Gaussian Mutation and Self-adaption for Numeric Genetic Algorithms

Robert Hinterding
Department of Computer and Mathematical Sciences
Victoria University of Technology
PO Box 14428 MMC, Melbourne 3000
email: rhh@matilda.vut.edu.au

ABSTRACT

By considering the function variables rather than the binary-bits as genes, new mutation operators can be devised for GAs used to optimise numeric functions. We implement Gaussian mutation operators for Genetic Algorithms used to optimise numeric functions and show it is superior to bit-flip mutation for most of the test functions. Gaussian mutation is a fundamental operator of both Evolutionary Strategies(ES) and Evolutionary Programming(EP). We also implement self-adaptive Gaussian mutation (also used in Evolutionary Strategies and Evolutionary Programming) which allows the GA to vary the mutation strength during the run, this gives further improvement on some of the functions. The performance our GA using a simple implementation of self-adaptive Gaussian mutation is now comparable to ESs. This shows the importance of mutation and the importance of using appropiate mutation operators.

1. Introduction

In Genetic Algorithms (GAs), bit-flip mutation is unchallenged when bit-string representation is used. Most research has concentrated on other issues as mutation is generally considered to be a background operator used only to replace bits lost by crossover (Holland, 1992; Goldberg, 1989). Other forms of mutation have been used when bit-string representation was inappropriate or was not used. Examples of GAs not using bit-flip mutation are Order based GAs (Davis, 1991), Grouping GAs (Falkenauer & Delchambre, 1992) and real valued GAs (Wright, 1991).

The other areas of Evolutionary Computation, Evolutionary Programming (EP) and Evolutionary Strategies (ES) consider mutation to be far more important (Bäck & Schwefel, 1993). In fact in EP, mutation is the only reproduction operator.

In Hinterding, Gielewski & Peachey (1995) it was shown that by considering function variables as genes in GAs using bit-string representation, important characteristics about representation and mutation could be discerned. By considering mutation as an independent reproduction operator, its importance to GAs was demonstrated. The probability distributions of bit-flip mutation on genes using Gray or binary decoding were developed and showed that neither of these types of mutation is ideal as the changes to the function variables cluster very strongly about zero and are sensitive to the granularity used to represent the function variables.

When we consider the function variables to be the genes, we can develop other mutation operators besides bit-flipping. We now have a much larger alphabet for the genes, and can consider mutation to generate a normal distribution of changes. This can be simulated by adding Gaussian noise to gene values (Gaussian mutation). Gaussian mutation is not new to Evolutionary Computation as it has been used in both Evolutionary Strategies and Evolutionary Programming (Bäck & Schwefel, 1993; Saravanan & Fogel, 1994).

We investigate the usefulness of Gaussian mutation in GAs used for numeric function optimisation. We maintain bit string representation so that the distinction between genotype and phenotype is maintained, and control over the granularity of representation is possible. Hence, the changes to the GA paradigm are minimised and the normal crossover operators can be used.

Self-adaption is a powerful tool that has long been used in ESs (Bäck, 1992; Schwefel, 1995) to set the values of the strategy parameters. Bäck (1992) used self-adaption in GAs to control the probability for flipping bits representing one of more function variables. We use self-adaption to allow the GA to control the variance for Gaussian mutation.

We show that Gaussian mutation can give superior results to bit-flipping mutation for most functions, in particular Gaussian mutation is better for some of the "deceptive" problems. We also show that using self adaption to control the variance for

Gaussian mutation gives significant improvement for most of the functions and that the changes of the variance over time are appropriate for the different types of functions being optimised.

2. Gaussian mutation

By considering the variables of the function that are being optimised as the genes, we can define new mutation operations besides the standard bit-flipping mutation operator. The distribution of changes to genes by using bit-flipping mutation with either Gray or binary decoding is not ideal. When Gray decoding is used, most of the changes are very small and it is very sensitive to choosing the correct granularity. Binary decoding has the effect that all possible changes are equally likely, but there are large gaps in the distribution of possible changes. It seems reasonable therefore that mutation which has normally distributed changes could produce better results and reduce the sensitivity to choosing the length of the variables.

We implement fixed Gaussian mutation by first decoding the gene to an unsigned integer and then adding Gaussian noise, the new value is then encoded and replaces the gene in the chromosome. Gaussian noise is obtained from a normally distributed random variable which has a mean of 0 and a standard deviation of 0.1 times the maximum value of the gene. The value 0.1 was chosen so that an interval of more than 3 standard deviations each side of the mid-point of the range will encompass most of the range. With this scheme new gene values can exceed the range at either end of the range if the original gene value is sufficiently far away from the mid-point of the range. These values can be treated in two ways, we can either truncate the value back to the end-point it exceeded, or the value can be allowed to "wrap around", that is multiples of the range are added or subtracted to give a value within the range.

The effects of Gray or binary decoding are removed from mutation, as the value is decoded and then mutated, and then encoded before replacing the original gene.

2.1. Self-adaptive Gaussian Mutation.

In ESs and in meta-EPs self-adaption is used to control strategy parameters. Here learning is used to adjust the strategy parameters while searching for the optimum. One of the problems with Evolutionary Computation algorithms is finding values for the parameters which will optimise their performance. The optimal values for the parameters are generally not constant during the whole run, and also depend on the function to be solved. Hence allowing the algorithm to adjust the values during the run can have large benefits.

We add one value to each chromosome, which will control the standard deviation of the Gaussian mutation used on the genes in the chromosome that will be mutated. This value is allowed to participate in crossover and mutation, but does not contribute directly to the fitness of the solution. This simple scheme should allow us to test its utility.

We implement self-adaptive mutation by adding an extra gene to the front of all chromosomes. The values of this gene are allowed to vary from 0.00002 to 0.2. This gene is allowed to participate normally in crossover, but when the chromosome is to be mutated the following steps are followed:

1. decode the gene to a value.
2. apply Gaussian noise to the value using a standard deviation of 0.013. This value was found experimentally to give good results.
3. use this value as the standard deviation of the Gaussian noise to mutate the other genes in the chromosome.
4. write the mutated special gene and the mutated genes back to the chromosome.

When the initial population is created, the special genes are given random values around 0.1.

3. Numeric GA Test functions

The test functions have been taken from a number of sources, and more difficult functions have been included. F1 - F5 are from De Jong (1975), F6 - F8 are from Scott & Whitley (1993), F9 is from Hoffmeister and Bäck (1992), and F10 is from Michalewicz (1994). The range, number of variables and number of bits used to represent each variable is summarised in Table 1.

- F1 Sphere Model. This is a continuous, strictly convex, unimodal function. The solution is at $x^* = (0, \ldots, 0)^T; f_1(x^*) = 0$. This is a scalable problem, with $n = 3$ the problem is easy, Hoffmeister & Bäck (1992) use $n = 30$, which we call test function F1a.
- F2 Rosenbrock's Function. This is a continuous, unimodal, bi-quadratic function of two variables. It is a standard test function in optimisation which was proposed by Rosenbrock (1960). The solution is at $x^* = (1, 1)^T; f_2(x^*) = 0$.
- F3 De Jong Step Function. This is a simple linear but discontinuous function, which consists of many small plateaus. Due to this characteristic f3 has a lot of local optima. The solution is at $x_i \in [-5.12, \ldots, -5); f_3(x^*) = 0$.
- F5, Shekel's Foxholes. This is a continuous, non-linear, multimodal function proposed by Shekel (1971). It is a difficult problem as it consists of a large plateau with some holes

Fn.	Range	No Vars	Bits/Var	Tot Len
F1	± 5.12	3	10	30
F1a	± 5.12	30	10	300
F2	± 2.028	2	12	24
F3	± 5.12	5	10	50
F5	± 65.536	2	17	34
F6	± 5.12	20	10	200
F7	± 512	10	10	100
F8	± 512	10	10	100
F9	± 65.536	20	17	340
F10	± 200.0	45	12	540

Table: 1: Function Characteristics

in it which have different objective values at their bottoms, while the plateau is made up of equal objective values. The solution is at $x_i^* = (-32, -32)^T; f_5(x^*) \approx 1$.

- F6, Generalised Rastrigin's Function. This function is a scalable, continuous, mulitmodal test function which is made from F1 by modulating it with $Acos(2\pi x_i)$. It was first proposed by Rastigin as a 2-dimensional problem (Törn & Žilinskas, 1989), and has been generalised by Rudolph (1991) as a test function for distributed parallel ESs. The solution is at $x^* = (0, \ldots, 0)^T; f_6(x^*) = 0$.
- F7, Schwefel's Function. This is a multimodal function characterised by a second-best minimum which is far away from the global optimum. The solution is at $x^* = (421, \ldots, 421)^T; f_7(x^*) = 0$.
- F8, Griewangk's Function. This function is difficult for GAs because the variables are strongly independent. The solution is at $x^* = (0, \ldots, 0)^T; f_8(x^*) = 0$.
- F9, Schwefel's Problem 1.2. This is a continuous, unimodal function which comes from the set of test functions that Schwefel (1977) once used to compare the performance of several optimisation methods. The difficulty of this function results from the fact that searching along the coordinate axes only gives a poor rate of convergence. The solution is at $x^* = (0, \ldots, 0)^T; f_9(x^*) = 0$.
- F10, Michalewicz's dynamic control problem. The range of x is (-200, 200), the function has a minimum at 16,180.4.

4. The GA

The Genetic Algorithm used is a steady-state GA based on the description of OOGA in Davis (1991). Tournament selection is used with a tournament size of 2 as this was faster and gave comparable results to roulette wheel selection with linear normalisation. It was developed using Smalltalk/V for Windows. The following parameters can be set:

- Population Size - set the size of the population.
- Allow Duplicates - set a flag to allow or disallow duplicates to exist in the population. If duplicates are not allowed, any duplicates produced by reproduction are discarded while they still count as an evaluation. We determine whether two chromosomes are the same by comparing their genotypes.
- Number of Evaluations - set the number of evaluations for the run. We use evaluations rather than generations so that we can compare between runs where the population size and replacement rate are different.
- Replacement Rate - set the percentage of the population that will be replaced by reproduction in one generation. The rate can be set from 0 to 100%.
- Crossover Rate - set the percentage of the replacement population that will be replaced by crossover in one generation. The remainder of the replacement population will be produced by mutation. The rate can be set from 0 to 100%.
- Poisson Mutation - use a Poisson distributed random variable to determine how many genes to mutate in a chromosome. If this is false only one gene per chromosome is mutated.
- Poisson Mean - set the mean (λ) for the Poisson distributed random variable. This controls the number of genes that will be mutated in the chromosome if Poisson Mutation is true.

Two point crossover was used for all the experiments, although others were available.

In the Genetic Algorithm used, a new chromosome is produced either by crossover or mutation but not both. In this way we treat crossover and mutation as independent reproduction operators. This was done so that the separate effects of these reproduction operators could be determined. By using these operators independently and varying the application rates we can determine the mix of these operators that produces the best results within a given number of evaluations.

The mutation rate for the GA is (100% - Crossover rate). The mutation rate is the percentage of chromosomes of the replacement population that will undergo mutation. If Poisson Mutation is false, then mutation of one gene is carried out. If Poisson Mutation is true, then the number of genes to be mutated in a chromosome is determined by sampling a Poisson distributed random variable with mean λ. If n genes are to be mutated in a chromosome, then we repeat n times: select a random gene from the chromosome and mutate the gene. Note that mutation is now controlled by the crossover rate, the number of genes in the chromo-

Fn.	Evals	Best	λ	μ	Crx.	Rep
F1	4,000	2510#	2.7	0.09	40	20
F1a	12,000	7.9e-2	2.1	0.007	50	40
F2	12,000	3.3e-5	3.6	0.15	20	80
F3	6,000	5090#	3	0.06	10	70
F5	4,000	1650#	2.4	0.07	10	90
F6	12,000	2.8e1	2.2	0.011	80	90
F7	12,000	2.5e2	4.2	0.042	10	40
F8	12,000	1.2e-1	3	0.03	40	40
F9	12,000	9.5e2	5.8	0.017	10	90
F10	20,000	9.6e4	3.4	0.006	10	20

Table: 3: Results from bit-flip mutation

Fn.	Best Mutation type
F1	Bit-flip
F1a	Gauss. Self Adapt
F2	Gauss. Self Adapt
F3	Gauss
F5	Bit-flip
F6	Gauss
F7	Gauss
F8	Gauss Self Adapt
F9	Gauss Self Adapt
F10	Gauss Self Adapt

Table: 4: Best results by GA type

some to mutate, and the standard deviation of the Gaussian noise if Gaussian mutation is used.

5. The Experiments

Table 2 show the results for the GAs using fixed and self-adaptive Gaussian mutation. Three sets of runs were performed on each problem, so that the parameters could be varied to get some indication of the best settings for each problem. The parameters that were varied were the crossover (and mutation rate), the replacement rate, and whether Poisson based mutation used. In the set of runs where Poisson based mutation was used, the Poisson mean was varied. In each run the GA is run on the problem 20 times and the results are averaged.

For comparison, results from GAs using bit-flip mutation are shown in Table 3. In Table 3, the column "Evals" shows the number of evaluations that the GAs performed in each run for that function. In tables 2 & 3 "Best" shows the best value found if the number is in E notation or the number of evaluations required to find the optimum if the number is suffixed by a "#". "Crx" shows the crossover rate for the best results, and "rep" show the replacement rate which gave best results for a crossover rate of 50%.

The figures in bold show the best results obtained for that function in the table.

In Tables 2 and 3, "λ" shows the Poisson mean which gave the best results. The data from Table 3 is from Hinterding, Gielewski and Peachey (1995), the same same GA was used to produce these results. Here bit-flip mutation and Gray encoding was used.

6. Discussion and Conclusions

By treating the function objective variables as genes, we are able to introduce new mutation operators which could not be envisioned when we treat the binary bits of the chromosome as genes.

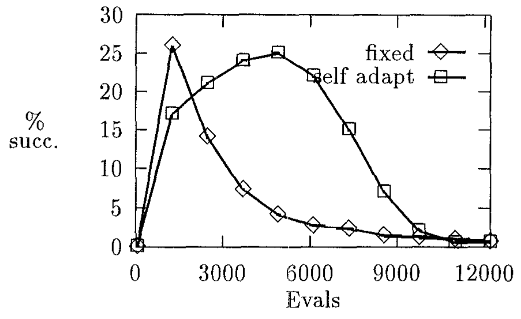

Fig. 1: Percentage successful mutations for F1a

We can see from Table 4 that GAs using Gaussian mutation produced the best results for most of the functions. We can therefore conclude Gaussian mutation is a useful mutation operator and is in most cases superior to bit flip mutation. Our GAs use mutation as an independent reproduction operator, that is a new individual (chromosome) is produced by either crossover or mutation and not both. By comparing Tables 2 and 3 we can see that when using Gaussian mutation, the crossover rate is more often near 50%. We speculate that this could be due to Gaussian mutation producing values that crossover can more easily utilise.

GAs using self-adaptive Gaussian mutation work, they produce values that are close to or better than the GAs using fixed Gaussian mutation. The GA keeps track of the number of mutations that are better than the individual that the mutant was produced from. Figure 1 shows the rolling average of successful mutations for function F1a. The self adaptive GA keeps the percentage higher for all of the run, showing that it can successfully optimise the variance for the Gaussian mutation during the run.

Figure 2 shows the change in the variance of the Gaussian mutation when self-adaption is used.

	fixed				self-adaptive			
Fn.	Best	λ	Crx.	Rep	Best	λ	Crx.	Rep
F1	**3570#**	1.5	50	80	3830#	2	40	60
F1a	4.6e-2	1.5	50	20	**9e-5**	10	40	60
F2	8.9e-5	5	40	90	**7490#**	5	10	80
F3	**1895#**	2	30	30	2550#	1	50	50
F5	**2690#**	2	50	80	3830#	1	60	60
F6	**1.9e0**	1.2	50	40	3.1e0	1.7	60	60
F7	**5.6e1**	2.5	50	20	7.3e1	4	60	90
F8	1.6e-1	1	60	20	**7.9e-2**	1.5	40	20
F9	3.6e2	2	10	90	**1.4e2**	10	10	40
F10	4.2e4	2.5	30	90	**2.6e4**	15	10	20

Table: 2: Results for Gaussian mutation

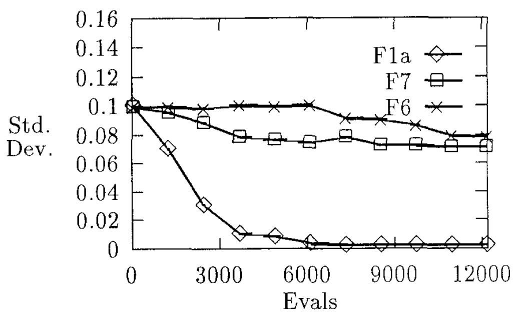

Fig. 2: Change in variance in Self Adaptive GA

For F1a the variance is decreased as expected and is very small towards the end of the run, while the variances for F6 and F7 are both kept high throughout the run. Function F6, Rastrigins's function is characterised by many local optima, in this case keeping the variance high would optimise the chance of hitting a peak. F7 is a "deceptive" problem with a local optimum a long way from the optimum, again keeping the variance high is advantageous. This shows that self adaption is capable of adapting to different situations successfully.

Part of the improvement for function F7, with the GAs using Gaussian mutation we attribute to the fact that "wrap around" is used in the Gaussian mutation operator. F7 is like the "deceptive" problems (Deb & Goldberg, 1992) in that it has a second best optimum near one end of the range while the optimum is at the other end (see Fig 3). By allowing values being mutated to "wrap around" a much short path to the optimum value is provided. This result looks very promising and could eliminate a class of "deceptive" problems.

When we now compare our results with those of ESs in Hoffmeister and Bäck (1992) (see Table 5), we see that our GAs using Gaussian or self-adaptive Gaussian mutation now have comparable performance. ESs appear to do better on the "smoother" functions, while our GAs do better on the "rougher" functions.

Fn.	ES	Our GAs
F1a	**1e-5**	9e-5
F2	**5000#**	7490#
F3	1e0	**1895#**
F5	3.5e0	**2690#**
F6	6e0	**1.9e0**
F9	**2e1**	1.4e2

Table: 5: Comparison of results with Hoffmeister and Bäck

Acknowledgments

The author acknowledges the anonymous reviewers for their helpful comments and remarks and would would like to thank Ted Alwast, Tom Peachey and Harry Gielewski.

References

Bäck, T. 1992. Self-adaption in Genetic Algorithms. *In: Proceedings of the First European Conference on Artificial Life.* Cambridge: MIT Press. pp. 263-271.

Bäck, T., & Schwefel, H-P. 1993. An Overview of Evolutionary Algorithms. *In: Evolutionary Computation.* 1, vol. 1. MIT Press.

Davis, L. (ed). 1991. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold.

De Jong, K. A. 1975. *An analysis of the behaviour of a class of genetic adaptive system.* Doctoral dissertation, University of Michigan.

Deb, K., & Goldberg, D. E. 1992. Analysing Deception in Trap Functions. *In:* Whitley, L. D. (ed), *Foundations of Genetic Algorithms - 2.* Morgan Kaufmann.

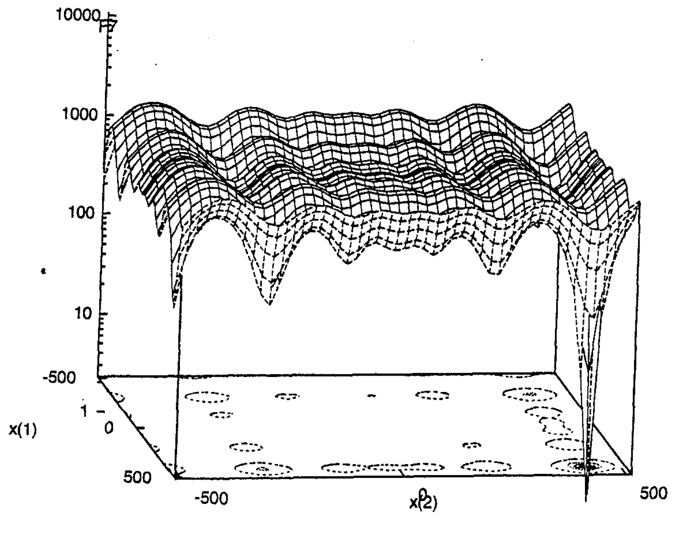

Fig. 3: Schwefel's Function

Falkenauer, E. A., & Delchambre, A. 1992. A Genetic Algorithm for Bin Packing and Line Balancing. *In: Proceedings of 1992 IEEE International Conference on Robotics and Automation(RA92).* pp. 1186-1193.

Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley.

Gordon, V.S., & Whitley, D. 1993. Serial and Parallel Genetic Algorithms as Function Optimizers. *In:* Forrest, Stephanie (ed), *Proceeding of the Fifth International Conference on Genetic Algorithms.* Urbana-Champain: Morgan-Kaufmann. pp 177-183.

Hinterding, R., Gielewski, H., & Peachey, T. C. 1995. The Nature of Mutation in Genetic Algorithms. *In:* Eshelman, L. J. (ed), *Proceedings of the Sixth International Conference on Genetic Algorithms.* Morgan Kaufmann. pp. 65-72.

Hoffmeister, F., & Bäck, T. 1992 (Feb). *Genetic Algorithms and Evolution Strategies: Similarities and Differences.* Technical Report No. SYS-1/92. Systems Analysis Research Group, University of Dortmund, Germany.

Holland, J. H. 1992. *Adaption in Natural and Artificial Systems.* 2nd edn. MIT Press.

Michalewicz, Z. 1994. *Genetic Algorithms + Data Structures = Evolution Programs.* 2nd edn. Springer - Verlag.

Rosenbrock, H. H. 1960. An automatic method for finding the greatest or least value of a function. *In: The Computer Journal.* 3, vol. 3. pp 175-184.

Rudolph, G. 1991. Global Optimization by means of distributed evolution strategies. *In: In Parallel Problem Solving from Nature.* Lecture Notes in Computer Science, vol. 496. Springer-Verlag.

Saravanan, N., & Fogel, D.B. 1994. Learning Strategy Parameters in Evolutionary Programming: An Empirical Study. *In:* Sebald, A.V., & Fogel, L.J. (eds), *Proceedings of the Third Annual Conference on Evolutionary Programming.* World Sci.

Schwefel, H-P. 1977. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie.* Interdisciplinary systems research, vol. 26. Basel: Birhäuser.

Schwefel, H-P. 1995. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology Series. Wiley.

Shekel, J. 1971. Test functions for multimodal search techniques. *In: Fifth Annual Princeton Conference on Information Science and Systems.*

Törn, A., & Žilinskas, A. 1989. *Global Optimization.* Lecture Notes in Computer Science, vol. 350. Springer-Verlag.

Wright, A.H. 1991. Genetic Algorithms for Real Parameter Optimization. *In:* Rawlins, G.E. (ed), *Foundations of Genetic Algorithms.* 3, vol. 3. Morgan Kauffmann. pp 205-218.