000 evaluations of the original function — indicating why such studies are only possible on test functions of the type considered here). The GA used 50 generations of 100 points and the SA 17 temperatures of

Table 1. Optimization results, niche parameters only

Parameter	Default	GA(7)	SA(7)
N_{gen}	10	3	2
N_{pop}	100	96	96
P[best]	0·8	0·8	0·8
P[cross]	0·8	0·8	0·8
P[invert]	0·5	0·5	0·5
P[mutation]	0·005	0·005	0·005
Proportionality flag	T	T	T
Penalty	1-Pass	1-Pass	1-Pass
D_{min}	0·1	0·543	0·471
D_{max}	0·2	0·953	0·909
N_{clust}	25	55	23
N_{breed}	5	23	41
α	0·5	4	4
Measure(5)	0·291	0·368	0·360
Measure(50)	0·289	0·286	0·271
%Peak A	50	32	28
%Peak D	40	48	40

294 points. The final results were further checked by averaging the results over 50 optimization runs of the underlying problem, leading to Measure(50).

Table 1 gives the results of this analysis together with those obtained using the default settings. In this table Peak A refers to the true optimal peak at (1·593, 0·471) and Peak D to the largest peak fully within the constraint boundaries at (3·087, 1·517). Thus %Peak A indicates the percentage of runs that resulted in the final solution lying on Peak A when 50 trials were carried out, i.e. in 25 out of 50 cases for the default GA. Figures 4–6 illustrate the three combinations of parameters in the table, showing all 50 optimization traces used when calculating Measure(5) in each case (plotting the best generation points only). The following observations may be made about these results:

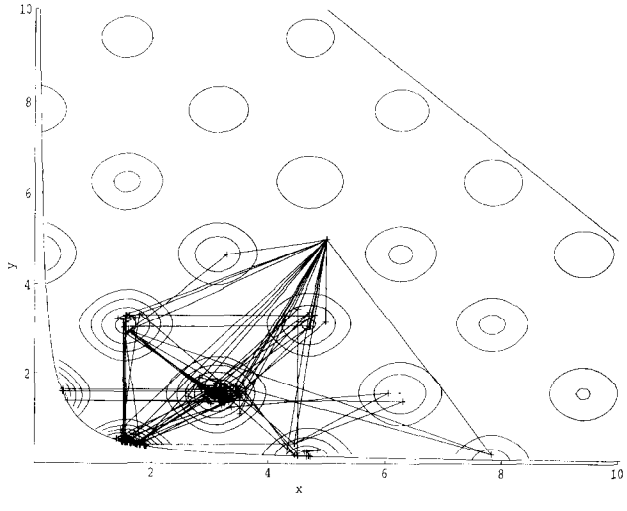

Fig. 4. The fifty optimization paths followed when evaluating Measure(50) for the GA with default control parameters.

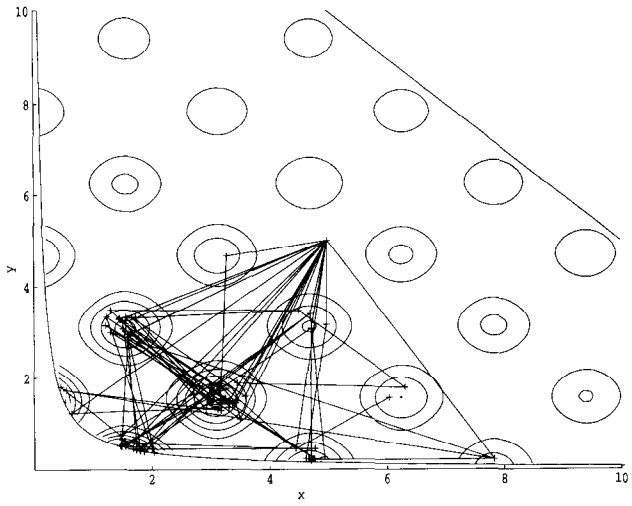

Fig. 5. The fifty optimization paths followed when evaluating Measure(50) for the GA with niche control parameters optimized by the GA.

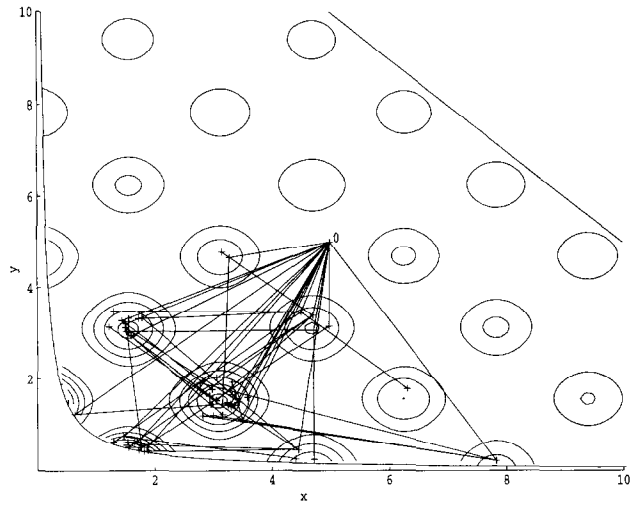

Fig. 6. The fifty optimization paths followed when evaluating Measure(50) for the GA with niche control parameters optimized by the SA.

(1) The default parameters are more robust than those tuned on just five optimization runs.
(2) The tuned runs work 3–5 times faster than the original set.
(3) The GA outperforms the SA on this recursive problem.
(4) Both methods suggest that cluster penalties must be large (high α), with small unfocused clusters, for the KMEAN algorithm to work well; this is probably related to the large number of peaks in the underlying test function.

It should be noted that, since both the parameter sets produced here use many fewer function evaluations than the default GA, both Measure(5) and Measure(50) benefit from the weighting detailed in eqn (3); if this is removed these measures are reduced by around 20%.

Thus far, the default approach gives a good account of itself, particularly with respect to accuracy and robustness, with the GA exhibiting some of the features mentioned earlier as being desirable.

6.2 All parameters

Next, all the control parameters in the GA were allowed to vary, leading to the results given in Table 2. Figures 7 and 8 illustrate the last two combinations of parameters in the table, again showing all 50 optimization traces used when calculating Measure(50) in each case. The following observations may be made about this second set of results:

(1) The default parameter set is no longer as robust as the tuned set.

Table 2. Optimization results, all parameters

Parameter	Default	GA(7)	SA(7)	GA(all)	SA(all)
N_{gen}	10	3	2	10	9
N_{pop}	100	96	96	66	94
P[best]	0·8	0·8	0·8	0·09	0·04
P[cross]	0·8	0·8	0·8	0·76	0·51
P[invert]	0·5	0·5	0·5	0·64	0·18
P[mutation]	0·005	0·005	0·005	0·17	0·19
Proportionality flag	T	T	T	T	T
Penalty	1-Pass	1-Pass	1-Pass	F&M	1-Pass
D_{min}	0·1	0.543	0·471	—	0·425
D_{max}	0·2	0·953	0·909	—	0·937
N_{clust}	25	55	23	—	24
N_{breed}	5	23	41	—	43
α	0·5	4	4	—	4
Measure(5)	0·291	0·368	0·360	0·382	0·369
Measure(50)	0·289	0·286	0·271	0·333	0·312
%Peak A	50	32	28	62	58
%Peak D	40	48	40	34	30

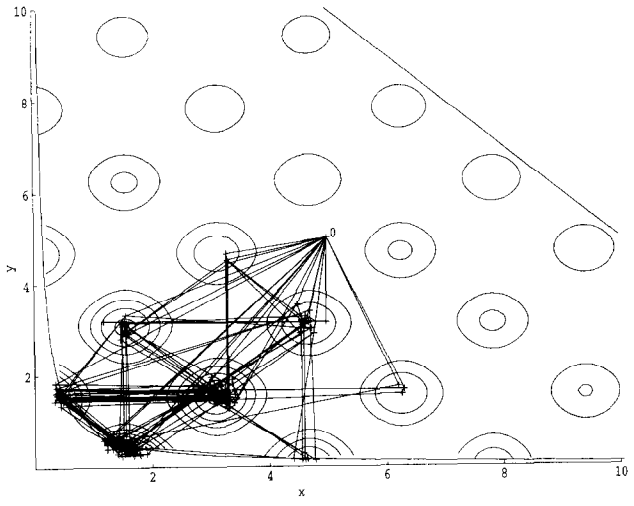

Fig. 7. The fifty optimization paths followed when evaluating Measure(50) for the GA with all parameters optimized by the GA.

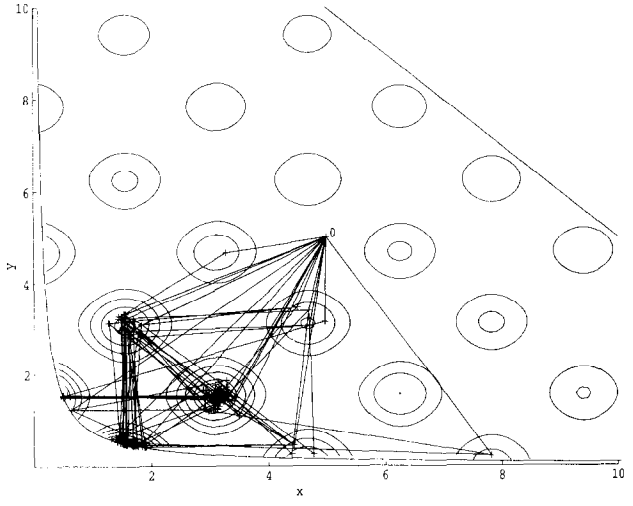

Fig. 8. The fifty optimization paths followed when evaluating Measure(50) for the GA with all parameters optimized by the SA.

(2) The tuned runs are only slightly faster than the default set.
(3) The GA again outperforms the SA in setting the control parameters.
(4) By adopting the Fiacco–McCormick penalty function the GA can reach very good results *without* the clustering algorithm.
(5) Both methods dramatically reduce P[best], which means that very few parents are used to produce each new generation.
(6) Both methods significantly increase the mutation level, leading to greater changes between parents and offspring.

The last two points seem to suggest that a pure random search might work as well as the GA; however, setting P[best] to zero and P[mutation] to 0·5 has this effect and Measure(50) then drops to 0·267, i.e. worse than for all the other approaches. To gain some additional confidence that optimal sets of parameters have been produced in at least some sense, the best generation/annealing temperature results of the overall problem have been plotted in Fig. 9. This demonstrates that the methods used had at least settled by the time that the results given in the tables were taken. Moreover, the highly oscillatory nature of the SA plots at low trial numbers indicates that the annealing schedule used did,

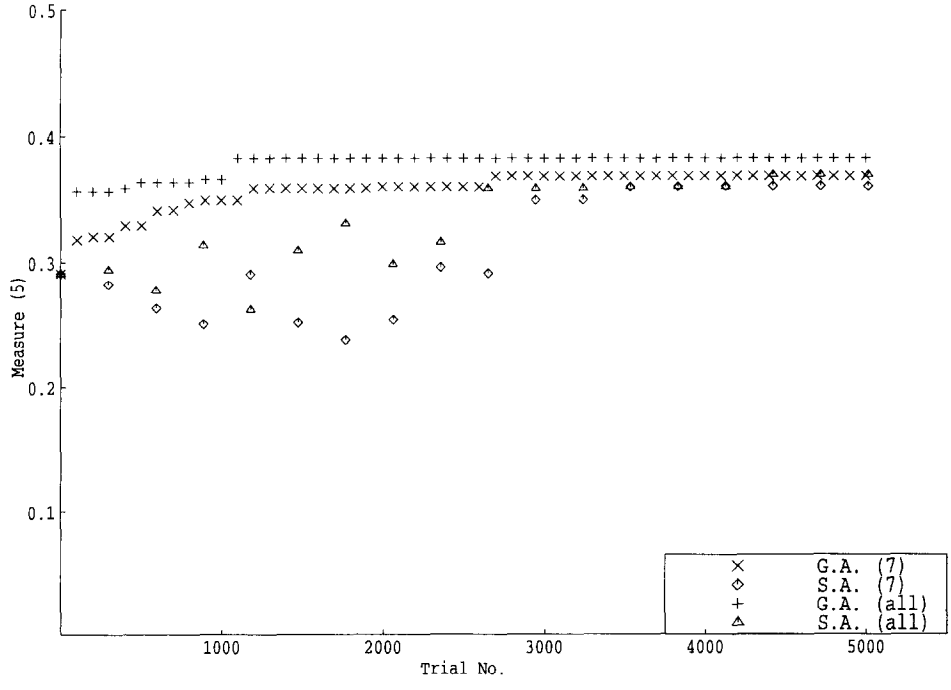

Fig. 9. The best generation/annealing temperature results for the four optimization runs applied to eqn (3).

in fact, start from a high enough temperature, where the solutions had 'melted'. The fact that the two methods converge to different solutions demonstrates, however, how complex this recursive problem is.

7 CONCLUSIONS

The following principal conclusions may be drawn from this brief study into optimizing multi-peak, constraint-limited optimization problems:

(1) GAs are fundamentally good multi-peak optimization routines.
(2) GAs must, none the less, be tried over several different sets of random numbers to guarantee that results are not flukes.
(3) Careful tuning of the niche control parameters may be needed to gain the best performance in these circumstances.
(4) The correct choice of the fundamental parameters may obviate the need for sophisticated niche control mechanisms, particularly if a good constraint mechanism is used and a sufficient number of trials are allowed.

In the context of this last point, the use of constraint penalty functions developed for classical, sequential unconstrained maximization techniques (SUMT) such as the Fiacco–McCormick function may well give significant advantages since they may be smoothly integrated into the GA method by setting increasingly severe constraints on each successive generation. As such they form a natural partner to the GA when dealing with constrained problems and deserve wider attention.

REFERENCES

1. Siddall, J. N. *Optimal Engineering Design: Principles and Applications*. Marcel Dekker, New York, 1982.
2. Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, 1989.
3. Kirkpatrick, S., Gelatt Jr, C. D. & Vecchi, M. P. Optimization by simulated annealing. *Science*, 1983, **220**, 671–80.
4. Anderberg, M. R. *Cluster Analysis for Applications*. Academic Press, 1975.
5. Yin, X. & Germay, N. A fast genetic algorithm with sharing scheme using cluster methods in multimodal function optimization, in *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, ed. R. F. Albrecht, C. R. Reeves & N. C. Steele. Springer-Verlag, Innsbruck, 1993.