A BRIEF COMPARISON OF SOME EVOLUTIONARY OPTIMIZATION METHODS

A.J. Keane (Andy.Keane@eng.ox.ac.uk)

Department of Engineering Science, University of Oxford,
Parks Road, Oxford, OX1 3PJ, U.K.

1. INTRODUCTION

This paper briefly reviews the behaviour of four different evolutionary optimization methods when applied to a pair of difficult, high dimension test functions. The methods considered are the Genetic Algorithm (GA)[1], Evolutionary Programming (EP)[2], Evolution Strategies (ES)[3] and Simulated Annealing (SA)[4]. The two problems considered are the royal road function proposed by Holland, here called 'jhrr'[5] and the fifty dimensional 'bump' problem introduced by Keane[6]. These two problems are very hard for most optimizers to deal with: 'jhrr' is highly discontinuous and very misleading for most methods. Conversely, 'bump' is quite smooth but contains tens of thousands of peaks, all of similar heights. Moreover, its optimal value is defined by the presence of a constraint boundary.

The purpose of the work presented is to see how well the different methods cope with these tasks and, more importantly, to investigate what parameter settings are required for robust performance on these two very different problems. It should be stated at the outset, however, that in terms of the sophistication with which each method is implemented, the GA used is the most complex and to some extent should therefore be expected to give the best results. Nonetheless, the comparisons are representative of the kind of tests typically set for new optimization methods, i.e., comparison with the best available alternatives. Still, the reader should be warned that more sophisticated implementations of the other methods could be expected to improve their overall best performance. Finally, in this introductory session, it must be remembered that for any given problem it is always possible to produce a dedicated optimizer that will work better than any other, more general purpose approach: thus comparisons between methods should be treated with care.

2. THE OPTIMIZATION PROBLEMS

The optimization problems to be tackled here have already been discussed in the literature and so will be only briefly outlined. The 'bump' problem is defined as

$$\text{maximize} \quad \frac{\text{abs}\left(\sum_{i=1}^{n}\cos^4(x_i) - 2\prod_{i=1}^{n}\cos^2(x_i)\right)}{\sqrt{\sum_{i=1}^{n} i x_i^2}}$$

for

$$0 < x_i < 10, \qquad i = 1, \cdots, n$$

subject to

$$\prod_{i=1}^{n} x_i > 0.75 \qquad \text{and} \qquad \sum_{i=1}^{n} x_i < 15n/2$$

starting from

$$x_i = 5, \qquad i = 1, \cdots, n$$

where the x_i are the variables (expressed in radians) and n is the number of dimensions. This function gives a highly bumpy surface (Figure 1 shows the surface for $n = 2$) where the true global optimum is usually defined by the product constraint.

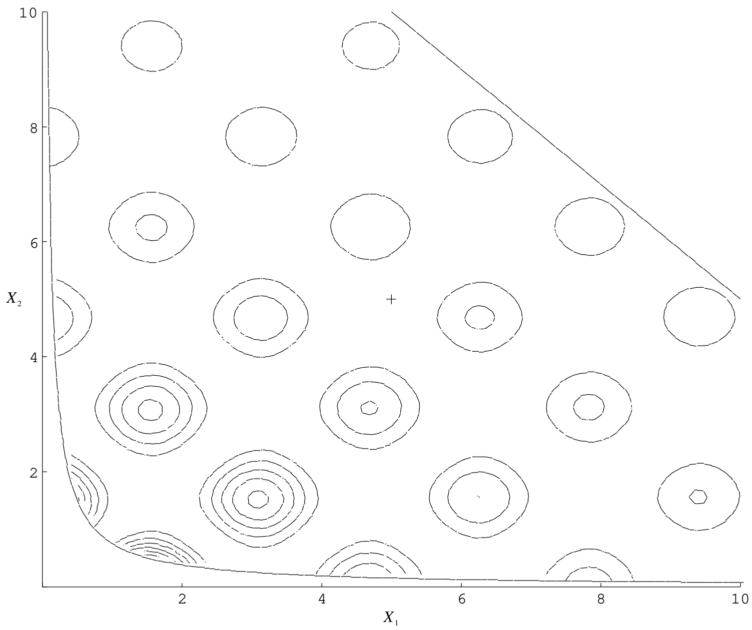

Figure 1 - contour map of the two dimensional 'bump' problem.

The 'jhrr' problem takes a binary string as input and produces a real value which must be maximized. There are no constraints to be satisfied. The string is composed of 2^k non-overlapping contiguous regions, each of length $b+g$. With Holland's defaults, $k=4, b=8, g=7$, there are 16 regions of length 15, giving an overall string length of 240. Each region is divided into two non-overlapping pieces. The first, of length b, is called the block and the second of length g the gap. In the fitness calculation only the bits in the block part of each region are considered. The calculation consists of two steps. The part calculation adds to the block's fitness by v for every 1 it contains up to a limit of m^*. If a block contains more than m^* 1's but less than b 1's it receives $-v$ for each 1 over the limit. The default settings are $v=0.02$ and $m^*=4$, so a block with six 1's is assigned a fitness of $(6-4)\times-0.02=-0.04$. Lastly, if a block consists entirely of 1's it receives nothing from the part calculation but it is then considered complete. This then leads to the bonus calculation which rewards complete blocks. The first complete block receives an additional u^*, default 1.0, and each subsequent block u, default 0.3. Next, adjacent pairs of complete blocks are rewarded in the same way and then four complete blocks in a row and so on until all 16 blocks are complete. This leads to the maximum objective value which is $1.0+(1.0+0.3)+(1.0+3\times0.3)+(1.0+7\times0.3)+(1.0+15\times0.3)=12.8$ The presence of the gap regions, which do not affect the calculation, ensures that there are a considerable number of strings that have this maximal value.

Although the royal road function was designed to be easy for a GA to deal with and hard for other methods, it turns out that even GA's find it difficult to solve[7]. It is therefore useful as an extreme example of a case where the relationship between the variables in the function and its value are extremely non-linear, exhibiting many step-like changes.

3. THE OPTIMIZERS

Optimization problems of the sort discussed here are characterized by having many variables, highly non-linear relationships between the variables, and an objective function that has many peaks and troughs: in short they are difficult to deal with. The search for methods that can cope with such problems has lead to the subject of evolutionary computation. Techniques in this field are characterized by a stochastic approach to the search for improved solutions, guided by some kind of evolutionary control strategy. There are four main methods in use today: (i) genetic algorithms (GA)[1], where the methods of Darwinian evolution are applied to the selection of 'fitter' designs; (ii) evolutionary programming (EP)[2], which is a more heuristic approach to the problem based on ranked mutations; (iii) evolution strategies (ES)[3], where individual designs are mutated using adaptive mutation rates, individually tuned to each variable in the problem and (iv) simulated annealing (SA)[4], where the control strategy is based on an understanding of the kinetics of solidifying crystals and where changes are sometimes allowed even if they make the solution worse. The first three of these methods work on groups of designs called populations while the last deals with only one solution at a time.

The versions of the methods used here are fairly standard and as discussed in the references but with the exceptions detailed below. All the methods are set up to use either a one-pass external penalty function or an evolving Fiacco-McCormick internal and external function that is made more severe as the search progresses[8]. These are, of course, not used on the 'jhrr' problem, which is unconstrained. The methods are also all set up to work with a binary encoding of the real valued design variables using up to 16 bit accuracy (default 12 bit) except the ES which works directly on the real valued quantities. This encoding is quite normal with a GA and also allows a simple method for effectively randomizing the variables for the SA and EP. Such a scheme helps the methods deal with the 'jhrr' problem, as there is then a direct mapping between mutations and function value.

The GA used encompasses a number of new ideas that have proven well suited to engineering design problems[9, 6]. 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. 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 the surviving population that are allowed to breed (default 0.8);

P[invert] the proportion of this 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 previous generation or alternatively if all P[best] survivors are propagated equally (default TRUE) and

the Penalty Function choice.

The niche forming method used here is based on MacQueen's Adaptive KMEAN algorithm[10] which has recently been applied with some success to multi-peak problems[11]. 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) and

α the penalising index for cluster members, which determines how severely members sharing an over-crowded niche will suffer, with small numbers giving less penalty (default 0.2), i.e., the objective functions of members of a cluster of m solutions are scaled by $m^{\min(\alpha,1)}[1-(E/D_{\max})^{\alpha}] + (E/D_{\max})^{\alpha}$, where E is the Euclidean distance of the member from its cluster centre (which is always less than $D_{\max}$).

The EP routine is exactly as per the references, except that, as already noted, it works with a binary encoding and uses a choice of penalty function. It works by forming a given number of random guesses and then attempts to improve on them, maintaining a number of best guesses as the process continues (the population). In this method evolution is carried out by forming a mutated child from each parent in the population where mutation is related to the objective function so that successful ideas are mutated less. The objective functions of the children are then formed and a stochastic ranking process used to select the next parents from the combined set of parents and children. The best of the solutions is kept unchanged at each pass to ensure that only improvements are allowed. The mutation is controlled by a variable which sets the order of the mutation process with ranking. In all cases the best parent is not mutated and almost all bits in the worst are changed. The stochastic process for deciding which elements survive involves jousting each member against a tournament of other members selected at random (including possibly itself) with a score being allocated for the number in the tournament worse than the case being examined. Having scored all members of both parent and child generations the best half are kept to form the next set of parents. The average number of solutions in the tournament is set by a control variable.

The ES routine used allows either the $(\mu+\lambda)$ or the (μ,λ) variants but does not use correlated mutations. It allows for either discrete or intermediate recombination on both design variables and the mutation control vectors, with the mutations being applied to non-dimensionalized versions of the design vectors. The algorithm works by forming a given number of random guesses and then attempts to improve on them, maintaining a number of best guesses as the process continues (the population). Evolution is carried out by forming a child population from mating pairs of parents, followed by mutation. The next generation is then made up from good members of the old and child populations. Two methods can be used: either the best from the combined and child populations are kept $(\mu+\lambda)$ or, alternatively, only the best from the child population are used (μ,λ), with any shortfall being made up from the best of the parents. The mutations are controlled by vectors of standard deviations which also evolve with the various population members. The rate at which these S.D.'s change is controlled by a parameter of the method. Additionally, the breeding of new children can be discrete and then individual design vector values are taken from either parent randomly, or intermediate when the values are the average of those of the parents. A similar interchange takes place between the vectors of S.D.'s of the parent designs. The best of the solutions is remembered at each pass to ensure that the final result is the best solution of all.

The SA used has a logarithmic cooling schedule which is based on the following strategy:

(1) choose the total number of trials N;

(2) set $N_T = N^P$

(3) then for $i=1$ to N_T let $T_i = W^{((N_T/C)-i)}$ and do N/N_T tests at this temperature.

Here there are three parameters: P, W and C: P lies between 0 and 1 with 0 implying a random search at one temperature and 1 a single trial at each temperature. W is set to lie in the range 1-10 and C 0.1-10, thus large values of W give a wide range of temperatures, while large values of C bias this range to lower, colder values (W of 1.0 fixes all temperatures to be the same while C of 2.0 biases the temperatures equally between hot and cold). Here the default values are 0.3333, 5.0 and 2.0 respectively.

Finally the 'Boltzman' test is applied as follows:

(1) always keep a better result;

(2) keep a worse result if $\exp(-(U_{\mathrm{new}}-U_{\mathrm{old}})/(\mathrm{d}U_{\mathrm{first}}T_i)) > R$ where U_{old} is the previous objective function value, U_{new} is the new (worse) value, R is a random number between 0 and 1 and $\mathrm{d}U_{\mathrm{first}}$ is the magnitude of the difference between the objective function of the starting point in the search and the first trial at the highest temperature (and takes the place of the Boltzman constant).

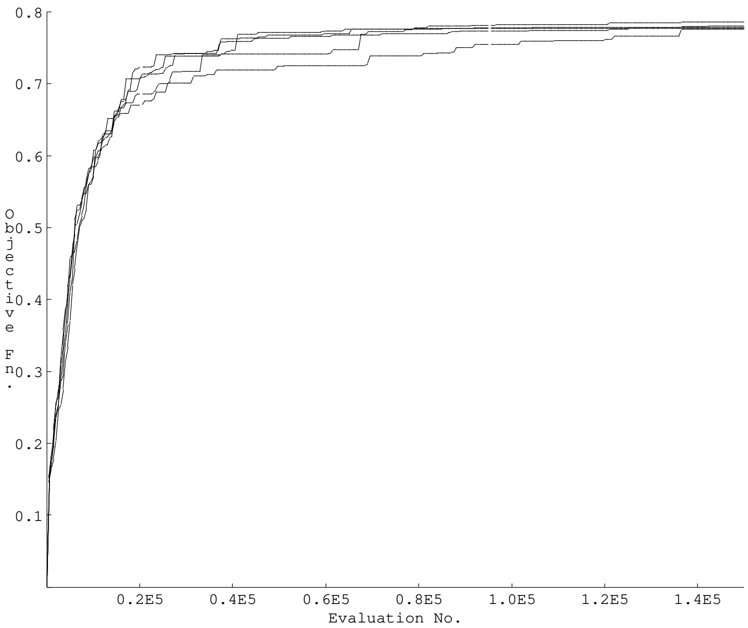

Figure 2a - optimization traces for ‘bump’ using the GA.

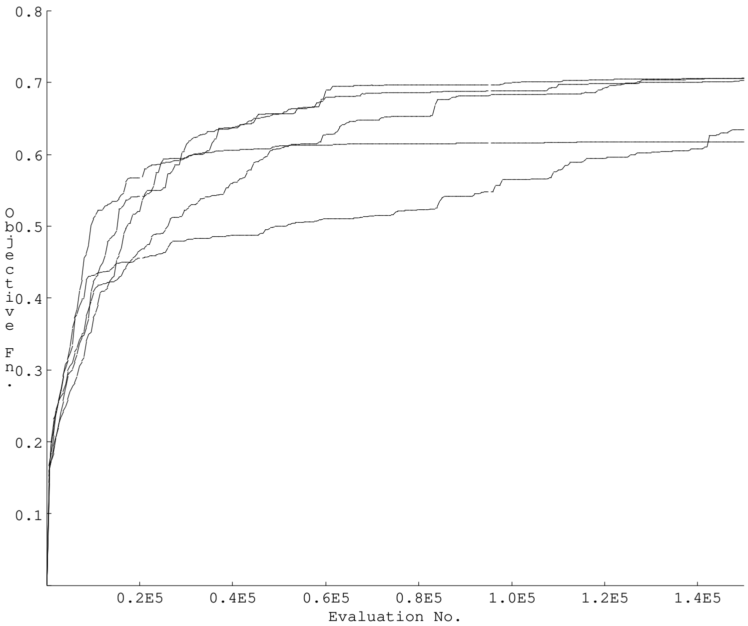

Figure 2b - optimization traces for ‘bump’ using EP.

The use of dU_{first} effectively non-dimensionalizes the annealing temperature and is vital for the schedule to be general purpose. The method used here also allows the user to control how big the changes are between the current point and the next point. This is done by using a binary discretization of the design vector and then allowing a given probability of a bit flip to get to the next trial (by default a 10% chance of each and every bit being flipped). This method of choosing neighbours allows quite radical changes to be proposed at all times and seems to help the search.

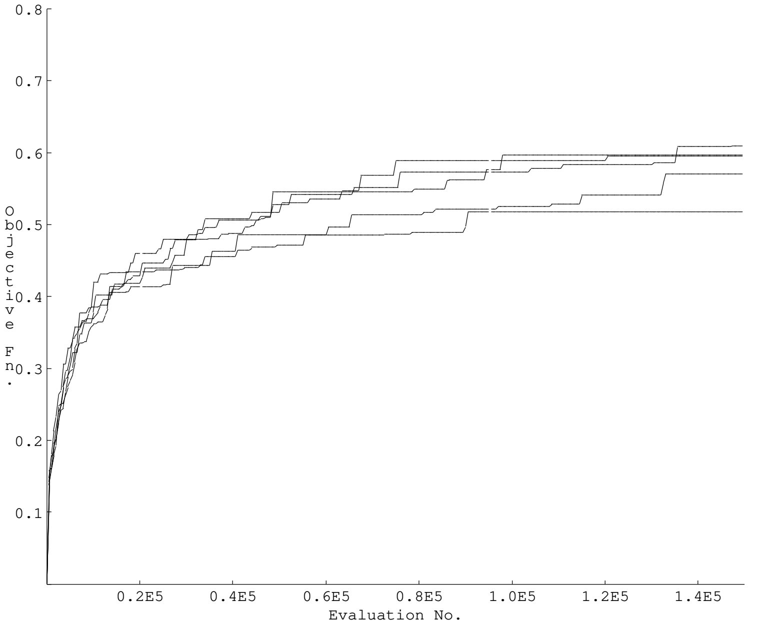

Figure 2c - optimization traces for ‘bump’ using ES.

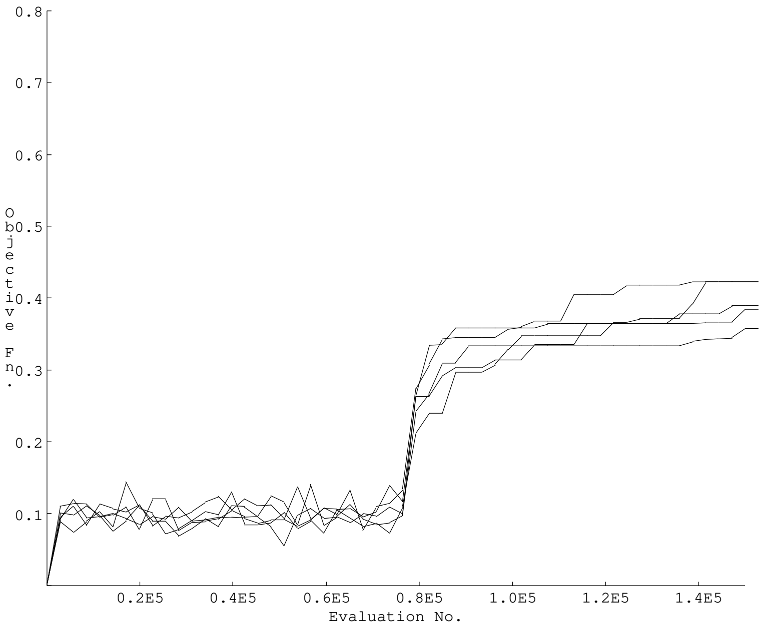

Figure 2d - optimization traces for ‘bump’ using SA.

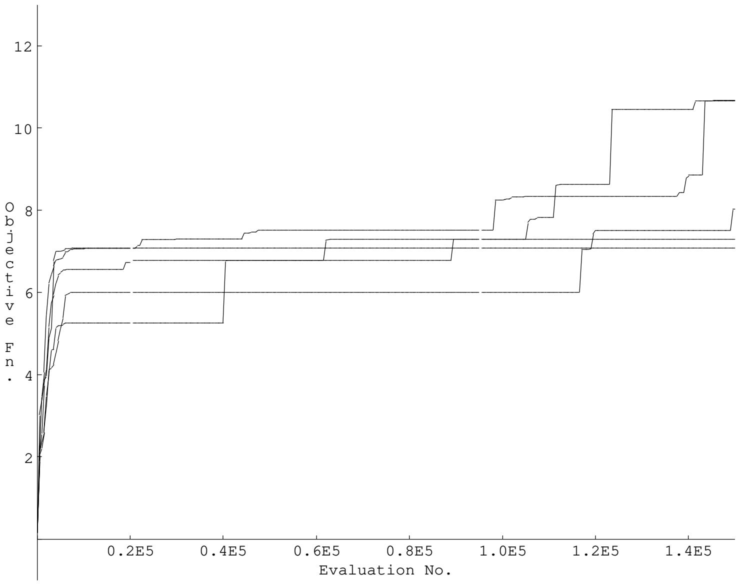

Figure 3a - optimization traces for ‘jhrr’ using the GA.

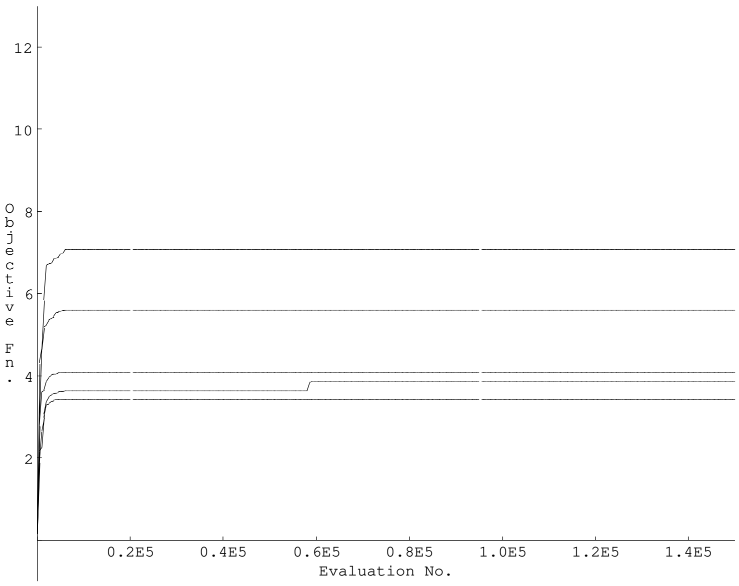

Figure 3b - optimization traces for ‘jhrr’ using EP.

4. INITIAL RESULTS

Application of the various optimizers to the two test problems using 150,000 trials leads to the results of table 1, which are illustrated in figures 2a-d and 3a-d. For these runs a population size of 250 was used for the GA, while for the EP and ES methods populations of 50 were taken. The (μ,λ) version of the ES was used together with a child population size of 100, i.e., twice that of the parent population. The large GA population size reflects the results of earlier studies using the clustering method which show that it works better with bigger populations. Similar increases for the EP and ES methods seem to reduce their efficiency on the functions optimized here. The one pass penalty function was adopted for ‘bump’ in all cases.

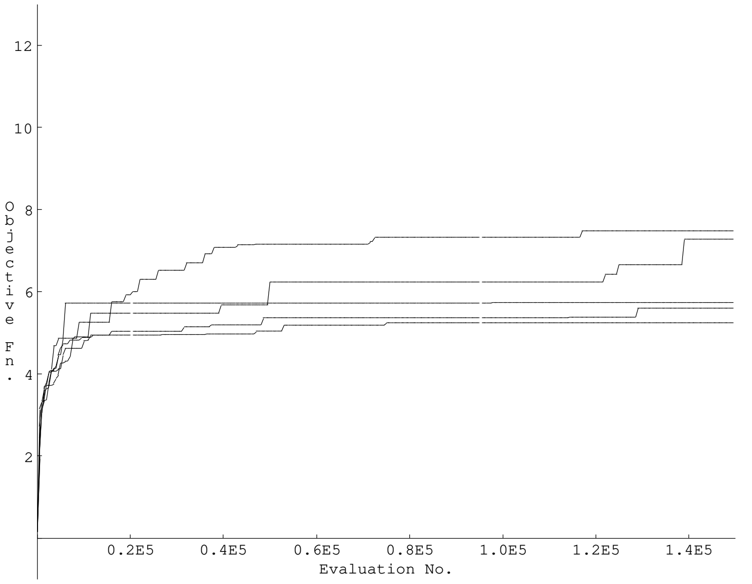

Figure 3c - optimization traces for 'jhrr' using ES.

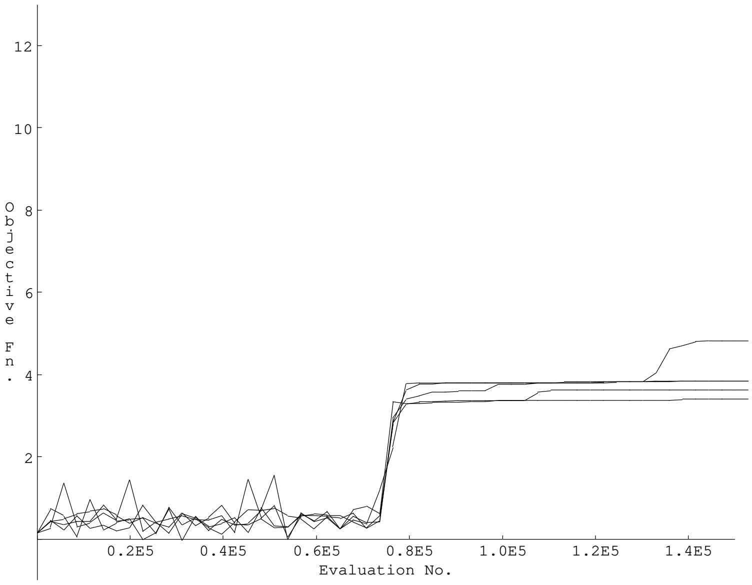

Figure 3d - optimization traces for 'jhrr' using SA.

The figures show the optimization traces for five runs in each case and it is seen that all the methods find the two problems very difficult to deal with. It is notable that the searches all tend to get stuck while dealing with the 'jhrr' problem This is no doubt due to the discrete, step-like nature of its objective function. The EP method in particular seems to suffer in this respect. Also clearly visible are the 'liquid', 'freezing' and 'frozen' stages in the SA runs, indicating that the generic schedules used have spanned the correct temperature ranges. It is also clear that the GA seems to work best and the SA worst with the other two methods somewhere in between. It is, of course, to be expected that all methods could be tuned to produce improved results (it is known, for example, that a hand-coded annealing schedule allows the SA to perform at least as well as the EP and ES methods on the problems studied here). A good optimizer should, however, need little attention in this regard if it is to be widely useful. The reminder of this paper addresses this aspect of optimization.

5. OPTIMIZER TUNING

Having illustrated the difficulties the four methods find with the two test problems, attention is next turned to tuning the parameters that control the actions of the optimizers. Selecting these parameters represents an optimization problem in its own right and a code *has* been developed to tackle this essentially recursive problem[6]. It is, as might be expected, a difficult and time consuming problem to deal with and is not pursued further here. Nonetheless, the GA used does reflect the results of this earlier study on the 'bump' problem, which may go some way to explain its good performance. It has not, however, been tuned to 'jhrr' and so its robustness with this very different problem with the same parameter settings is encouraging (if the mutation control rate, P[mutation] is increased, the GA often achieves the true optimum of 12.8 on this problem).

problem	'bump'						'jhrr'					
method	1	2	3	4	5	avg.	1	2	3	4	5	avg.
GA	0.778	0.777	0.785	0.780	0.776	0.779	7.30	8.02	7.08	10.68	10.66	8.75
EP	0.706	0.706	0.634	0.617	0.703	0.673	3.42	3.86	7.08	5.60	4.08	4.81
ES	0.610	0.597	0.595	0.570	0.516	0.578	7.48	5.60	7.40	5.24	5.74	6.29
SA	0.389	0.423	0.357	0.423	0.384	0.395	4.82	3.84	3.84	3.40	3.62	3.90

Table 1 - initial optimization results, 150,000 trials.

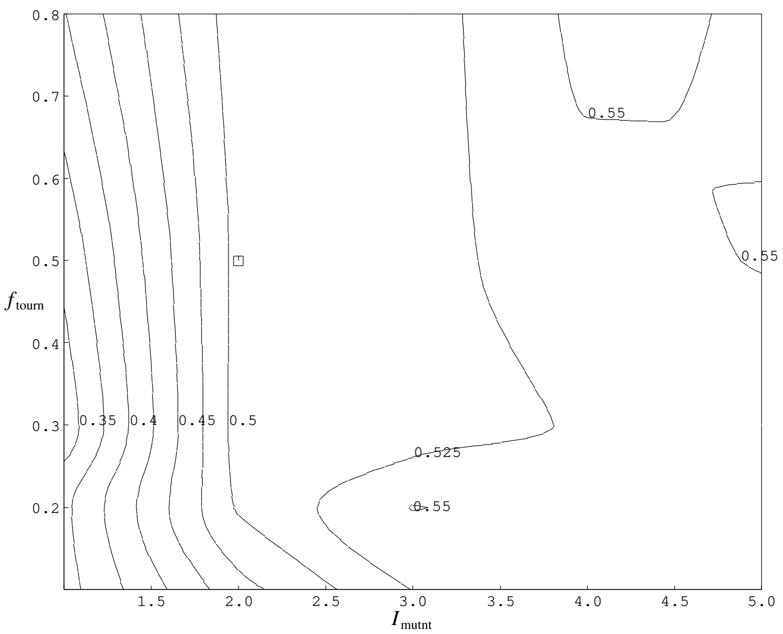

Figure 4a - effect on 'bump' optimizations of variations in EP control parameters.

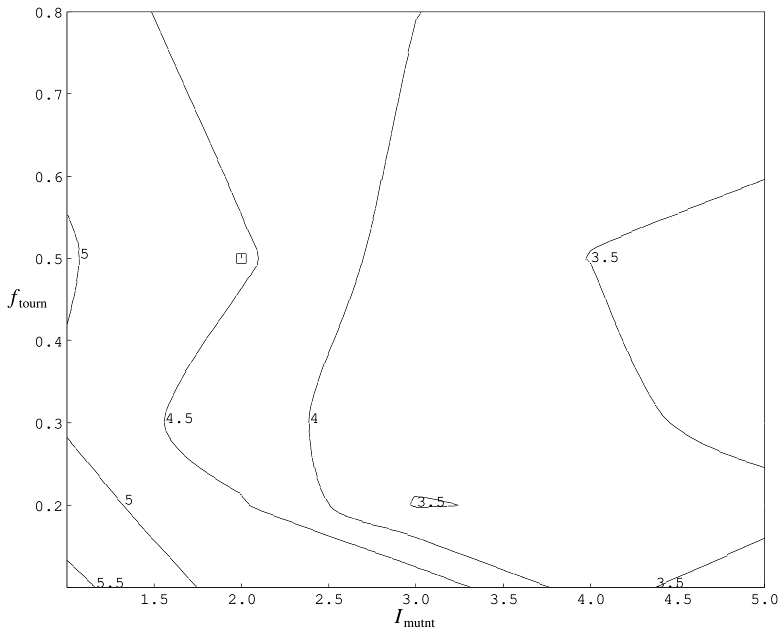

Figure 4b - effect on 'jhrr' optimizations of variations in EP control parameters.

Having noted that a considerable amount of effort has been devoted in the past to tuning the variant of the GA used, attention is next focussed on the EP and ES methods: is it possible to improve their performance by careful selection of their most important control parameters ? The EP method has two such parameters (aside from the choice of penalty function, run length, population size and number of bits in the encoding used here). These are the order of the mutation process with ranking (I_{mutnt} = 1 for linear, 2 for quadratic, etc.,) and the tournament size used in the selection process as a fraction of the population size (f_{tourn}). The ES method has slightly more controls, these being the size of the child population compared to the main or parent population (f_{child}), the mutation vector evolution control rate ($\Delta\sigma$), the choice of intermediate or discrete crossover for both the design and mutation vectors and lastly the distinction between the ($\mu+\lambda$) and (μ,λ) forms of the method.

Figures 4a and b show contour maps of the variation in the average objective function for 'bump' and 'jhrr', respectively. In both cases these are averaged over five runs and are plotted for changes in I_{mutnt} and f_{tourn} using the EP method. 20,000 evaluations have been allowed per run and all other parameters kept as per the previous figures. Figures 5a and b show similar plots using the ES method for variations in $\Delta\sigma$ and f_{child}, using discrete variable crossover, intermediate mutation vector crossover and the (μ,λ) form of the method, as before and which trials have shown best suit both these problems (n.b., as has already been noted, when the child population is smaller than the parent population the shortfall is made up from the best of the parents). In all cases the small square marker on the plots indicates the default values used in producing figures 2 and 3. The contour plots clearly illustrate how strongly the control parameters can affect the performance of the two optimizers.

Consider first the EP method and figures 4a and b. These both show that the tournament size does not strongly influence the method, with values in the range 0.1 to 0.3 being best for these test problems. The class of mutation has, by comparison, a much greater effect. Moreover, linear mutation variations seem to suit the 'jhrr' problem while cubic changes are more suitable for 'bump'. Thus no single setting of this parameter can be proposed as providing a general purpose value: the quadratic mutation variations used by default being clearly a compromise with variations around this value resulting in ±20% changes in performance on the test problems.

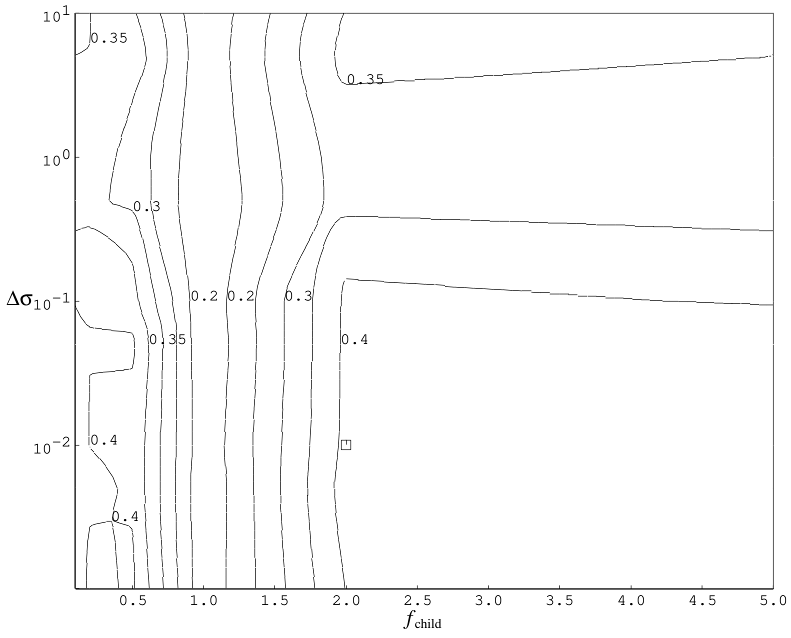

Figure 5a - effect on ‘bump’ optimizations of variations in ES control parameters.

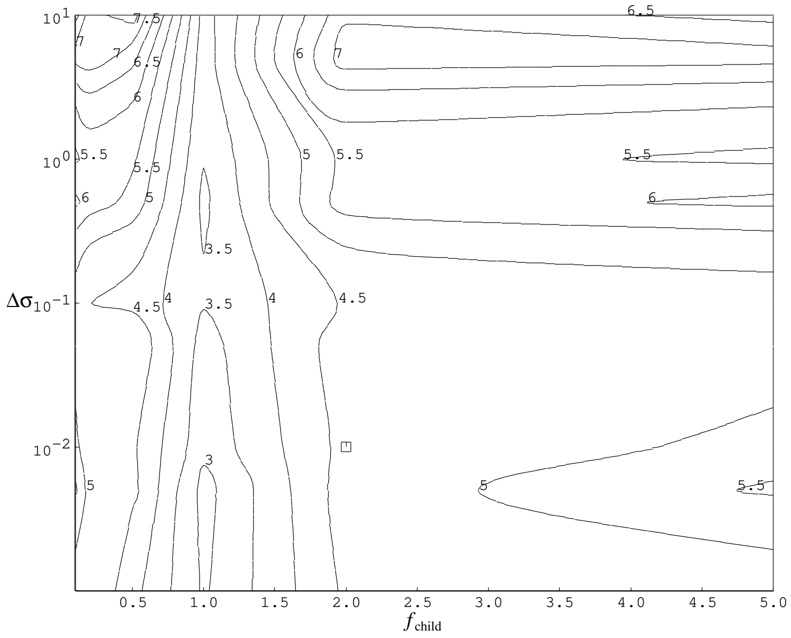

Figure 5b - effect on ‘jhrr’ optimizations of variations in ES control parameters.

Turning next to the ES method and figures 5a and b, it is apparent that both plots show a valley of poor performance when the child population is equal in size to the parent population. Moreover, diametrically opposed values of both $\Delta\sigma$ and f_{child} give the best results for the two test problems studied. Worse is the fact that the range of performances spanned by the plots vary from -50% to +67% when compared to the default values, indicating that the performance of the method is very sensitive to these controls. It does, however, appear that, when correctly set up, the ES method can give good performance on the ‘jhrr’ problem (i.e., with a high value of $\Delta\sigma$ and small child populations).

Finally, consider SA: the method is tuned by modifying the cooling schedule. As has already been set out, the SA schedule used here is a general purpose one that can be controlled via three parameters affecting the ratio of trials per temperature to number of temperatures, the width of the range of temperatures considered and the bias of this range between high and low temperatures. Figures 2d and 3d demonstrate that the temperatures where most improvements take place lie in the middle of those considered and so the parameters P and W were varied with C fixed. This leads to the contour maps of figures 6a and b, again for 'bump' and 'jhrr', respectively. Both figures indicate that, using the generic cooling schedule adopted here, SA performs significantly worse than any of the three other methods, even for quite broad changes in the two key control parameters of the schedule. Nonetheless, the method seems to offer relatively consistent performance over a wide range of controls. Such variation as there is does, however, tend in different directions for the temperature range parameter W, with 'bump' being best handled with $W = 1.2$ and 'jhrr' with $W = 7$. Both methods seem to perform best with low values of P, i.e., few distinct annealing temperatures, although variations in this parameter do not give such significant changes in performance. Finally, it should be noted, as has been mentioned earlier, that hand-coded schedules allow the SA to perform at least as well as EP and ES on these two problems.

6. CONCLUSION

This brief paper has shown that, although they have some advantages, basic implementations of evolutionary programming, evolution strategies and simulated annealing are still all outperformed by an up-to-date GA on the test functions considered here. No doubt further refinements of these methods would allow their performance to be improved. Such improvements would need, however, to go beyond simple parameter tuning and address new mechanisms, such as the niche forming adopted in the GA here. It should also be noted, however, that simple public domain GA's[12] are not so robust against such problems[8]. Perhaps less dependent on the sophistication of the implementations is the robustness of the various methods. The GA seems to be less sensitive to its control parameters than the other methods used here and it is difficult to see how such robustness can be achieved with these methods. So, in summary, it may be said that when comparing new optimizers with existing methods described in the literature, care should be

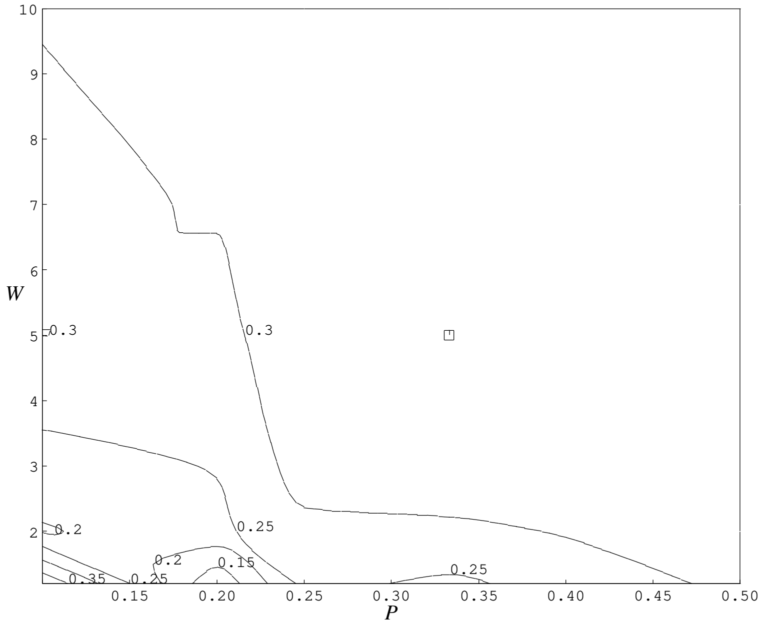

Figure 6a - effect on 'bump' optimizations of variations in SA control parameters.

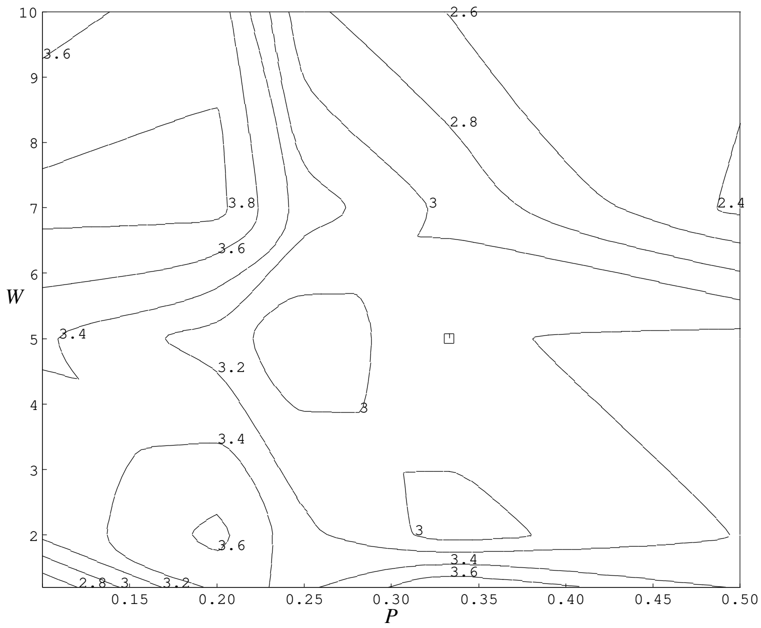

Figure 6b - effect on ‘jhrr’ optimizations of variations in SA control parameters.

taken not to test simply against readily available but rather simplistic public domain versions of existing methods and then only to compare best results. Nonetheless, it remains true that one of the difficulties of working in this field remains that of getting reliable test results for up-to-date methods on functions of interest.

REFERENCES

1. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley (1989).
2. D. B. Fogel, ‘‘Applying evolutionary programming to selected traveling salesman problems,’’ *Cybernetics and Systems* **24**(1) pp. 27-36 (1993).
3. T. Back, F. Hoffmeister, and H.-P. Schwefel, ‘‘A Survey of Evolution Strategies,’’ pp. 2-9 in *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA IV)*, ed. R. K. Belew and L. B. Booker,Morgan Kaufman Publishers, Inc., San Diego (1991).
4. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, ‘‘Optimization by simulated annealing,’’ *Science* **220**(4598) pp. 671-680 (1983).
5. M. Mitchell, S. Forrest, and J. Holland, ‘‘The Royal Road for Genetic Algorithms: Fitness Landscapes and GA performance,’’ in *Proceedings of the First European Conference on Artificial Life*, MIT Press, Cambridge (1992).
6. A. J. Keane, ‘‘Genetic algorithm optimization of multi-peak problems: studies in convergence and robustness,’’ *Artificial Intelligence in Engineering* **9**(2) pp. 75-83 (1995).
7. S. Forrest and M. Mitchell, ‘‘Relative building-block fitness and the Building-Block hypothesis,’’ in *Foundations of Genetic Algorithms 2*, ed. L. D. Whitley,Morgan Kaufman Publishers, Inc., San Mateo (1993).
8. A. J. Keane, ‘‘Experiences with optimizers in structural design,’’ pp. 14-27 in *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control 94*, ed. I. C. Parmee,P.E.D.C., Plymouth (1994).

9. A. J. Keane, ''Structural design for enhanced noise performance using Genetic Algorithm and other optimization techniques,'' pp. 536-543 in *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, ed. R. F. Albrecht, C. R. Reeves and N. C. Steele,Springer-Verlag, Innsbruck (1993).

10. M.R. Anderberg, *Cluster Analysis for Applications,* Academic Press (1975).

11. X. Yin and N. Germay, ''A Fast Genetic Algorithm with Sharing Scheme Using Cluster Methods in Multimodal Function Optimization,'' pp. 450-457 in *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, ed. R. F. Albrecht, C. R. Reeves and N. C. Steele,Springer-Verlag, Innsbruck (1993).

12. J. J. Grefenstette, ''A user's guide to GENESIS,'' Technical Report CS-83-11, Vanderbilt Univ. (1984).