1-D Parabolic Search Mutation

C. Robertson & R.B. Fisher

School of Informatics
University of Edinburgh
Edinburgh, EH1 2QL, UK
craigr@dai.ed.ac.uk

Summary. This document describes a new mutation operator for evolutionary algorithms based on a 1-dimensional optimisation strategy. This provides a directed, rather than random, mutation which can increase the speed of convergence when approaching a minimum. We detail typical comparative results of unimodal and polymodal optimisations with and without the operator.

1 Introduction

1.1 Canonical EA Definition

The general scheme for an EA optimisation consists of a number of operators used in succession :

- *Initialization* Sets of parameters (*genes*) encoded as strings (*chromosomes*) are initialized, generally randomly, within their domain constraints.
- *Evaluation* The strings are then assessed using the evaluation function to give their fitness values. Often these fitness values are stored alongside the chromosomes themselves in some data structure.
- *Selection* Selection schemes determine which of the strings should be propagated into the next generation. These schemes often generate a probability of propagation for each string.
- *Reproduction* A new set of candidate strings are generated from the old ones. Some method is used to take two strings from the old population (parents) to produce two new strings (offspring).
- *Mutation* Elements of strings are changed randomly under some probabilistic selection scheme.
- *Replacement* Offspring replace their parents in the new population if they have a better fitness value.

A canonical EA [1] is a subset of reproductive population algorithms. These are algorithms that keep a collection of candidate solutions that are iteratively

[1] Following and extending the definition given for canonical GA, given in [?].

improved over successive generations. The following characteristics are those that represent a large subset of those found in the literature:

- All candidate solution vectors are the same length.
- The population of solutions is always the same size as the algorithm progresses.
- Reproduction between populations always uses two parent chromosomes, chosen by some probabilistic selection strategy.
- Reproduction is performed by crossover operations.
- Mutation is possible, according to some predefined population-wide mutation probability.

1.2 Canonical EA Operators

The main operator of EAs is the crossover, which carries useful information forwards through population generations. There are many probabilistic ways of selecting chromosomes to crossover. Most are based on some ranking of chromosomes by their fitness value. One example of this is to generate a probability of selection by cumulative summing of the ascending sorted normalised fitness values and using the values as a probability of selection.

Another method is to pick two sets of two chromosomes from the population at random and choose the best of each pair, known as tournament selection.

Crossover itself is generally a matter of overwriting two chromosomes (parents) with the two others (offspring). The offspring are formed by overwriting a random number genes from one parent onto the chromosome of the other. This is done symmetrically to form the two offspring, as shown in figure **??**.

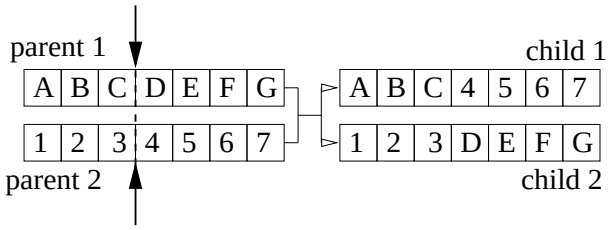

Fig. 1. Canonical 1-point EA Crossover

1.3 *Building Block Hypothesis* and the Problem of Premature Convergence

The reason put forward by Holland (and reported in [**?**]) for convergence in evolutionary algorithms is that they identify good building-blocks and eventually combine these into bigger building blocks. Premature convergence happens when all chromosomes throughout a population become the same. This

means that all crossover operations will yield offspring identical to their parents. If only crossover is used as an operator, it is clear that premature convergence would result very quickly as the best building blocks reached the same chromosome. In order to optimise it is necessary to generate new information. To get to an optimum from this situation though, even using mutation, is a very tedious process. It is also necessary to take timely action to ensure population diversity is maintained as well.

The function of mutation is to add new material into populations and thereby avoid just this premature convergence issue. It is possible to perform a random walk over the whole solution space using only mutation. Some even have argued that cross-over is not strictly necessary in EAs [?] since repeated mutations allow a random walk through the solution space.

1.4 Extensions: Types of Mutation

In this paper, we do not concern ourselves with the multiplicity of crossover operations although we acknowledge that some of them are likely to achieve faster convergence in specific problem domains. We present below a list of mutation operators, which is extensive but not exhaustive.

- *Random* This is the mutation operator as used in a canonical EA. In a chromosome selected by some probabilistic scheme, a random gene is set to a new value randomly assigned inside that gene's domain limits. In essence, this scheme adds new information into the gene pool, although at random. It has a destructive effect on a generational EA because it moves a solution a random amount in a single dimension.
- *N mutations* Essentially the same as one, performed N times. Adds N times as much information but is N times as destructive to an individual chromosome. Moves the solution chromosome in a random N dimensional direction. Similar to an annealing process.
- *Creep* This operator, and those that follow are extensions to the random mutation operator. In creep mutation, we move the selected chromosome a known amount in a single dimension, either forward or backwards, subject to its domain constraints. It is more stable than random mutation although, depending on creep size, likely to force a gene to its domain limits.
- *Directed Hill Climbing Mutation* This mutation requires some memory of previous mutations. If the last mutation of this chromosome produced a better fitness function, them perform the same mutation until it becomes worse, then back off.
- *Domain Limit Mutation* This operator randomly pushes the gene to one or other of its domain limits. This is useful if the solution lies close to the limits of its domain.
- *Non-uniform mutation* This mutation operator is essentially creep mutation with a variable step size. It allows a space to be searched uniformly at first then more locally as generations proceed.

One important consideration when designing mutation operators is *atomic versus complex mutation.* A useful rule when designing mutations is that it is better to have mutations that do one action rather than mutations that do a combination of actions. The reason for this is that the complex mutations can usually be built from a combination of the atomic ones.

2 Algorithm

2.1 New Mutation Operator: Parabolic Search Strategy

Parabolic mutation is a new form of mutation which is an approach to providing the best single step information for a single gene mutation.

1. Define a 1D search envelope of size τ.
2. A single gene is chosen at random from a given chromosome, with value x_1 and corresponding chromosome fitness y_1.
3. Create two new chromosomes by copying the original and replacing the selected gene with values of $x_0 = (x_1 - \tau)$ and $x_2 = (x_1 + \tau)$.
4. Evaluate the new chromosomes and get their fitnesses, $y_0 = f(x_0)$ and $y_2 = f(x_2)$
5. Make a decision based on table I.

Table 1. Decision table for parabolic mutation

Fitness state	1-D Landscape	Minimizer	Maximizer
$y_0 < y_1 < y_2$	increasing	choose x_0	choose x_2
$y_0 < y_1 > y_2$	hilltop	choose min(x_0, x_2)	fit parabola choose x_{best}=max
$y_0 > y_1 > y_2$	decreasing	choose x_2	choose x_0
$y_0 > y_1 < y_2$	valley	fit parabola choose x_{best}=min	choose max(x_0, x_2)

with parabolic fitting [**?**] as seen in figure **??**:
Let

$$m_0 = \frac{(y_1 - y_0)}{(x_1 - x_0)}$$

$$m_1 = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

$$k = \frac{(m_1 - m_0)}{(x_2 - x_0)}$$

The maximum or minimum is then:

$$x_{best} = \frac{(x_0 + x_1 - \frac{m_0}{k})}{2}$$

This will find either maximum or minimum values depending on the values of (x_j, y_j), as shown in figure 2.

6. Replace the gene by the new value and return the chromosome to the population.

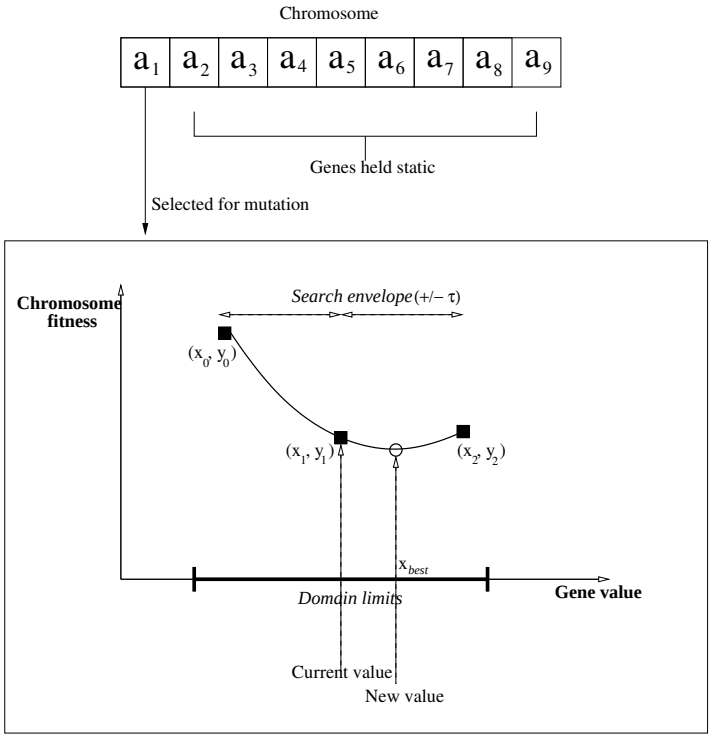

Fig. 2. Parabolic Fit Mutation

Note that if the space around the original value of the gene, x_1, is strictly increasing or strictly decreasing we have essentially a directed creep mutation. When our values straddle a local minimum, however, we can have very fast convergence to that minimum, depending on the size of τ.

2.2 Test Context

We use a canonical EA, as shown in figure **??**. Each of the modules is self-contained and is explained below.

- *Initialization* Each gene in each chromosome is randomly initialized within its domain constraints.
- *Evaluation* Each chromosome is evaluated with a specified fitness function.
- *New Population Generation* The new population is formed in four steps:
  - *Elitist.* This is a common method of ensuring that the best chromosome from the last population is preserved intact. It is simply copied as the first member of the new population.
  - *Crossover.* We use tournament selection to produce the next tranche of population.
  - *Mutation.* We switch between parabolic search mutation and random mutation every iteration.
  - *Re-initialization.* Some of the new population is entirely re-randomized each iteration. This is normally a very low percentage of the chromosomes in the population.

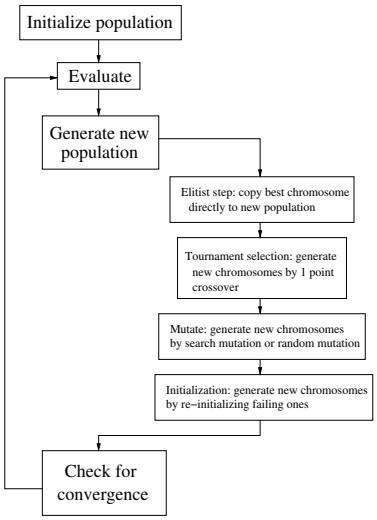

Fig. 3. Algorithm Context

3 Example Applications

3.1 Setup

In order to test speed of convergence, the new operator was employed to optimise some well known functions (outlined in Yao [**?**]). Example times for convergence in iterations and seconds are given, together with some discussion

where relevant. The evolutionary strategy he used $(\mu, \lambda) = (30, 200)$-ES is not directly mappable to our scheme, instead, we use a population of 200 (except where stated) with replacement as outlined in section I.

Note that for unimodal functions, the issues of *speed* of convergence as well as quality of solution are important for our applications. For polymodal solutions, *any* equally important solution is useful. We do not claim that we are able to find either all solutions or the global optimum. For a discussion of how we can find all optima, see section **??**.

In all cases, we give the average results over ten runs for clear convergence to an optimum, in iterations (i.e no further improvement). Note that we do not give timings as all experiments took less than 5 seconds on a 1GHz Athlon PC system running the Linux/GNU operating system.

We also give graphs showing averaged fitness function values of the best chromosome in the Appendix. Note that the fitness function numbering is as in Yao [**?**].

3.2 Single Minimum Function Optimization

The following functions were all optimised trivially (in one population iteration) using the new operator in the context specified:

- Sphere model:

$$f_1(x) = \sum_{i=1}^{30} x_i^2 \tag{1}$$

$$\text{with } -100 \leq x_i \leq 100, \min(f_1) = f_1(0, ..., 0) = 0$$

- Schwefel's Problem No.1:

$$f_2(x) = \sum_{i=1}^{30} |x_i| + \prod_{i}^{30} |x_i| \tag{2}$$

$$\text{with } -10 \leq x_i \leq 10, \min(f_2) = f_2(0, ..., 0) = 0$$

- Schwefel's Problem No.2:

$$f_3(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} (x_j) \right)^2 \tag{3}$$

$$\text{with } -100 \leq x_i \leq 100, \min(f_3) = f_3(0, ..., 0) = 0$$

3.3 Function Optimization with Many Local Minima

The following functions, which have an increasing number of optima of varying importance, were optimised.

Sine Product Function

$$f(\mathbf{x}) = sin(x_0) \times sin(x_1) \tag{4}$$
$$\text{where } x_i \in [-2\pi, 2\pi]$$

This function has two maxima and two minima of equal importance in the given range.

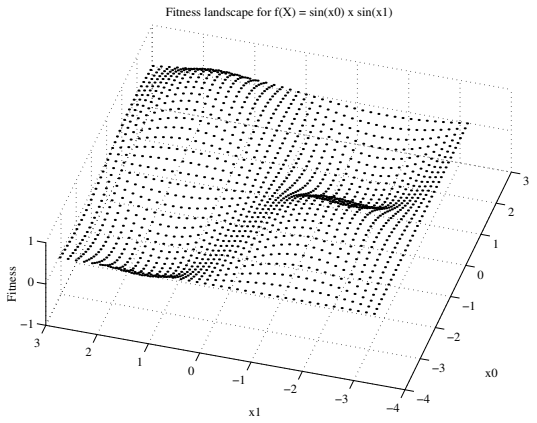

Fig. 4. Sin × sin Function

Product Function

$$f(\mathbf{x}) = x_0 \times x_i \times ...x_{10} \tag{5}$$
$$\text{where } x_i \in [-5, 5]$$

This function has 2 maxima and two minima of equal importance in the given range.

Himmelblau Function

Himmelblau is a set of quartic form functions, generally the following is used:

$$f(\mathbf{x}) = (x_0^2 + x_1 - 11)^2 + (x_0 + x_1^2 - 7)^2 \tag{6}$$
$$\text{with } x_i \in [-5, 5]$$

For which there is a set of known local optima of approximately the same importance:

$f(3, 2) = 0$
$f(-3.78, -3.28) = 0.0054$
$f(-2.81, 3.13) = 0.0085$
$f(3.58, -1.85) = 0.0011$

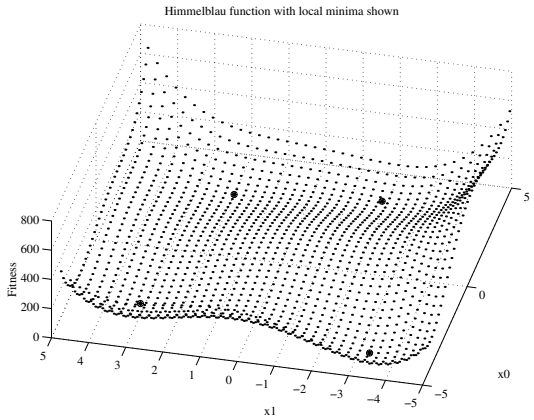

Fig. 5. Himmelblau Function

Six Hump Camel Function, f_{16}

This is a relatively simple function with six omptima inside the given domain.

$$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{7}$$

$$\text{with } -5.0 \leq x_i \leq 5.0,\ \min(f_{16}) = f_{16}(\pm 0.08983, \mp 0.7126) = -1.0316285$$

Generalized Rosenbrock's Function, f_5

$$f_5(x) = \sum_{i=1}^{29}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \tag{8}$$

$$\text{with } -30 \leq x_i \leq 30,\ \min(f_5) = f_5(1, 1, .., 1) = 0$$

Quartic Function with Noise, f_7

$$f_7(x) = \sum_{i=1}^{30} ix^4 + random[0, 1) \tag{9}$$

$$\text{with } -1.28 \leq x_i \leq 1.28,\ \min(f_7) = f_7(0, ..., 0) = 0$$

Generalized Rastrigin's Function, f_9

$$f_9(x) = \sum_{i=1}^{30}[x_i^2 - 10cos(2\pi x_i) + 10)] \tag{10}$$

$$\text{with } -5.12 \leq x_i \leq 5.12,\ \min(f_9) = f_9(0, ..., 0) = 0$$

In this case we used a population of 50 chromosomes.

4 Results

Graphs of convergence are shown in the Appendix (figures 6 to 12). It can be seen that in general the mutation search method works as well or better, that is faster, than regular mutation. In certain circumstances, for example the first three fitness functions, convergence is achieved in a single iteration, indicating that for this kind of (essentially gradient descent) problem search mutation is extremely fast. In other unimodal problems it is generally much faster than regular mutation. It should be noted that because we used regular mutation every few iterations as well, also we benefited from a form of annealing.

- *Sphere model, Schwefel's Problem No.1, Schwefel's Problem No.2.* These were all optimised in a single population iteration using the the new mutation operator. There is no graph shown in the appendix.
- *Sine Product Function.* One optimum for this function was found after a single iteration in both the case of maximization and minimization. A graph of convergence is shown in figure 6 in the appendix. This graph is of interest because the relative time for convergence using the regular operator is much slower and it does not reach the optimum, only approaching it in 5000 iterations.
- *Product Function.* Given the same set of starting chromosomes, the new operator performs as expected and convergences relatively quickly. Shown in figure 7.
- *Himmelblau Function.* Another instance of single iteration convergence with the new operator. It should be mentioned that finding the global minimum in this experiment is luck, rather than a product of the methodology. Shown in figure 8.
- *Six Hump Camel Function,* f_{16}.
  This result shows that this fitness function is particularly complex and varies very rapidly inside its domain limits. This kind of function is not well suited to the operator but it performed as well as the regular one. Both converged very quickly. Shown in figure 9.
- *Generalized Rosenbrock's Function,* f_5.
  Both operators performed similarly but the space is very complex with rapid gradient changes. Various sizes of search window would probably have brought even greater benefits in this example. Shown in figure 10.
- *Quartic Function with Noise,* f_7.
  Fast convergence was achieved with both operators. Notice that the noise causes even the same chromosome to have different evaluations with each iteration. Shown in figure 11.
- *Generalized Rastrigin's Function,* f_9.
  The error converged relatively quickly and to a lower value than the canonical EA. Shown in figure 12.

5 Conclusions and Further Work

We have found that a search mutation is as good or better than regular random mutation in every case we have tested. It has been particularly noticeable that in some contexts it can be used to solve optimisations in a single iteration. This is because the quality of the building blocks is good. Rather than being random they give the best possible information at the time.

In other cases, where the function is not particularly amenable to parabolic fitting, the operator acts as creep mutation until approaching a minimum, where the parabola fitting ensures fast convergence onto the local optimum.

Extension to Multi-Objective Optimisations

If it is necessary to find multiple optima or search for a global optimum there are two methods available to do this, non-random initialization and enforced diversity. If a good estimate of the solution is available, it makes no sense to randomly initialise a population, seeding around this solution is a much better practice. In large and complicated but regular spaces (i.e. locally continuous), one or two passes can be made to heuristically search for good areas in which to perform the optimisation. An example would be to place chromosomes on a regular grid, assess their fitness and then assign a number of search chromosomes to each area based on relative fitness.

One method of ensuring a good spread of chromosomes throughout the space is to test the diversity of the population and reject similar chromosomes. This method is not widely adopted since good measures of diversity are difficult to formulate. This is a problem on which further work is now being done.

Acknowledgements

The work presented in this paper was funded by UK EPSRC grant GR/M97138.

References

1. Robertson C., Fisher R. B., Werghi N., Ashbrook A. P. "An Evolutionary Approach to Fitting Constrained Degenerate Second Order Surfaces". in Evolutionary Image Analysis, Signal Processing and Telecommunications, Proc. First European workshop on evolutionary computation in image analysis and signal processing (EvoIASP99). Goteborg, Sweden, pp 1-16, Springer LNCS 1596, May (1999).
2. Robertson C., Fisher R. B., Werghi N., Ashbrook A. P., "An Improved Algorithm to Extract Surfaces from Complete Range Descriptions", Proc. World Manuf. Conf, WMC'99 (ISMT'99) - Sept. , pp 592-598, ICSC Academic Press, Durham (1999)

3. Robertson C., Fisher R. B., Werghi N., Ashbrook A. P., "Object reconstruction by incorporating geometric constraints in reverse engineering", Computer-Aided Design, Vol 31(6), pp 363-399, (1999).
4. Eggert D, Fitzgibbon A., Fisher R. B., "Simultaneous Registration of Multiple Range Views For Use In Reverse Engineering of CAD Models", Computer Vision and Image Understanding, Vol 69, No 3, pp 253-272, March (1998).
5. Rawlins G. J. E. (ed), *Foundations of Genetic Algorithms*, Morgan Kaufmann Inc., San Mateo, CA, (1991), pp3.
6. Michalewicz Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Third Edition, Springer, (1996).
7. Fogel D. B., "An Introduction to Simulated Evoutionary Optimization", in IEEE Transactions on Neural Networks, Vol.5, No.1, January, pp3-14., (1994)
8. Chen K. , Parmee I. C., Gane C. R., "Dual Mutation Strategies for Mixed-integer Optimisation in Power Station Design.", Proceedings of IEEE International Conference on Evolutionary Computation, Indiana University, April , pp. 385-390., (1997)
9. Roy R., Parmee I. C., "An Overview of Evolutionary Computing for Multimodal Function Optimisation", Proceedings of WSC2. June (1997).
10. Press W.H., Teukolsky S.A., Vetterling W.T. , and Flannery B.P.. *Numerical Recipes in Fortran.* Cambridge University Press, (1992).
11. Yao X. and Liu Y., "Fast evolution strategies," Control and Cybernetics. 26(3):467-496, (1997).
12. Benson K., "Evolving Automatic Target Detection Algorithms theat Logically Combine Decision Spaces", Proc. British Machine Vision Conference, M. Mirmehdi and B. Thomas (eds), BMVA Press, pp 685., (2000).

Appendix - Graphs of Convergence

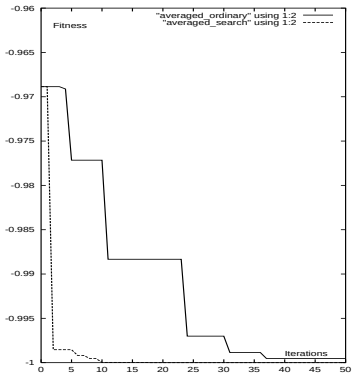

Figure 6: Convergence for Sine × sine Function

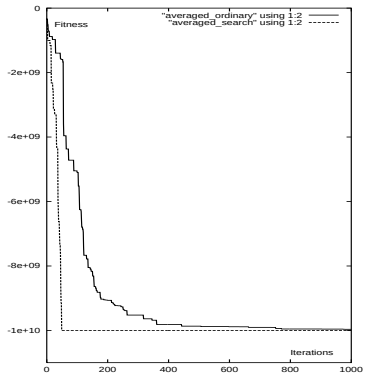

Figure 7: Convergence for Product Function with 10 Genes

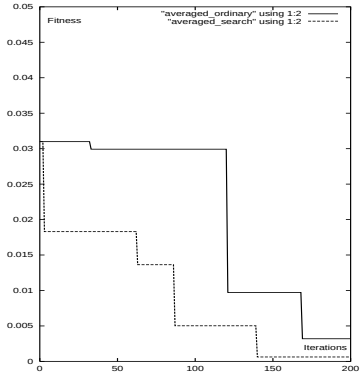

Figure 8: Himmelblau Function Convergence

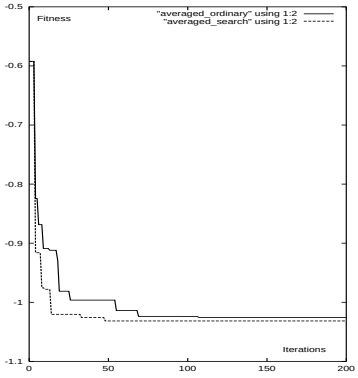

Figure 9: Six Hump Camel Function Convergence

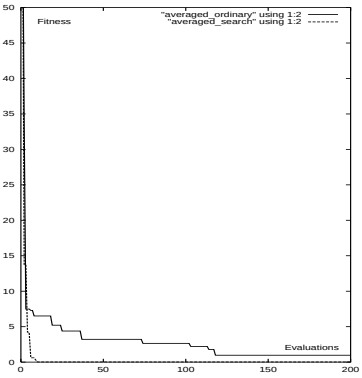

Figure 10: Generalized Rosenbrock's Function Convergence

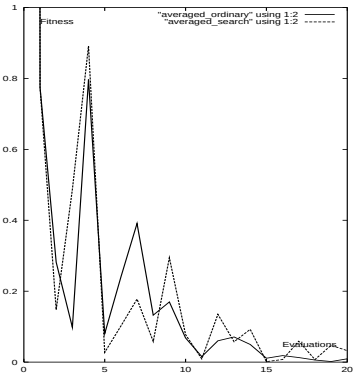

Figure 11: Noisy Quartic Function Convergence

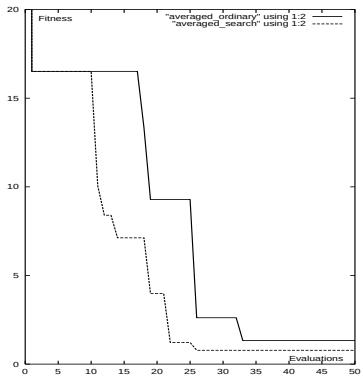

Figure 12: Generalized Rastrigin's Function Convergence