Binary versus real coding for genetic algorithms: A false dichotomy?

J. Gaffney[1] D. A. Green[2] C. E. M. Pearce[3]

(Received 24 December 2009; revised 25 May 2010)

Abstract

The usefulness of the genetic algorithm (GA) as judged by numerous applications in engineering and other contexts cannot be questioned. However, to make the application successful, often considerable effort is needed to customise the GA to suit the problem or class of problems under consideration. Perhaps the most basic decision which the designer of a GA makes, is whether to use binary or real coding. If the variable of the parameter space of an optimisation problem is continuous, a real coded GA is possibly indicated. Real numbers have a floating-point representation on a computer and the decision space is always discretised; it is not immediately evident that real coding should be the preferred method for encoding this particular problem. We re-visit this, and other decisions, which GA designers need to make. We present simulations on a standard test function, which show the result that no one GA performs best on every test problem. Perhaps the initial choice to code a problem using a real or binary coding is a false dichotomy. What counts are the algorithms for implementing

http://anziamj.austms.org.au/ojs/index.php/ANZIAMJ/article/view/2776 gives this article, © Austral. Mathematical Soc. 2010. Published June 22, 2010. ISSN 1446-8735. (Print two pages per sheet of paper.)

the genetic operators and these algorithms are a consequence of the coding.

Contents

1 Introduction C348

2 Genetic algorithms: overview C349

3 Implementing a genetic algorithm C349

4 Algorithms for genetic operators C351

5 Simulation examples C352

6 Concluding remarks C354

References C358

1 Introduction

There are many survey articles describing the application of GAs in engineering contexts [1]. This article discusses some of the design considerations involved in implementing a GA to solve a particular problem. We need to formulate the problem, code the solutions and develop algorithms to represent the genetic operators. Our animations of the simulations of our toy GA show how a GA works in simple settings, see Figures 2–4.

The question of whether to use a binary or real encoding of a GA can be contentious. The traditional GA uses a binary encoding [2]; however, in many applications real encoding is used [3]. Various arguments are given as to whether a binary or real encoding should be used; it is not always immediately evident which encoding method should be adopted. It appears that the discretisation of the parameter space plays a role in the computational efficiency of the GA.

2 Genetic algorithms: overview

The GA is a random search technique for finding good solutions [4]. Genetic algorithms are loosely based on the evolution of biological systems. An objective function is identified to quantify the fitness of candidate solutions and the candidate solutions are encoded by binary strings or by finite decimal representations of real numbers.

For a traditional GA, a population of N candidate solutions is chosen initially from the search space of encoded solutions and the fitness of each string is evaluated. A 'mating pool' is selected from this population of candidate solutions. The genetic operators crossover and mutation act on the mating pool to give the next set of candidate solutions for the next iteration.

A non-standard elitist operator is sometimes also applied after selection, crossover and mutation.

3 Implementing a genetic algorithm

The first step taken to implement a GA is to determine how the 'fitness' of solutions is to be calculated and, for engineering design problems, this can be a considerable challenge. However, the situation for a function optimisation problem is straightforward; use the value of the function. Next, a method for encoding the solutions is chosen. These steps are not independent and it is increasingly the case that real coding is recommended for optimisation problems where the parameter space is continuous.

Table 1: Example three bit binary code comparison.

Decimal	Gray code	Std Binary code
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

We consider the simple case of a one variable function optimisation problem and how solutions to this problem might be coded. Suppose that the parameter space is $[0, M]$. To implement a GA we discretise $[0, M]$. Coding mechanisms give rise to equally spaced possible decisions. We define the *grid spacing* Δ of a coding to be the difference between two consecutive coded decisions. For a binary code of fixed length l, the gridspacing $\Delta = M/(2^l - 1)$. In using a binary code we determine how many bits we should use in the binary string to give a grid spacing, which is fine enough. For real coding the precision is given by the fixed decimal point representation of real numbers on a computer, where for example $\Delta = 0.0000000001$ for a ten decimal point representation.

The binary code, known as a Gray code (reflected binary) after Frank Gray, is a binary numeral system where two successive values differ in only one bit. Table 1 gives the representation for each of the codes for a three bit binary string

The computational efficiency of a GA is an aspect of the choice of an encoding. We do not explore this question in this article; however, we note the interplay of the size of the parameter space and the representation of the real or binary numbers in determining computational efficiency.

4 Algorithms for genetic operators

We briefly discuss the algorithms (selection, crossover, mutation and elitist selection) we used in our simulations. We used both binary and real coding.

Selection (of the 'mating pool') The algorithm generates a new population of candidates from the current candidates by probabilistically selecting higher ranked candidates according to 'fitness'.

Crossover The algorithm exchanges the last k-bits between pairs of solutions in the mating pool according to some probability p_{cross} to produce two candidate solutions for the next iteration, if the solutions are binary coded. If real coding is used, the algorithm uses a convex combination of the two candidates in the mating pool according to some probability p_{cross} to produce two new candidate solutions for the next iteration.

Mutation With binary coding, the algorithm acts on candidate solutions generated by flipping bit-values according to some probability value p_{mut}. Alternatively, the algorithm acts on candidate solutions, which are real coded by manipulating the value of the candidates as follows: for each value place that may be held by a candidate solution, we add the value $\texttt{rand}([-5, 5])$ at that value place to the actual candidate solution with probability p_{mut}, where $\texttt{rand}([-5, 5])$ is a random integer in $[-5, 5]$. This is to cover a range of mutations 'about' the current candidate solution consistent with the place value separation in the decimal system. The final result of mutation is only accepted if it is a feasible candidate solution.

Elitist selection This algorithm is sometimes applied to ensure that the new solutions generated by Crossover and Mutation are only accepted for the next iteration if they are no worse than the two solutions used to generate them.

5 Simulation examples

We simulated the minimisation of *Ackley's function*, one of the common benchmarking functions given by Macnish [5], using a Gray coded and a real coded GA (in two dimensions for display purposes). Ackley's function is

$$f(x) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left[\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right] + 20 + e\,,$$

which over the domain $(-6, 6)$ in $\mathbb{R}^2$ is seen in Figure 1. Although not quantitatively comparable, the results demonstrate similar qualitative behaviour when the value of the mutation probability p_{mut} is increased from 0.03 to 0.6. The sample paths displayed are typical, but are still just sample paths and will vary due to initial starting positions and also due to the stochastic nature of the algorithms.

For example, with $p_{\mathrm{mut}} = 0.03$, both the real coded and the Gray coded simulations find the minimum to within ten decimal places under their respective grid spacings. When p_{mut} is increased, in all cases the outcome becomes progressively worse, with the real coded algorithm seemingly performing better at least with these settings on this problem. This essentially highlights the situation that whatever algorithms you employ, they must be tailored to the particular application.

A 37-bit Gray binary code and a ten decimal place real code have been used for comparison. This allows for a comparison in some sense as they both have the potential to realise a zero to ten decimal places. That is, the grid spacing for the Gray code is

$$\frac{6-(-6)}{2^{37}-1} = 8.7311 \times 10^{-11}\,,$$

whereas that for the real code is 10^{-10}. The crossover probability $p_{\mathrm{cross}} = 0.6$ in all simulations.

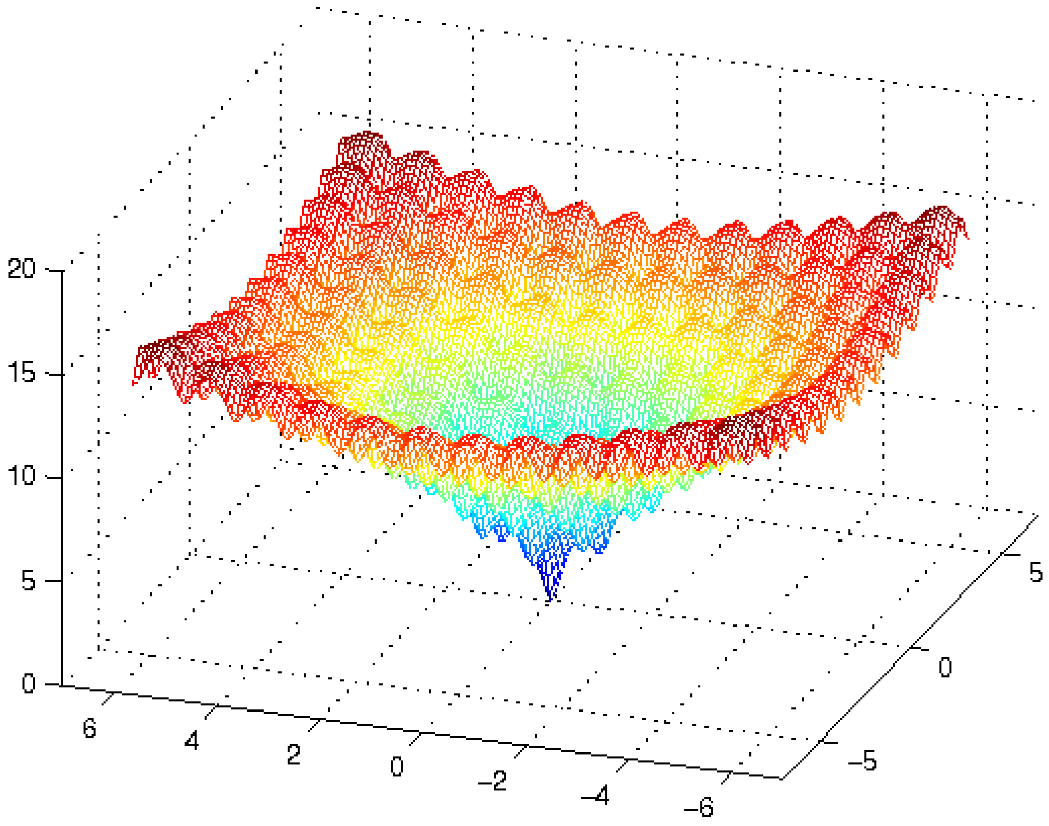

Figure 1: Ackley's function.

Thirty candidate solutions are used and animations of the actual moves (that is, where the solutions actually change) of the first two of the candidate solutions are given overlayed onto a contour map of Ackley's function, which is being minimised. The minimum is located in the centre of the symmetric contour map, in the 'cooler' region (dark blue). Animations of Figures 2, 3 and 4 (corresponding to mutation probabilities $p_{\text{mut}} = 0.03$, 0.3 and 0.6 respectively) are for the real coded strings. These animations must be viewed in an appropriate pdf viewer to use the animation control buttons beneath the plot. Each simulation runs for $10,000$ iterations.

Coordinates (where given in the captions) are only reported to four significant figures in the comparison between the solutions observed for the real coded and Gray coded simulations.

6 Concluding remarks

GAs perform well in engineering applications. However, significant effort may be involved in formulating the problem. The question of whether to code solutions using binary strings or floating point numbers has been contentious in the past. However, this choice is only the beginning of many choices that a GA designer needs to make. Gridspacing (the distance between consecutive numbers) is always an issue.

Our discussion of the algorithms we used for the genetic operators gives an overview of what is needed to implement a GA. The animations of our simulations illustrate how our toy GA performs on a standard benchmark problem.

To make a GA perform well in a complicated engineering application is a complex exercise involving significant questions of initial design and considerable experimentation to determine how to best make the design work. This remains the case irrespective of the method of coding.

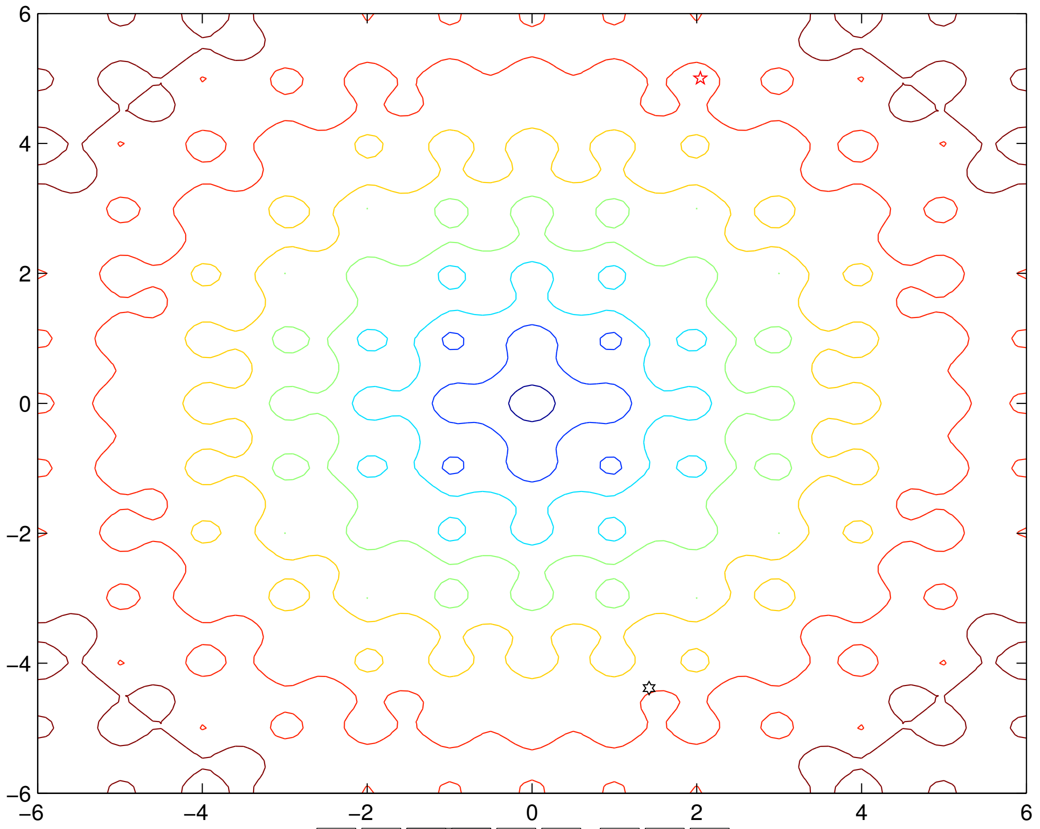

Figure 2: animation with $p_{\text{mut}} = 0.03$. *Real code:* found coordinates of minimum after 88 moves within 178 iterations. *Gray code:* found coordinates of minimum after 136 moves within 2326 iterations. At the time of writing one appears to have to use Adobe Acrobat to view these animations. However, Figure 4 is available as as MPG file at `http://anziamj.austms.org.au/ojs/index.php/ANZIAMJ/editor/downloadFile/2776/9021`

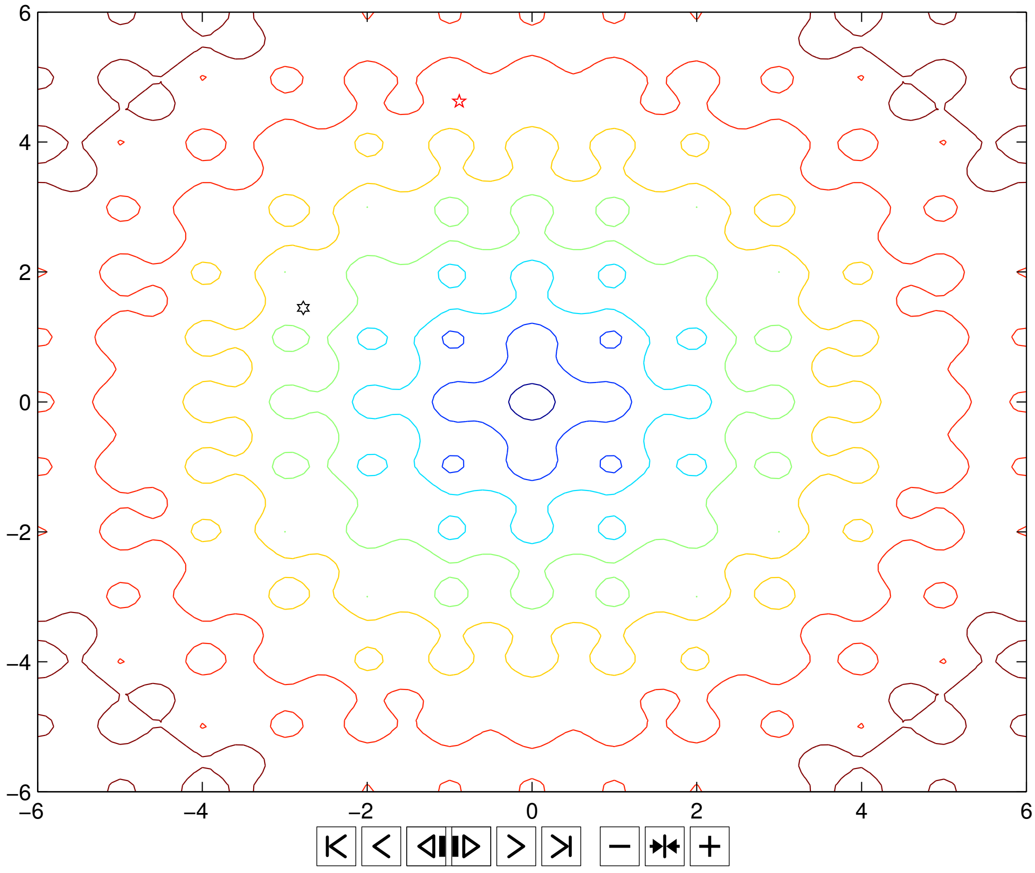

Figure 3: animation with $p_{\mathrm{mut}} = 0.3$. *Real code:* Candidate solution (not minimum) of $(2.28, -787.482) \times 10^{-7}$ after 38 moves within 7037 iterations. *Gray code:* Candidate solution (not minimum) of $(0.0002, 0.0088)$ after 28 moves within 8973 iterations.

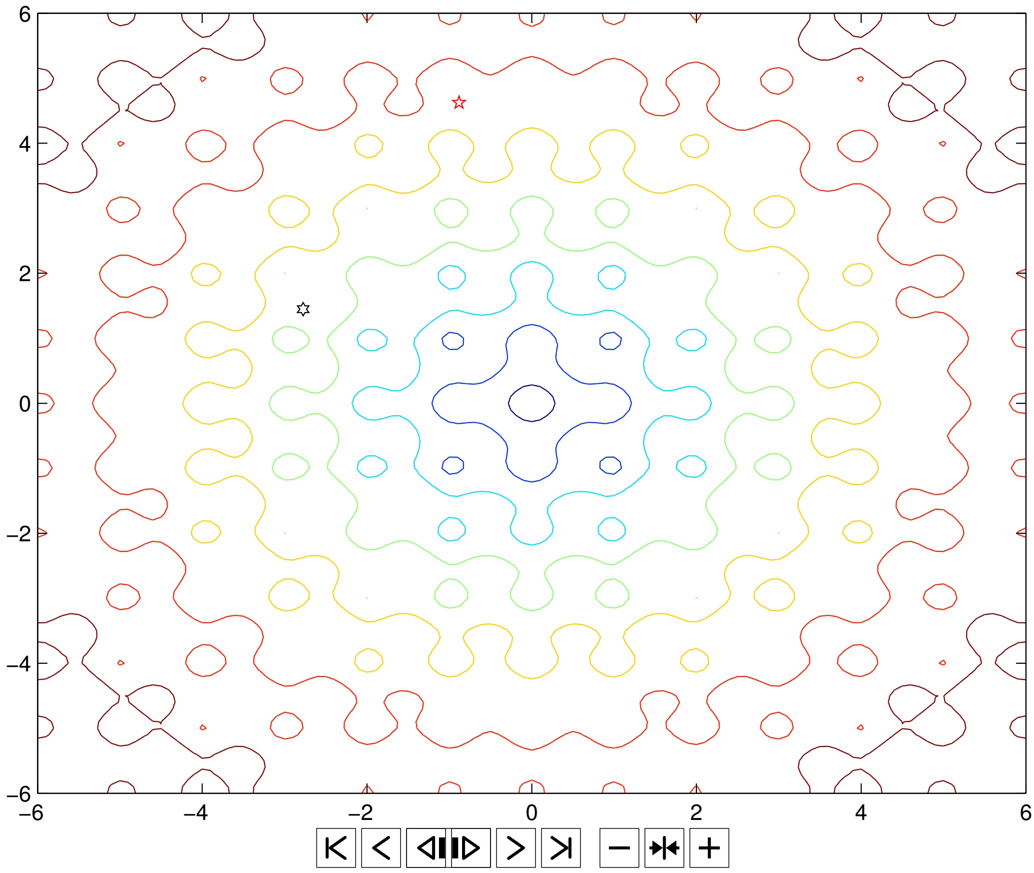

FIGURE 4: animation with $p_{\mathrm{mut}} = 0.6$. *Real code:* Candidate solution (not minimum) of $(-7.4969, 9.0371) \times 10^{-4}$ after 27 moves within 9747 iterations. *Gray code:* Candidate solution (not minimum) of $(0.1064, -0.0985)$ after 18 moves within 9051 iterations.

Thanks to Tony.

References

[1] D. Rani and M. M. Moreira. Simulation–optimization modeling: A survey and potential application in reservoir systems operation. *Water Resources Management*. Springer Netherlands, 2009. http://www.springerlink.com/content/p61p535r2277r852/ C348

[2] John H. Holland. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence. Ann Arbor: University Michigan Press, 1975. http://mitpress.mit.edu/catalog/author/default.asp?aid=3235 C348

[3] Z. Michalewicz. Genetic algorithms + data structures = evolution programs. 3rd edition, New York: Springer–Verlag. http://www.springer.com/computer/ai/book/978-3-540-60676-5 C348

[4] D. Fogel. Evolutionary computation: toward a new philosophy of machine intelligence. *3*rd edition, IEEE Press, 2006. http://ebooks.ebookmall.com/ebook/232338-ebook.htm C349

[5] C. Macnish. Towards unbiased benchmarking of evolutionary and hybrid algorithms for real-valued optimisation. *Connection Science*, **19:4**, 2007, 361–385. http://www.springerlink.com/content/2446104674981574/ C352

Author addresses

1. **J. Gaffney**, School of Mathematical Sciences, University of Adelaide, Australia.
   `mailto:janice.gaffney@adelaide.edu.au`
2. **D. A. Green**, School of Mathematical Sciences, University of Adelaide, Australia.
3. **C. E. M. Pearce**, School of Mathematical Sciences, University of Adelaide, Australia.