Constructional Selection, Methylation and Adaptable Representation: Preliminary Findings

Susan Khor
Concordia University
1455 de Maisonneuve Blvd. West, Montreal,
Quebec H3G 1M8, CANADA
slc_khor@cse.concordia.ca

ABSTRACT

This paper investigates the adaptability of genotypes produced by a modified genome growth algorithm (MGG) in a genetic algorithm. MGG uses constructional selection, a genome growth strategy modeled on the evolution of the biological genome [1]. Experiments did not favor the more costly MGG genotypes. This paper also proposes a new type of mutation operator: *meth*, which is loosely based on DNA methylation. Experiments suggest the utility of 'methylation' to improve the adaptability of representations and to build robust and adaptable digital structures.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods and Search**]: Heuristic methods.

General Terms

Algorithms, Performance, Design

Keywords

genetic algorithm, representation, epistasis

1. INTRODUCTION

Adaptive plans were introduced by John Holland [2] as a means to study adaptation in natural and artificial systems. These adaptive plans subsequently became known as genetic algorithms (GAs) [3, 4, 5]. The use of crossover and inversion genetic operators, and of a population of solutions to enable parallel global search distinguishes genetic algorithms from other meta-heuristic search methods. While GAs have had much success in a wide range of applications, theoretical and practical challenges remain. One of these challenges is representation, or what the search space should look like to a GA.

Representation is an important influence on the performance of GAs. Rothlauf [6] reports that when specific representations were used, some problems became easier for a GA to solve, while other problems became more difficult. There are, to our view, three facets to representation which are seldom made explicit in GA literature: (i) encoding, (ii) structure and (iii) directness. Encoding encompasses issues such as binary versus integer or real, and binary versus Gray coding. The structure of a representation may be designed to enable positional independence of the parts of a solution [2, 7], or to designate different parts of a solution to different tasks [8, 9]. A direct solution is one that is plugged into the fitness function directly. An indirect solution is transformed by at least one process which itself may be evolved, and the outcome of the transformation is evaluated by the fitness function of a GA. Examples of indirect representation include grammatical evolution [10] and chemical GA [11].

We deal with the structural aspect of representation in this paper. Specifically, we suggest the utility of a mechanism modeled on DNA methylation which is a form of mutation found in biological genetic systems, to improve the adaptability (the ability to evolve fitter variants) of representations in epistatic fitness landscapes. We call this mechanism *meth*. The effectiveness of *meth* on random genotypes is compared with the adaptability of constructed genotypes which are genotypes selectively grown using a variant of the method suggested by [1]. Results from our experiments suggest that *meth* is effective at improving the adaptability of genotypes with high or low epistasis levels. Further, with *meth*, the average epistasis level of a population of genotypes could increase without severely impacting the mean population fitness. Thus genotypes could grow in diversity and store potential alternative solutions without having to express them. Evolution with *meth* also produced more robust genotypes. These two factors suggest that *meth*, combined with other techniques, would be useful for building complex, robust but adaptable digital structures.

Paper organization: background material is provided in section 2 followed by a description of our modified genome growth algorithm (MGG) in section 3. Section 4 presents results from our experiments with MGG genotypes. Section 5 introduces the *meth* mechanism and reports on experiments with *meth*. Section 6 concludes with a discussion on the current work and possible extensions thereof.

2. BACKGROUND

Adaptive or fitness landscapes having epistasis are rugged with many narrow false peaks (or troughs). Adaptation in epistatic fitness landscapes is not easy because a slight change in a solution may result in a drastic positive or negative change in the fitness of the solution. This makes producing fitter offspring by varying fit solutions found so far, which is essentially how a GA conducts its search, a challenging task. The amplification or reduction of effect is due to existing non-linear dependencies between parts of the solution so that the effect of a gene depends strongly upon one or more other genes. These dependencies may not be avoided entirely if they are *part of the problem*. Besides the potential of advanced search methods including GAs to solve not linearly separable problems or at least nearly decomposable problems [12], is part of their appeal. Nevertheless, given its potential for detrimental effect on adaptability, it is preferable to reduce epistasis *in solutions* where possible.

Holland recognized that epistasis complicates credit apportionment and suggested the search for coadapted sets of alleles which he generalized under the term *schema* [2]. Whereas Holland was confident that linkage between coadapted alleles could be preserved with the inversion operator [2], others made explicit the effort to identify and preserve linkages within building blocks. A building block is a short schema with above average fitness. Kargupta and Park [13] provide a brief history of linkage learning. Learning linkage to avoid destroying linkages and hence preserve good building blocks does not preclude conflict between building blocks [14]. An alternative route is to avoid where possible, placing conflicting building blocks in a solution in the first place, and reduce the epistasis level in solutions. One way to accomplished this is by constructing the representation one building block at a time as suggested by Altenberg's *constructional selection* strategy. Constructional selection is modeled after the evolutionary process of the biological genome [1]. This strategy expands a genome (a solution) one gene at a time until the desired genome length N is reached. A gene is added to the genome only if the addition of the gene improves the fitness of the genome.

The constructional selection strategy is used in the *genome growth algorithm* to produce *constructed genotypes*. If constructional selection (steps 2, 3 and 4 below) is omitted, the genome growth algorithm produces *random genotypes*, the kind normally produced for the initial population of GAs.

The genome growth algorithm is:

1. Add a new gene to the genome.
   - (a) Create a pleiotropy vector. Pick k_i, a random integer in [0, N] then randomly pick k_i other genes to influence the new gene. Let $M = [m_{ij}]$ be an $N \times N$ zero-one matrix where entry $m_{ij} = 1$ means gene g_i influences fitness component f_j. The rows of M are pleiotropy vectors and give the fitness components influenced by a gene.
   - (b) Randomly pick an allelic value for the new gene.
2. Calculate the fitness of the genome.
   - (a) Construct the polygeny vector for each fitness component. The columns of M are polygeny vectors and give the genes influencing a fitness component.
   - (b) Calculate the fitness of each fitness component using the component's polygeny vector and the allelic values of the genome. This is done using the pseudo-random function described in [1].
   - (c) Calculate the fitness of the genome which is the average fitness value of the fitness components (as in Kauffman's NK model [15]).
3. If the new genome fitness is better than the previous, keep the gene. Otherwise, discard the 'new' gene and go back to 1.
4. Adapt the genome constructed so far to new (local) optimum using a "greedy" 1-mutant (using allelic substitution) adaptive walk.
5. If genome length has reached N, stop. Otherwise, go back to 1.

In contrast to the genotype-phenotype (g-p) map for a random genome, the g-p map for a constructed genome showed a distinct pattern of decreasing pleiotropy as the genome lengthened [1]. This happened without there being an explicit selection for low epistasis in the genome growth algorithm. A consequence of decreasing pleiotropy is that fitness levels achieved by fitness components are protected from major degradation as the genome grows. Therefore, constructed genotypes end up with higher fitness values and lower epistasis levels than random genotypes; but they are also more expensive to produce. In section 4, we see if this additional cost is worthwhile.

3. THE MODIFIED GENOME GROWTH ALGORITHM

Experiments in this paper use a modified genome growth algorithm (MGG) in anticipation of employing this method on real applications in which case fitness values for fitness components cannot be generated randomly but would have to be learnt from the problem on hand. MGG takes the following inputs:

1. N, the length of genotypes,
2. $M' = [m'_{ij}]$, a $|G| \times |F|$ *gene-function contribution map*,
3. Formula to calculate fitness of a function, and
4. Formula to calculate fitness of a genotype.

G is a set of genes, F is a set of fitness components or functions. N, $|G|$ and $|F|$ are pair-wise independent. When $N > |G|$, there will be multiple occurrences of a gene in a genotype. MGG ignores duplicate genes so multiple occurrences of a gene contribute only once to fitness calculations. Experiments in this paper use $N > |G|$ and $|G| = |F|$. Specifically, $N = 100$ and $|G| = |F| = 32$.

Inputs 2, 3 and 4 are related, and together they form a challenging hurdle towards the practical application of MGG. These components must capture sufficient knowledge of the problem for MGG to do better than a random search. This issue is outside the scope of this paper. Previous work on linkage learning and function induction mentioned in section 4 are fertile starting points for research on this issue. Experiments in this paper, work on an abstract problem where M' is a matrix with real-value entries. m'_{ij} = r means gene g_i contributes r to function f_j. r may be a positive real, a negative real or zero, in which case g_i does not influence f_j. Experiments in this paper use a gene-function contribution map filled with randomly generated real values in (-2, 2) with 1/3 probability of a negative value including zero. The mix of positive and negative values creates *frustration* and the ruggedness of the fitness landscape. Figures 1A and 1B contrast the adaptive landscapes for random genotypes with lengths 32 and 100, respectively. The jagged nature of the graphs in Figure 1A depicts the epistatic nature of the problem at hand. This ruggedness though is smoothed out considerably in Figure 1B when the genotypes are lengthened to 100, allowing for duplicate genes. The fitness values of genotypes in Figure 1B show less sensitivity to mutation – large declines in fitness are prevented, but so are big fitness improvements.

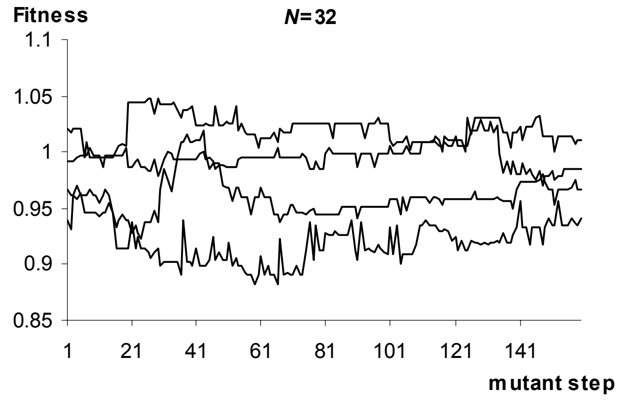

Figure 1A: 1-mutant random walks by four random genotypes, one for each problem (seed). Each genotype has 32 genes. This gives a total of 32 × 5 mutation steps. Each gene is 5 bits. Suppose the first 5 bits of a genotype is 11001. At mutation step 1, this sequence becomes 01001, at mutation step 2, it is 00001, and so on so that at *i*th mutation step, the Hamming distance between the genotype and the original genotype is *i*.

Fitness calculation: Suppose x is the constructed genome so far. Let x' be x with multiple occurrences of a gene removed (compressed x) and x''_j be x' without the genes that do not influence f_j. For example: if $x = \langle 0, 2, 7, 9, 2, 7\rangle$, then $x' = \langle 0, 7, 9, 2\rangle$ and $x''_j = \langle 7, 9, 2\rangle$ given $m'_{0j} = 0.0$. The fitness of a function f_j is calculated as follows:

$$F(f_j) = \frac{1}{|x''|} \sum_{i=0}^{|x''|-1} r_i \text{ where ri = m'x''[i]j.}$$

The fitness of a genotype is calculated using the "standard" (as in the NK-landscape model) method:

$$F(x) = \frac{1}{|F|} \sum_{i=0}^{|F|-1} F(fi).$$

The MGG steps are:

1. Append to genome x, a randomly selected gene g from G.
2. Calculate fitness of genome with new gene, F(x+g).
3. If F(x+g) ≥ F(x) or *tries* = 0, $x := x+g$, i.e. accept g into x. Else decrement *tries* and go back to 1.
4. If $|x| = N$, stop. Else, go back to 1.

tries is the number of chances MGG is allocated to find a suitable gene. In the experiments *tries* = 4.

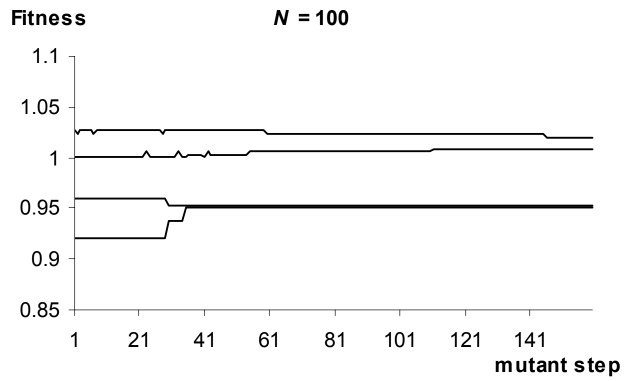

Figure 1B: 1-mutant random walks by four random genotypes, one for each problem. Each genotype has 100 genes, giving a total of 100 × 5 mutation steps. Only the first 160 steps are shown in this figure so that it can be compared with the Figure 1A. Figure 3A shows the full 500 mutation steps.

4. EXPERIMENTS WITH MGG GENOTYPES

The purpose of the experiments in this section is twofold: (i) to compare constructed genotypes (produced by MGG) with random genotypes; and (ii) to see whether using an initial population of constructed genotypes can improve the performance of a GA significantly enough to justify their production cost.

For the first objective, it is essential that MGG genotypes attain higher levels of fitness and lower levels of epistasis than random genotypes, since these are the two elementary properties of genotypes constructed using the genome growth algorithm with constructional selection. The experiments confirm the superiority of MGG genotypes over random genotypes in both aspects.

Tables 1A and 1B show the starting mean fitness and starting average epistasis levels for a population of 100 genotypes from 4 runs, each with a different random seed labeled s0, s1, s2 and s5. This means we tested MGG genotypes on 4 different exercises since each seed creates a different, unrelated gene-function contribution map. The average epistasis level says how many genes influence a function on average. It is measured by finding the average the number of genes that influence each function for each genotype in a population and then taking the average of these averages to get a per function average epistasis level measure. Genes with negative alleles (explained in Section 5) are excluded from this measurement. For the 24 runs reported in Table 1C, the average population fitness for constructed genotypes was on average 239.3 standard deviations higher than the average population fitness for random genotypes.

Table 1A. Average population fitness for one run on each problem at iteration 1

s	random (A)	constructed (B)	Difference (B-A)
0	0.9496	1.1034	+0.1538
1	0.9518	1.0677	+0.1159
2	1.0249	1.1156	+0.0907
5	1.0054	1.0864	+0.0810

Table 1B. Average population epistasis level for one run on each problem at iteration 1

s	random (C)	constructed (D)	Difference (D-C)
0	14.7003	10.0844	-4.6159
1	18.0209	12.1150	-5.9059
2	13.5275	9.2284	-4.2991
5	15.5691	9.8444	-5.7247

Table 1C. Averages and standard deviations of pop. fitness for six independent runs on each problem at iteration 1

s	random	constructed	Number of std devs
0	0.950367 (a) 0.000643 (b)	1.098845 (c) 0.003385	231.0321 (c-a)/b
1	0.951725 0.000536	1.064503 0.00701	210.3592
2	1.024258 0.000441	1.116522 0.003548	209.2485
5	1.00579 0.000257	1.084582 0.001713	306.6033

For the second objective, MGG was used to create an initial population of genotypes for a GA with the parameters set out in Table 2. The performance of this **MGG-GA** is compared with **rGA** which has the parameters in Table 2 but its initial population comprises random genotypes.

Table 2. GA Parameters

Bits per gene	5 ($\log_2 \|G\|$)
Initial population size	100
Population size	200
Crossover rate	0.65
Crossover type	Single point, at gene boundary
Mutation level	1 (default), 3, 5, 7
Mutation rate (P_m)	Mutation level / (N×bits per gene)
Iterations	800
Selection strategy	100 fittest

The next generation is created by first putting the 100 fittest genotypes (the parents) from the current generation into the next generation, then applying crossover (crossover rate × population size times) on these 100 genotypes to produce one offspring each time, cloning random parents to reach the population size and finally applying mutation on the entire next population, i.e.: parents and offspring. Crossover operates only at gene boundaries so that an offspring cannot obtain genes not found in either of its parents. Mutation rate (P_m) is the portion of bits flipped in a population of genotypes. At mutation level 1, P_m = 0.002 and there are 100 × 100 × 5 bits in the current generation, so 100 random bits are flipped when creating the next generation. It may be said that since the next generation (before selection) has twice as many bits as the current or parent generation the mutation rate is effectively halved.

Tables 3A and 3B record the end results of the experiments. Overall, the use of constructed genotypes produced higher fitness and lower epistasis levels, at the end of the runs. What is striking is the inverse relationship between fitness and epistasis levels. However, if we examine the results from the perspective of adaptability and look at the change columns, it appears that constructed genotypes are less adaptable than random genotypes because they show more modest improvements. Considering the estimated cost incurred to produce a constructed genotype (on average (N-1) × *tries*/2 evaluations), the experiments do not depict a convincing argument for the use of constructed (MGG) genotypes as the initial population for a GA. The results in Table 3C corroborate this conclusion. The average population fitness achieved by MGG-GA was on average 0.659 standard deviations higher than the average population fitness level attained by rGA.

Table 3A. Average population fitness for one run on each problem at iteration 800 with P_m = 0.002

s	rGA (E)	Change (E-A)	MGG-GA (F)	Change (F-B)	Difference (F-E)
0	1.0804	+0.1308	1.1897	+0.0863	+0.1093
1	1.0840	+0.1322	1.0872	+0.0020	+0.0032
2	1.1471	+0.1222	1.1977	+0.0821	+0.0506
5	1.1265	+0.1211	1.1349	+0.0485	+0.0084

Table 3B. Average population epistasis level for one run on each problem at iteration 800 with P_m = 0.002

s	rGA (G)	Change (G-C)	MGG-GA (H)	Change (H-D)	Difference (H-G)
0	7.5313	-7.1690	5.0000	-5.0844	-2.5313
1	7.4063	-10.6146	8.3750	-3.7400	+0.9687
2	5.5625	-7.9650	4.0938	-5.1346	-1.4687
5	6.3750	-9.1941	5.9063	-3.9381	-0.4687

Table 3C. Averages and standard deviations of population fitness for six independent runs on each problem at iteration 800 with P_m = 0.006

s	rGA	MGG-GA	Number of std devs
0	1.174115 (d) 0.005065 (e)	1.181833 (f) 0.021875 (g)	1.523951 (f-d)/e
1	1.07977 0.007595	1.088727 0.002877	1.179332
2	1.148392 0.010811	1.148473 0.013932	0.007554
5	1.137517 0.013036	1.13656 0.018231	-0.07339

There are two possible problems with a straightforward use of constructional selection in a GA: (i) the assumption that it is possible to quantify the genome fitness contribution made by any set of genes with any allelic combination, and (ii) there is no (explicit) mechanism to preserve the low epistasis level attained in the initial population through constructional selection, in subsequent iterations of a GA. The first problem is essentially a credit assignment problem and efforts have been made to learn the mapping between genotypes and phenotypes in GAs under various guises [16, 13, 17]. Function induction techniques used in other fields could also be useful here. The second problem is related to the length of schemata. Because the constructional selection strategy considers the entire genome before adding a gene (or more precisely giving an allelic value to a gene if one permits genes without allelic value), each gene in a genome is coadapted to the set of genes already present in the genome so that it would seem that there are few to no loci where crossover can take place without destroying some schema. This raises the question whether crossover is a suitable variation operator in the case of constructed genotypes. In GAs, crossover enhances building block mixing and assists in the search for coadapted sets of alleles in a population of random genotypes. Constructed genotypes are already exposed to a degree of searching and mixing during their genome growth stage. Does crossover help or hinder adaptation of constructed genotypes in a GA? Is a mutation only variation strategy better? The flatness of the graphs in Figures 1B and 3A hints at the answer.

Table 4A. Average population fitness at iteration 800 for s1 at different mutation levels and the effect of a mutation only strategy.

P_m	MGG-GA (I)	Change (I - 1.0677)	mutonly (J)	Change (J-1.0677)	Difference (J-I)
0.002	1.0872	+0.0195	1.0902	+0.0225	+0.0030
0.006	1.0886	+0.0209	1.0708	+0.0031	-0.0178
0.010	1.0441	-0.0236	1.0263	-0.0414	-0.0178
0.014	1.0225	-0.0452	1.0131	-0.0546	-0.0094

Table 4B. Average population fitness at iteration 800 for s5 at different mutation levels and the effect of a mutation only strategy.

P_m	MGG-GA (K)	Change (K-1.0054)	mutonly (L)	Change (L-1.0864)	Difference (L-K)
0.002	1.1349	+0.1295	1.1289	+0.0425	-0.0060
0.006	1.1500	+0.1446	1.1437	+0.0573	-0.0063
0.010	1.0936	+0.0882	1.0876	+0.0012	-0.0060
0.014	1.0735	+0.0681	1.0719	-0.0145	-0.0016

To answer these questions, we ran MGG-GA with and without crossover at different mutation levels. The results from these experiments are summarized in Tables 4A and 4B. We report on s1 and s5 because they performed best and worst respectively in the previous experiment. The results suggest that crossover does help constructed genotypes achieve higher levels of fitness. In all but one instance, MGG-GA outperformed the mutation only (**mutonly**) strategy. Further, genotypes under MGG-GA show more adaptability or less fitness degradation than genotypes under mutonly. The positive changes for MGG-GA are larger and the negative changes are smaller, than mutonly. This indicates that crossover may not be as disruptive as previously thought. The differences in performance increase as the mutation rate ascends. A possible explanation for this is that as in nature, crossing over reduces the accumulation of harmful mutations in individuals.

5. METHYLATION

In this section a new kind of variation operator for GAs, *meth*, is introduced. *meth* is a highly simplified version of DNA *methylation*. DNA methylation is a chemical change to the genome which adds methyl groups to bases of DNA after DNA synthesis and is a reversible form of gene inactivation [18]. The idea for methylation in a GA comes from the following observation: mutation and crossover exert two effects on existing epistasis levels of genotypes. They remove the pleiotropic effects of genes leaving a genotype and add the pleiotropic effects of genes joining a genotype. These two effects happen one after the other with *no chance for a GA to evaluate the intermediate result*. To enable such evaluation, there needs to be an operator which can remove genes and their corresponding pleiotropic effects without adding new genes and their pleiotropic effects. *meth* is this operator. The reverse case: adding new genes and their corresponding pleiotropic effects without having to first remove existing genes and their pleiotropic effects was done during the genome growth stage.

meth is implemented in MGG-GA as a special mutation operator which has the probability of occurring once every one hundred mutations. Its syntactic effect is to negate the allelic value of the target gene. Suppose we are dealing with integral representations, the allelic value for gene g_i is n_i and g_i gets chosen for mutation. If the mutation operator happens to be *meth*, the allelic value of g_i will be $-n_i$ after the mutation completes. Strictly speaking, *meth* also does demethylation, negative value alleles can become positive again. Negative alleles in a genotype suppress the expression of their positive counterparts in the genotype. For example: if $x = \langle 6, -7, 3, 2, 7, -3, 4, -3, 9, 2, 7\rangle$ then the alleles considered in the fitness calculation of x is 6, 2, 4 and 9. The order of appearance in a genotype is not important. A negative allele may appear before or after its positive counterpart. The genes in MGG-GA are position independent.

Table 5A. Averages and standard deviations of population fitness for six independent runs on each problem at iteration 800 with $P_m = 0.006$ and with *meth*

s	r_meth	Number of std devs	c_meth	Number of std devs
0	1.2637(h) 0	17.6882 (h–d)/e	1.2637 (i) 0	3.7425 (i-f)/g
1	1.1936 0.0037	14.9881	1.1777 0.0061	30.9170
2	1.2399 0.0039	8.4622	1.2407 0.0038	6.6199
5	1.2028 0.0082	5.0110	1.2042 0.0033	3.7115
Average		11.5374		11.2478

Table 5B. Comparing average population epistasis levels for one run on each problem using different evolutionary strategies at iteration 800 with $P_m = 0.006$

s	rGA	r_meth	MGG-GA	c_meth
0	6.6847	11.7672	5.7734	12.4416
1	9.2678	13.7853	8.2550	14.8453
2	6.5463	11.3219	6.4447	11.1641
5	6.7688	12.2188	5.8044	12.0309

We experimented with *meth* on rGA and MGG-GA at mutation level 3, which gave the best overall performance in previous experiments. The results are summarized in Tables 5A and 5B.

r_meth is rGA with *meth*, random is rGA (by default without *meth*). **c_meth** is MGG-GA with *meth* and constructed is MGG-GA (by default without *meth*). Both rGA and MGG-GA benefited from *meth*; their runs convincingly attained higher fitness values than runs that did not use *meth*. For instance, on problem s5, the average population fitness attained by r_meth was 5 standard deviations higher than that of rGA. However, very little distinguishes the improvements by r_meth and c_meth over their non-*meth* counterparts (rGA and MGG-GA respectively). The average improvement was 11.54 standard deviations for r_meth, and 11.25 standard deviations for c_meth. Since c_meth uses constructed genotypes which are more costly to produce in the initial stage, we conclude that although *meth* improved the adaptability of constructed genotypes, it also cancelled any adaptive advantage had by constructed genotypes because random genotypes with *meth* did just as well.

Table 6. Population diversity for s5 with P_m = 0.006 at iteration 800

	rGA	r_meth	MGG-GA	c_meth
Range of genotype diversity	18-19	24-29	16-17	24-27
Gene pool (allele) diversity	19	30	17	28
Number of distinct genotypes	2	52	2	27

The r_meth and c_meth columns in Table 5B show higher levels of epistasis. Upon closer investigation, this reflects the greater variety of genotypes and diversity of the population gene pool. Table 6 gives some diversity measurements. Gene pool (allele) diversity is the number of alleles found in a population of genotypes. The gene pool is the reservoir of raw material for future adaptations and its diversity helps prevent suboptimal adaptations or premature GA convergence. Typically, mutation is relied upon to replenish genetic variety loss through directed selection and genetic drift. There are 32 alleles in the experiments; we exclude negative alleles in our variety calculations. That r_meth and c_meth managed to retain at least 87.5% of possible alleles suggest the utility of *meth* as a complement to mutation's role as variety provider. Table 6 also shows that runs with *meth* produced populations with many more distinct genotypes. In a population of 100, r_meth had 52 unique genotypes and the genotype lengths ranged from 24 to 29 distinct alleles. Genotypes are differentiated on the basis of their alleles; the order of alleles is irrelevant. The c_meth population had fewer distinct genotypes (27) than r_meth's but more than constructed which like random had only 2 distinct genotypes.

Table 7 shows the mean population fitness for s5 at the end of runs using different strategies and with different mutation rates. At higher mutation rates, runs with *meth* (r_meth and c_meth) suffer smaller fitness degradation than runs without *meth*. This suggests that evolution with *meth* produces more robust genotypes in terms of withstanding higher levels of mutation. This suggestion is supported by the evolutionary graphs in Figure 2 which depict the progression of each strategy at P_m=0.006. Upon reaching their best mean fitness levels, the graphs of runs with *meth* exhibit long periods of stasis and loose their initial "noisiness".

Table 7. Average population fitness for one run on s5 using different strategies and mutation levels at iteration 800

	P_m	rGA	r_meth	MGG-GA	mutonly	c_meth
(M)	0.002	1.1265	1.2085	1.1349	1.1289	1.2085
	0.006	1.1206	1.2085	1.1500	1.1437	1.2021
	0.010	1.0961	1.2021	1.0936	1.0876	1.2021
(N)	0.014	1.0756	1.1876	1.0735	1.0719	1.1900
(N-M)		-0.0509	-0.0209	-0.0614	-0.0570	-0.0185
Std. dev.		0.0234	0.0098	0.0355	0.0338	0.0077

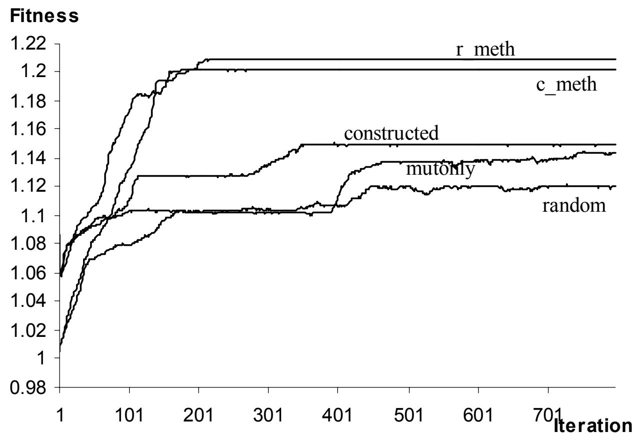

Figure 2: Evolutionary graphs of five runs using different evolution strategies for problem s5 with P_m = 0.006. The best performances were by strategies with *meth*.

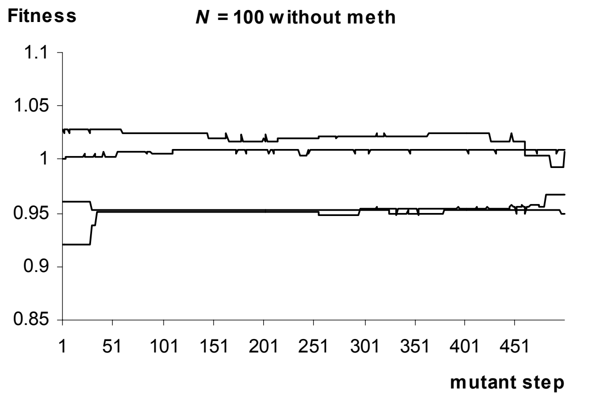

Figure 3A: 1-mutant random walks by four random genotypes, one for each problem. Each genotype has 100 genes, giving a total of 100 × 5 mutation steps.

While more investigation needs to be done, we suspect that *meth* works as a mask, revealing new schema to a GA. Without *meth*, the adaptive landscape as indicated in Figure 3A is rather flat. With no hill to climb or valley to descend, it is conceivably difficult to discriminate genotypes for their potentially useful schema. With *meth*, the adaptive landscape changes dramatically as indicated in Figure 3B. Figure 3B shows how the fitness of a genotype is affected by accumulating mutations, which includes *meth*. The graph in Figure 3B (magnified in Figure 3C for the first 160 steps) is neither as erratically jagged as the graph in Figure 1A, nor as level and smooth as the graph in Figure 3A.

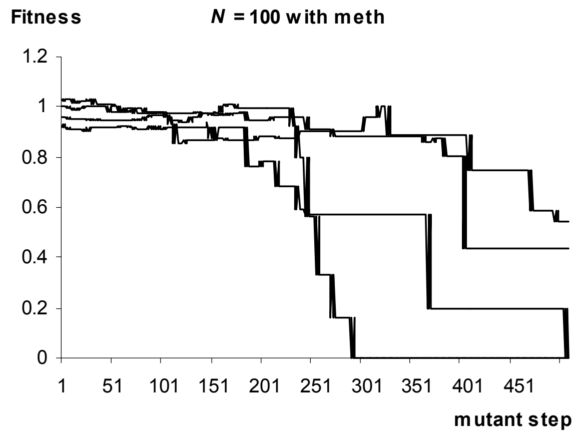

Figure 3B: 1-mutant random walks by four random genotypes, one for each problem. Each genotype has 100 genes, giving a total of 100 × 5 mutation steps. Mutation includes *meth*, negation of an allele, with a 0.01 probability. To generate the graphs in this figure, which are meant to represent changes in fitness as a result of 1-bit mutations, an ordinary mutation, i.e. bit flip, of a negative allele, consists of negating the negative allele. So when *meth* occurs, fitness values oscillate for a 2-5 subsequent mutations. This accounts for what looks like at this scale, stout vertical bars in the graph. Ordinarily, an ordinary mutation of a negative allele consists of replacing the negative allele with a randomly selected positive allele from *G*.

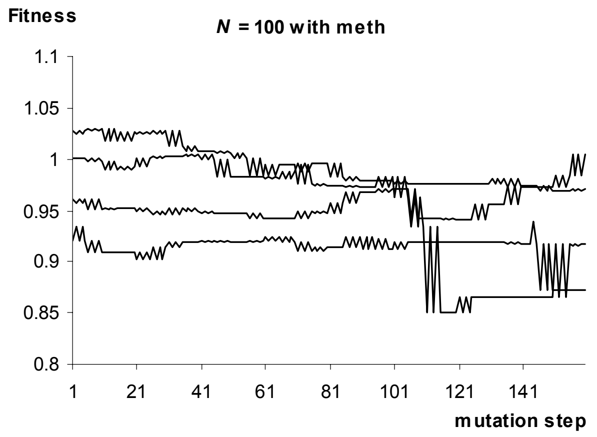

Figure 3C: 1-mutant random walks by four random genotypes, one for each problem. Each genotype has 100 genes, giving a total of 100 × 5 mutation steps. Only the first 160 steps are shown in this figure so that it can be compared with Figure 1B. Figure 3B shows the full 500 mutation steps.

6. CONCLUSION

This paper explored two methods by which adaptability of representations for inherently epistatic problems: constructional selection and methylation. Constructional selection, first proposed by [1], was investigated using our modified genome growth algorithm (MGG). The genotypes produced by MGG bore similar characteristics as that reported in [1], namely higher fitness and lower epistasis. Having established this, we subjected genotypes constructed by MGG to evolution via a genetic algorithm (GA) and compared their adaptability and end results with that of random genotypes. Although MGG genotypes achieved better end results in terms of higher fitness and lower epistasis than random genotypes, their adaptability was sluggish. This left us skeptical about the net benefits of using the more costly MGG genotypes in GAs and we set out to investigate if crossover was a culprit for the dismal adaptability of MGG genotypes. Contrary to our expectations, the experiments suggest that crossover serves two purposes in adaptation for MGG genotypes: accumulation of good sub-solutions and the disbandment of accrued harmful mutations. The link between types of epistasis, whether synergistic or antagonistic, and crossover has been studied in biology [19]. With more details about the relationships between functions, the model proposed in this paper could be used to extend the biological studies to digital organisms.

The epistatic nature of our test problems and hence sensitivity to change, led us to ask how variation of genotypes can be made one effect at a time. By default, crossover and mutation make two changes to a genotype; they remove and insert alleles so that the epistasis level of a genotype is doubly disturbed with larger unpredictable consequences for the fitness of the genotype. Methylation or the *meth* mechanism is a way to reduce such disturbance and make one change at a time to a genotype so that a GA has the opportunity to evaluate the intermediate genotype and possibly increase its *intrinsic parallelism*. The experiments confirm the utility of *meth* for MGG genotypes and random genotypes on our test problems. GA evolution with *meth* produced significantly fitter and more robust genotypes while conserving the diversity of the gene pool for future adaptations. These qualities of *meth* suggest its potential for building complex and robust but adaptable digital structures from random genotypes.

Although MGG genotypes thrived with *meth*, their improvement was not significant enough to justify their higher initial cost. However, this disadvantage may only be so due to the environment of our test problems, which could be extended with additional constraints such as imposing a minimum or maximum threshold on fitness values of some or all functions, adding relationships between functions and creating a hierarchy of functions. In addition, GA evolution began with fully specified genotypes. If partially specified genotypes were allowed to compete and adapt, constructional selection might prove advantageous. In this situation, as in nature, there are no partial solutions; all so-called intermediate stages must be fit to survive and adaptations are explained in terms of "immediate selective advantage" [19] to the individual rather than "the ultimate benefits" they might confer to future individuals. Nevertheless, it seems there are limits to self-assembly in nature and "the larger-scale structures require the differential activation of genes, in time and space" [19] which in artificial systems could be provided in part by *meth*.

There are several roles open to m*eth* in adaptation: as a mechanism to backtrack from a decision made earlier, a way to store alternate solutions for the future and preserve the diversity of the gene pool, a way to regulate genes, a mechanism to effect phenotypic polymorphism [19], or as a "suppressor mutation" in the 'parliament of genes' [19]. To some extent, this paper has exploited *meth* in the last sense. This was unplanned; and an important pre-condition to make this possible is $N >> |G|$. The impact of redundant representation on GAs is discussed in [20]. Runs without *meth* and $N = 32$ produced random and constructed genotypes with higher average fitness values than runs without *meth* and $N = 100$, but these values were lower than the average fitness values produced by runs with *meth* and $N = 100$ (compare Tables 8, 5A and 3C). Average fitness of runs with *meth* and $N = 32$ were also lower than that of runs with *meth* and $N = 100$ by about 6 standard deviations. In each of the four problems, MGG-GA with *meth* produced the best average population fitness. *meth* had a smaller impact when $N = 32$. While the average population fitness was improved approximately 11 standard deviations when $N = 100$, it only improved by about 4 standard deviations when $N = 32$. The many possible roles of *meth* would be an interesting subject for future investigations.

Table 8. Averages and standard deviations of population fitness for six independent runs on each problem at iteration 800 with $P_m = 0.006$ and $N = 32$.

s	rGA	r_meth	No. std devs	MGG-GA	c_meth
0	1.2256 0.0037	1.2331 (j) 0.0053 (k)	5.7736 (h-j)/k	1.2220 0.0059	1.2320 0.0046
1	1.1252 0.0073	1.1545 0.0029	13.4828	1.1253 0.0065	1.1546 0.0046
2	1.1866 0.0053	1.2310 0.0075	1.1867	1.1823 0.0059	1.2311 0.0096
5	1.1738 0.0088	1.1887 0.0037	3.8108	1.1723 0.0038	1.1880 0.0052
		Average	6.0635		

Finally, there are several unexplored areas in the parameter space of MGG-GA. These include: $|G| > |F|$, $|G| < |F|$, making some genes position-dependent and taking into account the contributions of duplicate genes so that the frequency with which an allele appears in a genotype is significant.

7. ACKNOWLEDGMENTS

The author thanks Dr. Peter Grogono for comments on an early draft of the paper. This work received support from NSERC and the Faculty of Engineering and Computer Science, Concordia University.

8. REFERENCES

[1] Altenberg, L, Evolving Better Representations through Selective Genome Growth. In *Proceedings of the IEEE World Congress on Computational Intelligence*, 1994, 182-187.

[2] Holland, J. H. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[3] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[4] Mitchell, M. *An Introduction to Genetic Algorithms*. The MIT Press, 1996.

[5] Fogel, D. B. *Evolutionary Computation: Toward a new philosophy of machine intelligence*. IEEE Press, 1995.

[6] Rothlauf, F. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag Heidelberg, 2002.

[7] Goldberg, D. E., Korb, B. and Deb, K. Messy Genetic Algorithms: Motivation, Analysis and First Results. *Complex Systems* 3:493-530 (1989).

[8] Kargupta, H. The gene expression messy genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1996, 814-819.

[9] Kvasnička, V. and Prospíchal, J. Emergence of Modularity in Genotype-Phenotype Mappings. *Artificial Life* 8:295-310 (2002).

[10] O'Niell, M. and Ryan, C. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer Academic Publishers, 2003.

[11] Suzuki, H., Sawai, H. and Piaseczny, W. Chemical Genetic Algorithms: Evolutionary Optimization of Binary-to-Real-Value Translation in Genetic Algorithms. *Artificial Life* 12(1):89-116 (2005).

[12] Simon, H. A. *The Sciences of the Artificial*. MIT Press, 1969.

[13] Kargupta, H. and Park, B.-H. Gene Expression and Fast Construction of Distributed Evolutionary Representation. *Evolutionary Computation* 9(1):45-68 (2000).

[14] Yu, T.-L., Sastry, K. and Goldberg, D. E. *Linkage Learning, Overlapping Building Blocks, and Systematic Strategy for Scalable Recombination.* IlliGAL Report No. 2005016, University of Illinois at Urbana-Champaign, 2005.

[15] Kauffman, S. A. Adaptation on rugged fitness landscapes. In *Lectures in the Sciences of Complexity*. Ed.: D. Stein, Addison-Wesley, 1989, 527-618.

[16] Goldberg, D. E., Deb, K., Kargupta, H. and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*. Ed: S. Forrest, Morgan Kaufmann Publishers, 1993, 56-63.

[17] Pelikan, M, Goldberg, D. E., Ocenasek, J. and Trebst, S. *Robust and Scalable Black-Box Optimization, Hierarchy and Ising Spin Glasses*. IlliGAL Report No. 2003019, University of Illinois at Urbana-Champaign, 2003.

[18] Campbell, N. A. *Biology*. Third Edition. Benjamin-Cummings Publishing Company, Inc., 1993.

[19] Maynard-Smith, J. and Szathmáry, E. *The Major Transitions in Evolution*. W.H. Freeman & Co. Ltd., 1995.

[20] Rothlauf, F. and Goldberg, D. E. Redundant Representations in Evolutionary Computation. *Evolutionary Computation* 11(4):381-415 (2003).