Phlegmatic Mappings for Function Optimisation with Genetic Algorithms

Steve Margetts and Antonia J Jones
Department of Computer Science, Cardiff University,
Queen's Buildings, Newport Road, PO Box 916,
Cardiff CF24 3XF, U.K.
Email: S.Margetts@cs.cf.ac.uk
Phone: +44 (0)29 2087 4812

Abstract

The nature of the mapping between the genotype and the phenotype is vital to the performance of a genetic algorithm. We develop the notion of a *phlegmatic mapping function*, which encapsulates the idea that small changes in the genotype ought to produce small changes in the phenotype. After illustrating a mapping function with this property, we validate the idea using a test suite of function optimisation problems. We show that using a phlegmatic mapping can greatly improve the performance of a genetic algorithm for this problem domain.

1 INTRODUCTION

Many problems can be reduced to finding the maximum (or equivalently, the minimum) of a function. The multidimensional function optimisation problem is a direct generalisation of this: given some function f which takes as input the n-dimensional vector $\underline{x}$, find the values for $\underline{x}$ for which f is maximised. In real-world problems, analytical solutions can be very hard to find, especially if f is non-linear, highly multimodal or only partially known. Genetic algorithms [Holland, 1975] are a natural solution to this type of problem, as the function f is an excellent candidate for a fitness function.

The use of genetic algorithms for function optimisation has a long and rich history. Since the beginning of the field, function optimisation has formed the heart of many standard test suites (e.g. [De Jong, 1975], [Michalewicz, 1994], [Fogel, 1995], [Foster, 1995]). More recently, the technique has proved its worth in numerous practical applications (eg. [Obayashi et al., 1999], [Pazos et al., 1999]). However, given that we cannot guarantee that a genetic algorithm will locate the global optimum, we must be willing to accept solutions which are "good enough" for the task at hand.

Naturally, we hope to find the best possible solution in the time available. There is then a continuing research effort to improve our genetic algorithms to accomplish this (within the bounds imposed by the "No Free Lunch" theorem [Wolpert and MacReady, 1996]). The aim of this paper is to investigate the importance of the genotype to phenotype mapping when using genetic algorithms for function optimisation. We will start by looking at a theoretical approach to this problem, and then validate our ideas on a set of test functions.

2 THE GENOTYPE TO PHENOTYPE MAPPING

In general, the structures used in the internal workings of a genetic algorithm (the genotype) are different to those required by the fitness function (the phenotype). This is because the genotype is usually chosen to make the application of the genetic operators (mutation and crossover) as simple as possible, whereas the phenotype is fixed by the problem. Hence when measuring the fitness of a genotype, we must first convert it into a phenotype. This process is shown in figure 1.

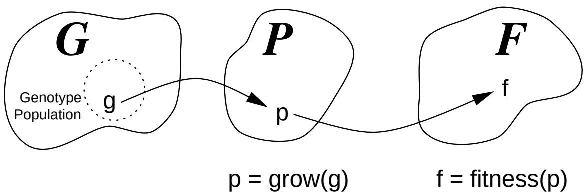


Figure 1: The Genotype to Phenotype Mapping

The diagram shows the relationship between the set of genotypes comprising the population $Pop \subseteq \mathbb{G}$, the set of phenotypes $\mathbb{P}$, and the set of fitness values $\mathbb{F}$. The arrows show a member of the population $g \in Pop$ being converted into a phenotype $p \in \mathbb{P}$ by the mapping function $\texttt{grow} : \mathbb{G} \rightarrow \mathbb{P}$. Only once we have the phenotype p can we apply the fitness function to get a fitness value for g. Note that we normally take $\mathbb{F}$ to be the positive real numbers (i.e. we use $\mathbb{F} = \mathbb{R}^+$).

If we accept that the purpose of a genetic algorithm is to find a genotype with the best possible fitness, then this diagram highlights the importance of the mapping function. In effect, we are trying to use the genetic algorithm to find a genotype for which the *composition* of `grow` and `fitness` is as large as possible. We can therefore expect the choice of the mapping function to have a great impact on the algorithm's performance.

To put this in less abstract terms, let us consider using a genetic algorithm to maximise the n-argument function $f : C^n \rightarrow \mathbb{R}$, where C is some closed and bounded subset of the real numbers $\mathbb{R}$. We want the algorithm to find a vector of length n, which when applied to f gives a result which is as large as possible. In terms of the genetic algorithm, the vector of length n is the phenotype, and f becomes the basis of the fitness function.

If we use a canonical genetic algorithm, the genotypes will be fixed length binary strings. Therefore we need some method of converting such strings into real-valued vectors. Fortunately, transforming a fixed-length binary string into a real-valued vector is only slightly more complex than transforming a string into a single real number. If we represent each component of the vector using N binary digits, giving a binary string of length $N \times n$, then we can construct the vector by repeatedly applying the transformation to each block of N bits in turn. The choice of N depends on the application, but it is easy to see that increasing N by one doubles the number of possible real values we can generate. For this reason, N sometimes referred to as the *precision* or *resolution* of the binary-to-real mapping.

We now need to choose a mapping function. A binary string of length N can take on 2^N different values, which implies that there are $(2^N)^{2^N}$ different mapping functions to choose between: a huge number even for small values of N. As the mapping function affects the behaviour of the genetic algorithm, we clearly need to find some principles to help us decide which of these to use.

3 PHLEGMATIC MAPPINGS

Experimental evidence suggests that as a canonical genetic algorithm progresses, the population tends to "home in" on one particular solution. As it does so, we expect the utility of the crossover operator to decrease, as there is less opportunity for it to combine portions of promising individuals into something novel. When the population reaches this stage, we would expect the mutation operator to become more useful, as it implements a form of local search.

Once a population has reached this stage, the nature of the mapping function from genotype to phenotype becomes critical. We would ideally like a small change in the genotype to cause a small change in the phenotype and hence a small change in fitness[1].

To see why this is a useful principle, consider for a moment the two extremes. Firstly, suppose that a small change in a genotype produces a large change in the phenotype. Clearly *any* optimisation algorithm will have a hard time locating an optimum, as there is very little correlation between a change in the genotype and a subsequent change in fitness. The resulting function acted on by the algorithm is effectively random.

Conversely, suppose that all small changes to a genotype produce small changes to the phenotype. In this case, the performance of the algorithm depends only on the fitness function. The mapping function is "transparent": it does not add anything to the function we wish to optimise.

We are now ready to introduce a formal definition for this idea. We will call those mappings for which a small change in the genotype does produce a small change in the phenotype *phlegmatic*, defined as follows:

Definition 1 *A function $m : \mathcal{X} \rightarrow \mathcal{Y}$ is* phlegmatic *if and only if m is onto and*

$$\forall a, b \in \mathcal{X}, \forall k > 0 \, [d(a, b) \leq k \Rightarrow d'(m(a), m(b)) \leq c.k]$$

where $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$ is a distance metric for the finite set $\mathcal{X}$, and $d' : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$ is a distance metric for the finite set $\mathcal{Y}$.

As the sets we are mapping between may be very different, we use c as a normalisation constant. It is chosen so that the maximum distances returned by d' and d are equal. Note that as both $\mathcal{X}$ and $\mathcal{Y}$ are finite, the

[1]Disregarding for the moment the problem of defining a "small change", our assumption here is that most (non-pathological) fitness functions have enough structure to them to make this causal link possible. Again, caveats about the "No Free Lunch" Theorem apply.

distance measured between any two points from either of these sets must be also finite. If the maximum distances returned by d and d' are d_{max} and d'_{max}, then we set $c = d'_{max}/d_{max}$.

This is similar to the definition of continuity, but modified to better capture our requirements for the finite sets $\mathcal{X}$ and $\mathcal{Y}$. It is also similar to the idea of *causality* [Sendhoff et al., 1997].

As we are interested in mapping a binary string of length N into a real number, we have $\mathcal{X} = \{\mathtt{0}, \mathtt{1}\}^N$. However, instead of specifying $\mathcal{Y}$ to be some closed and bounded subset of $\mathbb{R}$, we will take $\mathcal{Y}$ to be the discrete set $\{0, 1, \ldots M\}$, where $0 \leq M \leq 2^N - 1$. The reason for this is that $\mathcal{Y}$ is supposed finite. Also, we require m to be onto, so we need to choose $|\mathcal{Y}| \leq |\mathcal{X}|$. This is a minor restriction, as it is a simple matter to scale the outputs of m into the desired range.

Now that we have chosen $\mathcal{X}$ and $\mathcal{Y}$, we can specify the distance metrics we will use. We will take d to be the Hamming distance, and d' to be the absolute difference:

$$\begin{aligned} d(a, b) &= \textstyle\sum_{i=1}^{N} |a_i - b_i| \\ d'(a, b) &= |a - b| \end{aligned}$$

We will now examine a few genotype to phenotype mappings in the light of this idea.

3.1 THE TRADITIONAL MAPPING

The "traditional" method of converting a binary string of length N into a numeric value is to use the standard binary-to-decimal conversion:

$$\mathtt{traditional}(s) = \sum_{i=1}^{N} s_i 2^{N-i}$$

where s_i refers to the i^{th} component of the string. As an example, we find that $\mathtt{traditional}(\mathtt{011010}) = 26$. Note that any string of zeros maps to 0, and that the largest possible result is generated by a string of ones, which maps to $2^N - 1$. Using this, we find the normalisation constant c in the definition above to be $(2^N - 1)/N$.

We can represent this mapping function by labelling the vertices of the hypercube in the Hamming space. An example of this is shown in figure 2, for the case where $N = 3$.

We can visualise the effects of a small change in the genotype space by looking at the labels of neighbouring vertices. The diagram shows that the strings 000 and 100 are neighbours, but that their labels differ by 4, over half the possible range for the output. Our informal idea of a phlegmatic mapping is one in which all the neighbours of every vertex have labels which are close in value. But from this viewpoint, it appears that this mapping does not have this property.

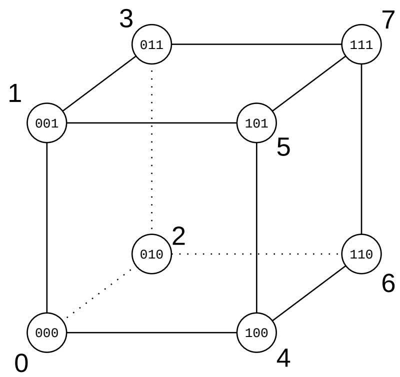


Figure 2: The Traditional Mapping

Proposition 1 *The traditional mapping is not phlegmatic*

Proof: Let $a = \mathtt{00...0}$ and $b = \mathtt{10...0}$, where both a and b are drawn from $\{\mathtt{0}, \mathtt{1}\}^N$. Applying the mapping, we find that $\mathtt{traditional}(a) = 0$ and $\mathtt{traditional}(b) = 2^{N-1}$. Now, the Hamming distance between a and b is 1, so if the traditional mapping were phlegmatic, we could use the fact that a and b differ by a single place to infer that

$$|\mathtt{traditional}(a) - \mathtt{traditional}(b)| \leq \frac{2^N - 1}{N}$$

by taking $k = 1$. We also know that $|\mathtt{traditional}(a) - \mathtt{traditional}(b)| = 2^{N-1}$, so we can substitute this into the above to get:

$$\begin{aligned} 2^{N-1} &\leq \tfrac{2^N - 1}{N}, \text{ so} \\ 1 &\leq 2^N - N2^{N-1}, \text{ hence} \\ 1 &\leq 2^{N-1}(2 - N) \end{aligned}$$

Hence the traditional mapping will only be phlegmatic if $N < 2$. □

Another popular choice for the mapping function is Gray-encoding. We will define this using the traditional mapping:

$$\begin{aligned} \mathtt{gray}(s) &= \mathtt{traditional}(\mathtt{g}_1(s), \mathtt{g}_2(s), \ldots \mathtt{g}_N(s)) \\ &\quad \text{where} \\ &\qquad \mathtt{g}_i(s) = \left(\textstyle\sum_{j=1}^{i} s_j\right) \bmod 2 \end{aligned}$$

Using this definition it is a simple matter to repeat the above argument to show that the Gray-encoded mapping is also not phlegmatic.

3.2 THE COUNTING-ONES MAPPING

Given that neither the traditional mapping nor Gray encoding is phlegmatic, we might wonder if any phlegmatic functions exist at all. In fact, there are many. Consider the following:

$$\texttt{countingOnes}(s) = \sum_{i=1}^{N} s_i$$

This function simply adds all the bits in the string together. It is in fact the archetypal "GA-easy" function, first described in [Ackley, 1987], and used throughout the literature as a simple test function (e.g. [Syswerda, 1989]). A diagram of the function when $N = 3$ is shown in figure 3; this clearly shows that the label for any vertex is simply the Hamming distance from the all-zero vertex.

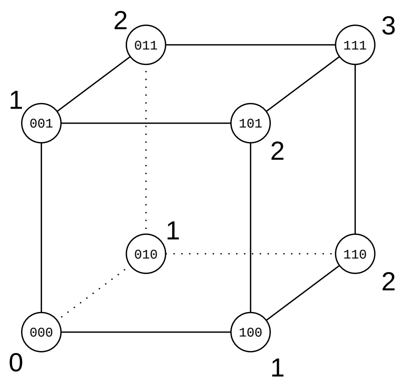


Figure 3: The Counting Ones mapping

The fact that this is a phlegmatic mapping is easily established:

Proposition 2 *The counting-ones function is phlegmatic.*

Proof: Suppose that the Hamming distance between two elements $a, b \in \{0, 1\}^N$ is k. From the definition of the Hamming distance, we know that a and b must differ in exactly k places. Now suppose that in each position i that they differ, $a_i = 0$ and $b_i = 1$, so that a and b differ by the maximal number of ones. Then b must have exactly k more ones than a, i.e. $|\texttt{countingOnes}(a) - \texttt{countingOnes}(b)| = k$. Hence we have shown that for any k, a and b:

$$\texttt{hamming}(a, b) \le k \Rightarrow \\ |\texttt{countingOnes}(a) - \texttt{countingOnes}(b)| \le k$$

The only difference between this and our definition of a phlegmatic function is the normalisation constant c. However, as the maximum value returned by `countingOnes` is N, we have that $c = N/N = 1$. Therefore, the counting-ones mapping is indeed phlegmatic. □

However, this is not the end of all our problems: to achieve the same level of precision as an N-bit string under the traditional (or Gray-encoded) mappings, we would need a string $2^N - 1$ bits long. The mapping is also highly redundant: ${}_N C_r$ distinct strings map to the value r. We will see the practical implications of these features later.

4 BUILDING BLOCKS AND PHLEGMATIC MAPPINGS

Let us suppose that a genetic algorithm operates by composing solutions from "building blocks" made from short, low order schema with high fitness [Goldberg, 1989]. We will consider the portion of a binary string which corresponds to a single input of the test function (i.e. a section consisting of N bits), and examine the effect the counting ones mapping has on building blocks within this region.

We can see that any permutation of this region will give rise to exactly the same phenotype, so all permutations of a given region must have the same fitness. This implies that all permutations of the schema in this region also have the same fitness. Since this region is small in relation to the length of the string, the schemata it contains must be short i.e. they are potential building blocks. So in effect, the counting ones mapping creates classes of equivalent building blocks. We would expect this to aid the search to find the best building blocks within each region. It also allows the algorithm more freedom in choosing building blocks at higher levels, such as those which span many inputs.

5 EXPERIMENTAL COMPARISON

We will compare the properties of the traditional, the Gray-encoded and the counting ones mapping functions on a set of benchmark test functions. Most of these are well-known functions which have been used widely in the literature (eg. see [Shinn-Ying Ho et al., 1999], [Fogel, 1995], [De Jong, 1975]). The test set is given in table 1, and includes unimodal functions (f_3, f_5 and f_{10}) as well as more challenging multi-modal functions (f_1, f_2 and f_4, and f_6 through f_9). While easy for a genetic algorithm to solve, the unimodal functions provide some idea of the global convergence rate of the algorithms tested on them. Each of the test functions is parameterised by n, the number of inputs.

We based our genetic algorithm on the steady-state model (as described in [Goldberg, 1989]), and used the same settings for all experiments. Each used a population of 100 individuals, running for a total of 1000 cycles after initialisation of the population. We used a combined mutation and 2-point crossover operator to create every new individual, and we took the mutation probability per bit to be 1%. To help preserve diversity, each new child generated was compared against the current population. If not unique, the child was repeatedly mutated (using the mutation operator) until it either became unique or a set number of iterations elapsed. The algorithm uses the Mersenne Twister [Matsumoto and Nishimura, 1998] as its random number generator.

We ran two sets of experiments, the first with n, the number of inputs to the problem, set to 10 and the other with n set to 100. For each set we chose the number of binary bits per input N, to be 4 and 10 for the traditional and Gray-encoded mappings, and 10 and 16 for the counting ones mapping. This allows us to compare the two mappings when they both use the same number of bits per input (when $N = 10$) and when the number of values mapped to are equal (when $N = 4$ for the traditional or Gray-encoded mapping and $N = 16$ for the counting ones mapping). Each run was repeated 10 times, and the best values found during the run averaged.

5.1 DISCUSSION OF RESULTS

The results for $n = 10$ can be found in table 2, and those for $n = 100$ in table 3. Both tables are arranged so that the the case where all the mappings have the same number of values to map to are in the first column of results for each mapping. The second column contains the results for the case where each mapping uses the same number of bits. Underlined entries indicate the best-performing mappings in each case.

We will start with table 2, and look at the case where the traditional and Gray-encoded mappings use 4 bits per number, and the counting ones mapping uses 16, so that they all map to the same number of values. Discounting functions f_1 and f_9 (where all mappings performed equally well), we can clearly see that the counting ones mapping outperforms the other two on all of the test functions. It is also interesting to note that the traditional mapping outperforms the the Gray-encoded mapping on all but function f_3.

When all mappings use 10 bits per input (the second column of results in each case), and again discounting f_1 and f_9, we can see that the counting ones mapping outperformed the other two on five of the functions (f_3, f_4, f_5, f_8 and f_{10}). We also find that the traditional mapping outperforms the Gray-encoded mapping on all but f_6.

The poor performance of the counting ones mapping on test function f_6 is rather puzzling. A closer examination of the collected data reveals that with these settings, the algorithm made no progress on the function at all. The reason for this is that with $N = 10$ and a domain range of $[-1, 2]$, the range of values produced by the mapping is $\{-1, -0.7, -0.4, \ldots, 1.7, 2\}$. When we put these into f_6, we can see that the argument to sine will always be a whole multiple of π; i.e *any* binary string will evaluate to zero. This effect also explains the performance of the counting ones mapping on f_1. We stress that this is caused purely by our choice of N: when $N = 16$ the mapping behaves quite differently.

A selection of graphs for these results can be found in figure 4. These show the mean best value found at each cycle of the algorithm. The vertical bars represent 95% confidence intervals for the mean, calculated using the Student's T distribution. The near-invisibility of the trace for the counting ones mapping in the graphs for f_3 and f_8 shows the power of this technique. However, the graph for f_7 demonstrates its downside. Although this mapping starts off well, its performance quickly tails off as the mapping reaches the limit imposed by its reduced resolution.

Moving on to table 3, we will first look at the case where all mappings map to the same number of values. Again, the counting ones mapping comes out on top, outperforming the others on all but three of the functions (f_1, f_4 and f_9). Interestingly, the trend noticed with the 10-input problems is reversed: the Gray-encoded mapping outperforms the traditional mapping on all but f_2 and f_6.

When given 10 bits per number, the counting ones mapping outperforms the rest on all but f_6. However, the poor performance on f_6 is due to the choice of N, as described above. We also find that the Gray-encoded mapping is better than the traditional mapping on all but f_1, f_2 and f_9.

Graphs showing the performance of the mappings on some of these harder problems are shown in figure 5. The main difference between these and the graphs shown in figure 4 is that on the whole, the algorithm was still finding improvements when the run was stopped after 1000 iterations. It is interesting to note that the confidence intervals on the graphs for f_3 and f_{10} are distinct: this amounts to accepting the hypothesis that the mappings are significantly different at the 5% level.

Table 1: The test functions

Test Function	Domain Range	Optimum
$f_1(\underline{x}) = -\sum_{i=1}^{n} \left\| \frac{\sin(10\pi x_i)}{10\pi x_i} \right\|$	[-0.5, 0.5]	0
$f_2(\underline{x}) = -\sum_{i=1}^{n-1} \left[\sin(x_i + x_{i+1}) + \sin\left(\frac{2x_i x_{i+1}}{3}\right) \right]$	[3, 13]	$\approx 2n$
$f_3(\underline{x}) = -\sum_{i=1}^{n} \lfloor x_i + 0.5 \rfloor^2$	[-100, 100]	0
$f_4(\underline{x}) = -\sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10 \right]$	[-5.12, 5.12]	0
$f_5(\underline{x}) = -\sum_{i=1}^{n} x_i^2$	[-5.12, 5.12]	0
$f_6(\underline{x}) = \sum_{i=1}^{n} x_i \left(\sin(10\pi x_i) \right)$	[-1, 2]	$\approx 1.85n$
$f_7(\underline{x}) = -\sum_{i=1}^{n} \left[\sin(x_i) + \sin\left(\frac{2x_i}{3}\right) \right]$	[3, 13]	$\approx 1.216n$
$f_8(\underline{x}) = -\sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 - (x_i - 1)^2 \right]$	[-5.12, 5.12]	0
$f_9(\underline{x}) = -6n - \sum_{i=1}^{n} \lfloor x_i \rfloor$	[-5.12, 5.12]	0
$f_{10}(\underline{x}) = \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) - \frac{1}{4000}\sum_{i=1}^{n} x_i^2$	[-600, 600]	0

Table 2: Test function performance for $n = 10$

Function	Traditional Mapping		Gray-encoded Mapping		Counting Ones Mapping	
	$N = 4$	$N = 10$	$N = 4$	$N = 10$	$N = 16$	$N = 10$
f_1	-3.90×10^{-16}	-3.90×10^{-16}	-3.90×10^{-16}	-3.90×10^{-16}	-3.90×10^{-16}	-3.90×10^{-16}
f_2	15.073	$\underline{15.919}$	13.905	14.827	$\underline{15.127}$	14.512
f_3	-504.000	-91.200	-339.300	-797.300	$\underline{0}$	$\underline{0}$
f_4	-14.278	-15.311	-43.325	-20.379	$\underline{-4.431}$	$\underline{0}$
f_5	-1.065	-0.843	-1.229	-3.194	$\underline{0}$	$\underline{0}$
f_6	13.178	14.225	12.846	$\underline{14.537}$	$\underline{13.281}$	0
f_7	11.811	$\underline{12.033}$	10.952	10.516	$\underline{12.068}$	11.494
f_8	-204.705	-196.361	-271.061	-843.662	$\underline{-1.664}$	$\underline{-23.045}$
f_9	0	0	0	0	0	0
f_{10}	-4.535	-2.526	-5.076	-7.151	$\underline{0}$	$\underline{0}$

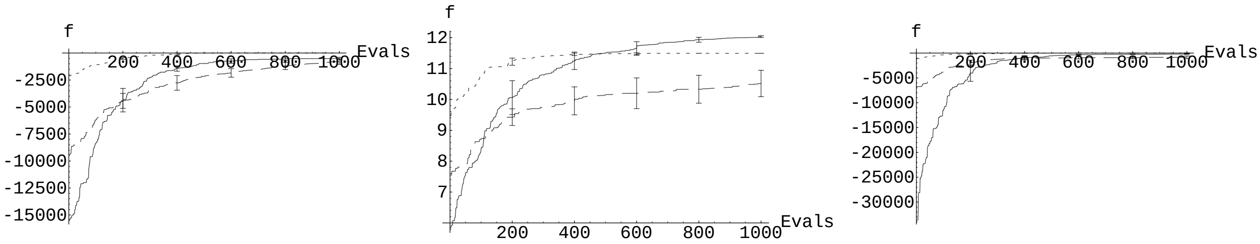
All these observations help strengthen our hypothesis that phlegmatic mappings tend to perform better than the traditional and Gray-encoded mappings on this class of problems, particularly when the number of inputs is large. This adds weight to our hypothesis that mutation becomes increasingly important as the run progresses. The cases where the other mappings are superior are largely explained by the limited resolution of the counting ones mapping: the small range of values available to it appears to prevent it from locating the true optimum.

6 CONCLUSIONS

A phlegmatic function is one for which a small change in the input causes a small change on the output. Using a test suite of function optimisation problems, we have shown that the choice of mapping function does affect the performance of the algorithm, and that using a phlegmatic mapping improves the algorithm's performance in almost all cases. The improvement is particularly striking for functions with many inputs.

Table 3: Test function performance for $n = 100$

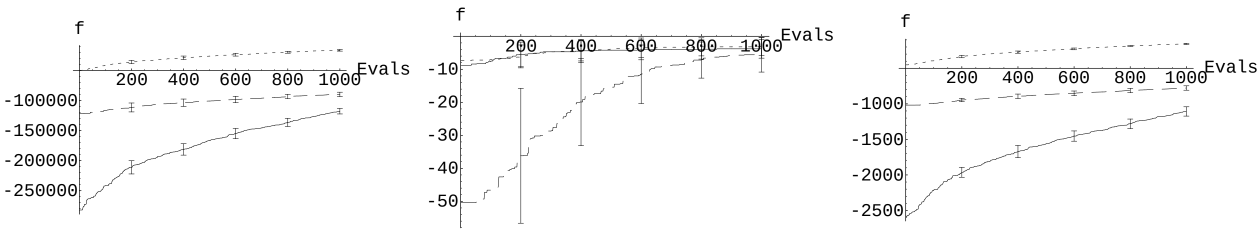
Function	Traditional Mapping		Gray-encoded Mapping		Counting Ones Mapping	
	$N = 4$	$N = 10$	$N = 4$	$N = 10$	$N = 16$	$N = 10$
f_1	-2.57×10^{-3}	-2.57×10^{-3}	$\underline{-4.88 \times 10^{-12}}$	-2.08×10^{-2}	-0.100	$\underline{-3.90 \times 10^{-15}}$
f_2	92.472	85.288	89.706	75.930	$\underline{106.969}$	$\underline{113.111}$
f_3	-117848	-132372	-89561.4	-89851.9	$\underline{-16510}$	$\underline{-24880}$
f_4	-1025.600	-1058.440	$\underline{-496.952}$	-1018.520	-520.861	$\underline{-75.520}$
f_5	-310.600	-338.156	-227.451	-231.014	$\underline{-46.653}$	$\underline{-71.198}$
f_6	101.654	55.361	88.723	$\underline{64.977}$	$\underline{123.897}$	0
f_7	63.807	66.006	96.077	80.674	$\underline{106.338}$	$\underline{99.582}$
f_8	-297279	-355270	-113181	-70367	$\underline{-9470.220}$	$\underline{-21944}$
f_9	-1.400	-3.800	$\underline{-0.700}$	-5.600	-6.300	$\underline{-3.100}$
f_{10}	-1106.870	-1114.570	-803.656	-777.123	$\underline{-157.094}$	$\underline{-244.360}$



(a) f_3 (b) f_7 (c) f_8

—— Traditional Mapping – – – Gray-encoded Mapping ……… Counting Ones Mapping

Figure 4: Selected convergence graphs for $N = 10$ and $n = 10$



(a) f_3 (b) f_9 (c) f_{10}

—— Traditional Mapping – – – Gray-encoded Mapping ……… Counting Ones Mapping

Figure 5: Selected convergence graphs for $N = 10$ and $n = 100$

7 FUTURE WORK

Although we have shown that phlegmatic mapping functions can be very useful in their own right, we have seen that their lack of resolution can cause problems. The nature of the counting ones mapping is that the resolution increases linearly with the number of bits used to represent each input. Hence to gain the same level of resolution as a 10-bit binary string under the traditional mapping, we would need to use 1023 bits per input. This is clearly impractical.

We are tackling this problem using "divide-and-conquer": we start by locating promising areas of the search space, then refine the search by restricting the domain range of the inputs. We will achieve this by using multiple populations, co-evolving in a cooperative fashion in the sense of [Potter, 1997]. Each successive population refines the domain selected by the last, and each uses the phlegmatic mapping investigated here. Our preliminary results are very encouraging.

Acknowledgements

The authors would like to thank Rupert Kapp-Rawnsley for his insightful ideas and enlightening discussions. They would also like to thank the anonymous reviewers for their helpful comments.

References

[Ackley, 1987] Ackley, D. (1987). A connectionist machine for genetic hillclimbing. Kluwer Academic Publishers.

[De Jong, 1975] De Jong, K. A. (1975). *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* Doctoral dissertation, Department of Computer and Communication Sciences, University of Michigan. (University Microfilms No. 76-9381).

[Fogel, 1995] Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press.

[Foster, 1995] Foster, J. A. (1995). *Lecture Notes on Genetic Algorithms*, chapter On Test Suits. IEEE Press.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley Publishing Company Inc. ISBN: 0-201-15767-5.

[Holland, 1975] Holland, J. H. (1975). *Adaption in Natural and Artificial Systems.* MIT Press, 1992 edition. ISBN: 0-262-58111-6.

[Matsumoto and Nishimura, 1998] Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modelling and Computer Simulation*, 8(1):3–30.

[Michalewicz, 1994] Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag.

[Obayashi et al., 1999] Obayashi, S., Sasaki, D., and Takeguchi, Y. (1999). Evolutionary computation of supersonic wing shape optimization. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1791.

[Pazos et al., 1999] Pazos, A., Dorado, J., Santos, A., and Rabuñal, J. R. (1999). Optimization of GA parameters to train recurrent ANN though weight adjustment and selection of activation functions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1793.

[Potter, 1997] Potter, M. A. (1997). *The Design and Analysis of a Computational Model of Cooperative Coevolution.* PhD thesis, George Mason University, Fairfax, Virginia. Supervised by Kenneth A. De Jong.

[Sendhoff et al., 1997] Sendhoff, B., Kreutz, M., and von Seelen, W. (1997). A condition for the genotype–phenotype mapping: Causality. In *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA '97)*, pages 73–80, San Francisco. Morgan Kaufmann.

[Shinn-Ying Ho et al., 1999] Shinn-Ying Ho, Li-Sun Shu, and Hung-Ming Chen (1999). Intelligent genetic algorithm with a new intelligent crossover using orthogonal arrays. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 289–296.

[Syswerda, 1989] Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *3rd International Conference on Genetic Algorithms*, pages 2–9.

[Wolpert and MacReady, 1996] Wolpert, D. H. and MacReady, W. G. (1996). No free lunch theorems for search. Technical report, Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM, 87501. Tech Report SFI-TR-95-02-010.