Genet Program Evolvable Mach (2006) 7:131–144
DOI 10.1007/s10710-006-9001-y

Finding needles in haystacks is harder with neutrality

Mark Collins

Received: 2 October 2005 / Revised: 5 January 2006 / Published online: 1 July 2006

Abstract This research presents an extended analysis of the reported successes of the Cartesian Genetic Programming method on a simplified form of the Boolean parity problem. We show the method of sampling used by the CGP is significantly less effective at locating solutions than the solution density of the corresponding formula space would warrant. We present results indicating that the loss of performance is caused by the sampling bias of the CGP, due to the neutrality friendly representation. We implement a simple intron free random sampling algorithm which performs considerably better on the same problem and then explain how such performance is possible.

1. Introduction

This paper investigates the expected and actual success rates of the Cartesian Genetic Programming (CGP) algorithm as reported by Yu and Miller in [8]. The CGP system has gained attention following its performance on several well known benchmarks including the Boolean parity problems, a classic test problem in both Evolutionary Algorithm and Evolutionary Hardware research.

We compare the performance of the CGP algorithm reported in [8] to the performance of a CGP style random walk algorithm and a random re-sampling algorithm, using the same problem definition, parameters and representation. We were unable to repeat the same performance as reported for the CGP algorithm, but are able to show that all (1 + 4) CGP based sampling methods—including the unrepeated reported performance—are biased and underperforming in terms of solutions per sample when the solution density is considered. How the addition of automatically defined functions—as in recent CGP work [6]—alters this situation is unknown.As will be discussed later in this work, the focus on the problem specific

M. Collins (✉)
Centre for Intelligent Systems and their Applications, School of Informatics, University of Edinburgh, Edinburgh, EH8 9LE, Scotland, UK
e-mail: s9740861@sms.ed.ac.uk

performance of the CGP algorithm—successful or otherwise—is misleading, however, the implications of thebias in the sampling should not be underestimated.

We also implement and test a trivial random sampling algorithm which uses a fully expressed genotype. We show that this algorithm considerably exceeds expectations, performing better than random sampling with intron expression. An analysis of how simply removing introns from the representation can produce such a result is provided.

2. Previous work

Previous work by Langdon and Poli [3] into the structure of solutions to Boolean problems, including a special form of the parity problem using a reduced operator set of only the XOR EQ operators, indicated the solution density in the space of formulas tended to a limit. Hereafter this special form of the Boolean parity problem which is solved using only EQ and/or the XOR operator will be referred to as the 'reduced Boolean parity problem' to distinguish it from the version with unrestricted logical operators.

Yu and Miller used the reduced Boolean parity problem as a test case for the CGP algorithm. CGP used a representation which supports introns, the belief being that the introns and the'neutrality' which they permit could be exploited by the search algorithm.

Previous work [1, 2] has provided and empirically proved the accuracy of a method for counting the number of solutions to the reduced Boolean parity problem in the space of all possible formulas. This gives a method for calculating the number of solutions which are in existence in the space sampled by the algorithms. This work supported the findings of Langdon and Poli and gave exact results for the solution density and the limit. For convenience the critical sections of these works are included.

3. Extensions to the original investigation

Following the original publication of this work at GECCO 2005, we have had the opportunity to improve our sampling and repeat with greater computational investment the experiments we previously published. We have also had the benefit of the valued contributions of Julian Miller who advised how to improve the experimental range—particularly with regard to studying different forms of output selection in CGP. The general CGP algorithm allows evolution of the choice of output gate, hereafter referred to as the *active point*, it is equally possible to fix the active point at the last gate in the representation, both possibilities are considered in this work.

In the light of the opportunities offered, we have thus re-presented our original work alongside work obtained more recently. The primary difference is in the quality of the graphs obtained, which are now considerably smoother. The basic findings of the work remain the same, but are now explicitly demonstrated to be applicable to both forms of the CGP interpretation. The terminology used in this work has also been clarified. To avoid confusion the term *intron* is now used only for sections of genotype that are not expressed in the phenotype. The cancellation of expressed effects in the phenotype is simply referred to as cancellation.

4. Reported performance

The CGP algorithm has performance reported for the 5, 8, 10, and 12 parity problems. We will concentrate on the 12 parity problem, since this is (a) the hardest of the reduced Boolean parity problems for which CGP results are reported, b) Yu and Miller reported the failure of random sampling on this problem see ([8] page 5, Table 1), and, c) Yu and Miller obtained an outstanding >55% success rate from a hundred runs, each of a mere 10,000 iterations. We use only reported results for comparison in this investigation.

5. Representation issues

In the Boolean parity problem the input alphabet I of size $|I|$ is a set of distinct Boolean values $\{I_1, \ldots, I_{|I|}\}$, corresponding to the parity of the problem. In the reduced Boolean p-parity problem, the input alphabet is p elements, and the function set is either $\{\text{XOR}, \text{EQ}\}$ or simply only $\{\text{EQ}\}$, the size of the function set is denoted by ϕ. The number of possible arrangements using from 1 up to F_{Max} functions is:

$$\sum_{i=1}^{F_{\text{Max}}} \phi^i p^{i+1} \tag{1}$$

6. Solution structure

The commutative and associative properties of the XOR, EQ operators create equivalence relations between candidate solutions and a set of functionally equivalent representations. For instance I_3 XOR I_1 EQ I_2 is rearranged to give I_1 EQ I_2 XOR I_3, which whilst potential distinct entities in the representation space, are identical in functional terms.

Solutions occur for this form of the Boolean parity problem iff all Boolean inputs are referenced once in the equivalent simplified function representation. All solutions to a p-parity problem must at least contain p input references. Duplicate references to inputs cancel and vanish from the simplified functional form: XOR and EQ are both commutative and for any Boolean input I_x, I_x XOR I_x = FALSE and I_x EQ I_x = TRUE. Even numbers of references to the same input either invert or leave intact the exact functionality of the other functions; arrangements of this type can not alter the acceptability of the function in terms of distinguishing even and odd parity. Finally choosing between the XOR and EQ functions is actually irrelevant to the acceptability of the solution. XOR is simply the negation of EQ and instances of it can be replaced by NOT EQ without altering the function of the expression. Negation of a result is irrelevant to the identification of parity, since itsimply changes an even parity function into an odd parity function and vice versa.

Using the fact that even numbers of references to the same input cancel and are neutral to the output validity, the number of solutions possible to the p-parity problem which reference n inputs, $S(p, n)$, can be counted by separately calculating the number of arrangements of the $n - 1$ functions used and multiplying this by the number of input sequences which meet the requirement of referencing each input once and only once in the simplified representation. There are $n - 1$ functions, each of which can be one of the ϕ function types, the number of function arrangements is then $\phi^{(n-1)}$. The number of input sequences which generate a reference to each input only once in the simplified representation is denoted by $I(p, n)$.

$$S(p, n) = \phi^{(n-1)} I(p, n) \tag{2}$$

The number of input arrangements which can be simplified to reference each input only once is best counted by ignoring the p inputs which must be distinct for a solution, and counting the number of ways the remaining $n - p$ inputs can be arranged to cancel. Evidently such cancelling arrangements can only exist if there are an even number of the

$n - p$ remaining references, and further each of the inputs referenced by this surplus must be referenced an even number of times by the surplus.

The number of ways the n input references can make a solution to the p parity problem is then given in two steps. The first step calculates the ways cancelling arrangements can be made from the surplus inputs. This is the number of ways that pairs of the $n - p$ surplus input references can be assigned to groups where all pairs in the group have the same input reference and no two groups have the same input reference. The second step involves multiplying the number of cancelling arrangements by the number of distinct permutations of n input references.

Enumerating the groups is performed by generating the set of integer partitions of the $(n - p)/2$ pairs. As an example the integer partitions of four are: $\{\{4\},\{3,1\},\{2,2\},\{2,1,1\},\{1,1,1,1\}\}$ and represent the number of ways cancelling arrangements can be made from the XOR EQ operators with 8 input references; as an octuple $\{4\}$, a hextuple and a pair $\{3,1\}$, two quadruples $\{2,2\}$, a quadruple and two pairs $\{2,1,1\}$, and as four pairs $\{1,1,1,1\}$ respectively.

The number of solutions which are present in a partition π, denoted $N(\pi)$ is the number of arrangements of groups to the partition, $A(\pi)$, multiplied by the number of distinguishable input assignments to the n inputs which have this partition arrangement $D(\pi)$.

$$N(\pi) = A(\pi)D(\pi) \tag{3}$$

Using 3-parity and the partition arrangement $\{2,1,1\}$ as an example, the number of ways of choosing the input for the first element of the partition is simply $\binom{3}{1}$. The second and third elements of the partition are identically sized and can not be distinguished, also one input reference has been used in choosing the assignment for the first partition, so the number of ways of assigning inputs to the second and third elements is the unassigned input alphabet choose two: $\binom{2}{2}$. The total number of arrangements of the partition is $A(\{2, 1, 1\}) = \binom{3}{1}\binom{2}{2} = 3$. To be specific, the basic labellings for the 3-parity 2,1,1 partition (those made from one quadruple and two pairs using an alphabet of three) are $\{\{1111,22,33\},\{2222,11,33\},\{3333,11,22\}\}$. The possible permutations of the partition labeling are considered in Eq. (6). In order to generalise this process we require a utility function (a form of Kroneckers delta function) to count the number of distinct elements of a particular size in the partition: Let the function $C(\pi, s)$ represent the number of partition elements of size $0 < s \leq I/2$ in the partition π, and let$|\pi|$ represent the number of elements and π_i represent the ith element in the partition π.

$$C(\pi, s) = \sum_{i=1}^{|\pi|} (\pi_i = s) \tag{4}$$

$$A(\pi) = \prod_{i=1}^{|\pi|} \binom{p - \sum_{j=0}^{i-1} C(\pi, j)}{C(\pi, i)} \tag{5}$$

Any solution containing the cancelling arrangement represented by the partition π thus contains $\pi_i + 1$ references to the input assigned to π_i. These $\pi_i + 1$ assignments are indistinguishable giving:

$$D(\pi) = \frac{n!}{\prod_{i=1}^{|\pi|} (2\pi_i + 1)!} \tag{6}$$

And so the total number of cancelling arrangements possible for the p-parity problem using the XOR, EQ operators is:

$$I(p,n) = \sum^{\forall\pi} N(\pi) \tag{7}$$

The number of solutions possible for all possible functions referencing up to a maximum of n inputs, is then:

$$\sum_{i=p}^{n} S(p,i) \tag{8}$$

Which for some number of function choices ϕ gives the full equation as shown in Eq. (9).

$$\sum_{i=p}^{n} \phi^{(n-1)} \sum^{\forall\pi} \left\{ \frac{n!}{\prod_{a=1}^{|\pi|}(2\pi_a+1)!} \prod_{b=1}^{|\pi|} \binom{p - \sum_{c=0}^{b-1}\sum_{b=1}^{|\pi|}(\pi_b = s)}{\sum_{d=1}^{|\pi|}(\pi_d = s)} \right\} \tag{9}$$

7. The CGP representation

In the reported CGP representation the genotypes are composed of sequences of integer triples, each of which represents a unit equivalent to a reprogramable gate or cell. The first integer represents the gate type, though the reported work on the 12 parity problem only used one gate type. The second and the third integers represent the connections to either one of the 12 parity inputs or the outputs of previous gates. The output of the gate is then the result of applying the operation represented by the gate type to the inputs referenced by the gate. The values of the input references are limited to enforce a feedforward connectivity which with a fixed interpretation order removes the requirement for a clock and prevents looping. There are two forms of interpretation for a given CGP representation, which differ in how the final output is selected; the output gate being either freely chosen from the range of gates, or fixed as the output of the last gate. The final output is determined by selecting the output gate and tracing back the connections to previous gates and inputs, which are then evaluated in the logical order.

Figure 1 shows a simple CGP arrangement corresponding to the genotype (EQ,A,B)(XOR,1,C)(EQ,D,E)(XOR,2,F)(EQ,3,G) (interpreted from unit 5) where letters represent references to the p inputs and numbers represent references to the output of previous units. The actual active expression of this genotype is D EQ E EQ G. The output from the fourth unit corresponding to the expression A EQ B XOR C XOR F is unused.

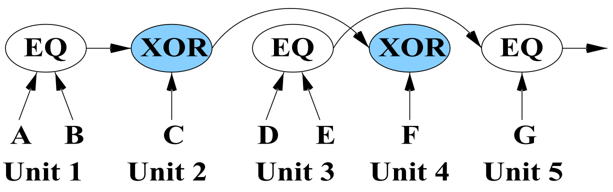

Fig. 1 Interpretation of a CGP genotype is dependent upon the unit selected to generate the output and the subsequent linkage of the units. Here the output is chosen from the last unit.Where units are not linked to the output unit (here marked with colour) they are introns and candidates for neutral mutations

The significant feature of the CGP representation is that the genotype implicitly supports 'neutral' mutations. Introns in CGP are sections of the genotype which are left out of the interpretation due to simply not being included in the chain of references from the chosen output gate. Mutation of the introned code has no effect on the expression of the genotype or the phenotype and is termed 'neutral'. Neutral movement in the genotype is hypothesised to aid search by allowing small mutations to accumulate to create large and complex genotype changes. These changes can then be easily brought into and removed from the phenotype by relatively small mutations in referencing code.

Since the CGP representation has introns as an enevitable part of its representation, it has a bias towards representing smaller solutions more frequently than a representation in which all the code was expressed: There is no one to one mapping from the formula which are represented by CGP solutions to the space of possible formula. The question is, does this help or hinder? As a benchmark for researching neutrality the reduced Boolean parity problem is actually a strange choice—due to the fitness function, all movement except that which finds a solution is score neutral. Differences in performance between the CGP and other tactics on the same landscape must then be a function of the effect of neutrality on the sampling strategy, and not due to permitting score neutral movement.

8. The context of this investigation

The aim of this investigation is to determine if the CGP representation is better or worse at generating solutions than one would have a right to expect from a given solution density.

Previous work [1, 2] has provided calculations for the number of solutions present in the space of possible functions. For Boolean formulas using between 1 and 100 operations, the solution density of the reduced Boolean 12 parity problem space is 0.0019531 percent for two operator types, and 0.003756 percent for one operator type. Since the reduced Boolean search space is devoid of fitness gradient clues, and consequently search is unguided, this represents the best expected performance for an algorithm which samples the space without bias.

In [8] Yu and Miller report CGP has a peak success rate of over 55% on the reduced Boolean 12 parity problem. The average success rate is approximately 45% for the optimal combination of parameters. These results are achieved using the CGP algorithm over 10,000 iterations, which is a sample of at most 40,000 points.

A search using one operator type (EQ), sampling evenly from the search space of possible formulas until finding a solution, performing 40,000 samples has an expected success rate of $1 - (1 - 0.00003756)^{40000} = 0.778$. This contrasts with the empirically obtained expectation of just over 0.55 which was reported for the $(1 + 4)$ CGP algorithm. The difference in performance is then a consequence of the mapping imposed by the CGP implicitly neutral representation and the CGP search tactics.

9. The effect of an implicitly neutral representation

The CGP representation method encourages a simple graph structure, in which functional sequences are established by chaining outputs of earlier units to inputs of later ones. Enevitably some sections of the genotype are not interpreted, and it is these sections in which neutral mutation is expected to occur, though due to the scoring function of the reduced Boolean parity problem, all mutations except those which create solutions are score neutral. Using some of the genotype to represent introns shortens the expressions which can be composed for any given genotype length. This in turn has an interesting effect on theability of CGP style representations to sample the reduced Boolean formula space effectively.

9.1. Candidate generation in CGP

Irrespective of the quantity of the genotype that is expressed, all CGP candidates have a genotype which is fully defined for all units. Assuming the use of a suitable random selection method, the following proceedure may be used to create a candidate solution to the reduced Boolean parity problem in the CGP representation:

```
set the output of the expression to either:
   the output of a randomly selected unit, or
   the output of the last unit

for each unit, do
   set the type of the unit to:
       a randomly selected gate type
   set each of the input connections to:
       a randomly selected choice from all the
       previous gate outputs and the initial
       input lines
done
```

The CGP population strategy is a $(1 + \lambda)$ evolution strategy with $\lambda = 4$ samples being created by mutating the sole parent. Samples replace the parent if certain criteria regarding active genotypic difference and score are met, (see [4, 5, 7, 8] for details). In the reduced Boolean parity problem, the score is always the same unless a solution is found. As reported by Yu and Miller in [8], relaxing the active genotype similarity constraint does not impair search performance.

Figures 2–4 show the selection bias is persistant in common variations of the CGP algorithm. As Fig. 2 shows generating more offspring in each iteration does not smooth out the sampling. The sampling bias is of greater significancewhen it is taken into account that there are vastly more lengthly formulas than there are short.

By increasing the length of the representation used, more promising areas of the formula space can be sampled by the CGP algorithm. Figure 3 shows the distribution of samples as performed by the CGP algorithm, and by random sampling of the 1000 unit CGP representation space.

Notice that no 1000 unit representation has a frequent larger effective size than 200 units (the largest record in the experiments graphed is 212 units)—a vast amount of the space is not sampled. Increasing the representation length from 100 to 1000 units moves the peak in unit activity from 10 units to 48. Moving the peak in unit activity to 48 units reduces the amount of samples which are wasted on sampling formulas less than 11 operators in length. This has a noticable effect on the success rate of the algorithm, the 1000 unit representation is considerably more effective than the 100 unit representation.

Instead of allowing the algorithm to halt upon success, by running the 100 unit CGP (1 + 4) algorithm continuously for 10,000 iterations (40,000 samples), fixing the active point at the end of the representation and replacing successful formulas with randomly generated formulas; we were able to boost the CGP success rate to 18 solutions from 30 repeats (60%)

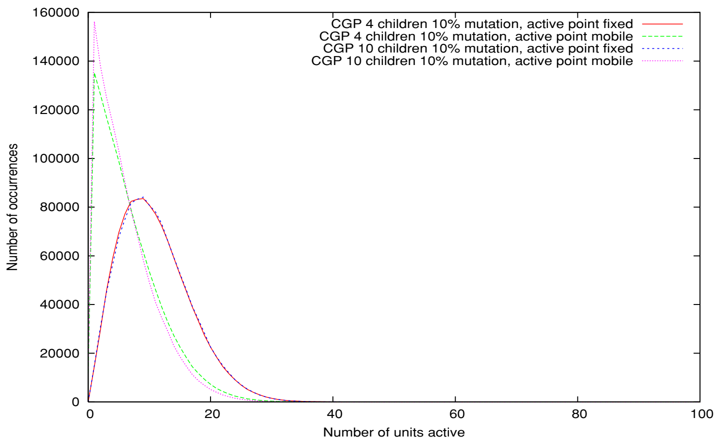

Fig. 2 How the number of children (4 or 10) alters the distribution of samples, in terms of the number of gates active, accumulated over 30 runs of the CGP algorithm

on the 12 parity problem. The success rate under the same circumstances with a mobile active point is 20%. Obviously this method of measuring successhas a potential success rate of more than 100%, indeed the non-CGP random sampling method described in section 9.4 has a success rate of 800% from the equivalentof 30 repetitions (240 successes from 120,000 iterations).

9.2. Randomly sampling CGP representations

The significance of the inheritability of traits from the parent to effectiveness of the CGP search can be tested by comparing the performance of CGP to that of an algorithm which simply samples from the CGP representation space.

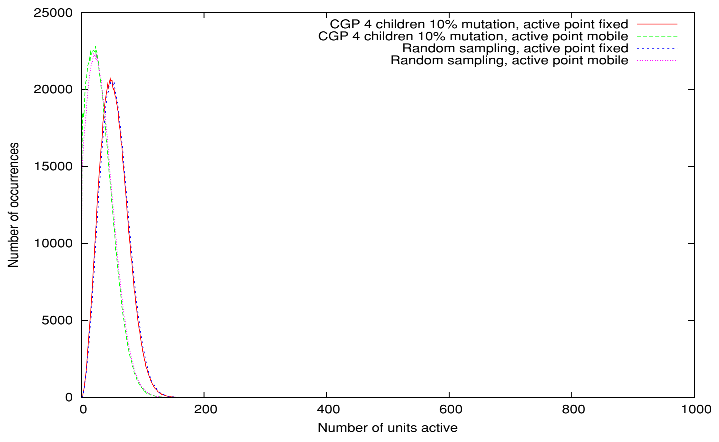

Fig. 3 The distribution of samples for representations using 1000 units, in terms of the number of gates active accumulated over 30 runs of the CGP algorithm. Results shown are from the CGP (1 + 4) sampling method and random sampling from the CGP representation space

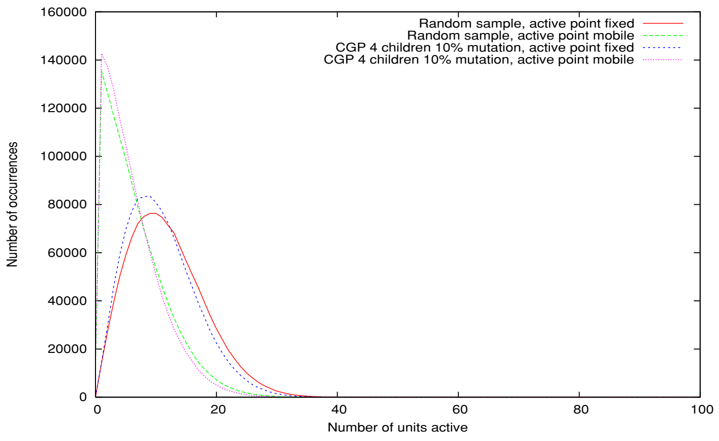

Fig. 4 The distribution of samples in terms of the number of gates active. The results shown here are accumulated over 30 repetitions of each of 40,000 samples. Two results are shown: Independently randomly selected samples from the CGP representation space, and samples from 30 runs of a standard setting of the CGP algorithm: (1 + 4) with a 10% mutation rate

The random sampling of the CGP representation shares essentially the same function length distribution as the sampling by the CGP algorithm. Contrasting the distribution of the random samples from the CGP space and that of the CGP algorithm illustrates the difference in the distribution caused by the CGP algorithm accumulating chain fragments during the run. Figure 4 shows the distribution of samples generated at random using the CGP representation plottedon the same graph are the equivalent number of samples generated by repeated runs of the CGP algorithm with a standard parameter setting. Each case is repeatedwith a mobile active point and a fixed active point. The distribution of samples is largely independent of the CGP search method and appears to be a consequence of the representation.

9.3. An 'average' CGP candidate

To understand why the distribution of active gates is so skewed in the CGP representation, we will consider the construction of an idealised 'average' candidate to tackle the 12 parity problem. This candidate will have 100 units and the active point will be fixed at the last (100th) unit. Tracing back the inputs to the 100th unit, there are 111 possible connections that the inputs may be connected to (99 other units and 12 original inputs), from which the connection is chosen with uniform probability.

An input connection to the 100th unit will on average be attached to the output of the 43rd or the 44th unit. In turn, the 44th unit will on average connect to the 16th unit, and so on down the chain. The expected sequence of active units is then 100, 44, 44, 16, 16, 16, 16, 2, 2, 2, 2, 2, 2, 2, 2, with the remaining inputs being taken from the 12 original inputs. This gives a total of 15 active units, sampling an input sequence of 16 inputs. This is an artificial 'average' candidate, the selections are only representative of the expected region in which inputs will be sourced. The exact selections are subject to the normal variation expected for a uniform random sample. Consequently this particular 'average' candidate is somewhat unlikely, but serves to illustrate the point that the effect of chaining such sequences, at least initially, is to skew the sample of active units.

To control the chaining of units CGP has a parameter setting called "levels back" which sets a maximum limit on how far from the current unit the inputs may be. The use of the levels back parameter can be seen to impose a maximum distance which the chain can jump, or conversely it imposes a minimum number of units which must be active for any given active point. Fixing the active point at the end of the representation and ignoring the levels back parameter creates the expected chain as discussed earlier (which is based upon a log_2 sequence). Using the levels back parameter alters the range that governs the expected chain sequence (e.g. a levels back setting of 10 units gives an expected chain jump of 5 units for the 'average' candidate). Thus the levels back parameter strongly alters the expected number of active units. The levels back parameter is fixed prior to run-time. This single choice determines if the intially placed CGP samples are likely to be in the vicinity of solutions, which in turn alters the success prospects of the run. In most recent CGP work the levels back parameter has been ignored.

9.4. Randomly sampling non-CGP representations

Replacing the CGP representation with one that does not permit introns allows us to compare the effect of using a CGP representation with a representation which does not permit 'neutral' genotype changes. All changes to the genotype in such representation will be interpreted, though it does not necessarilly follow that they will alter the actual phenotype.

The following method generates candidates which sample from the space of possible reduced Boolean formulas. The chaining property, where the output of one unit is fed into the next, must be maintained if introns are to be avoided. This is achieved here by marking the reference immutable, which allows the similarity between the representation generation methods to be clearly seen. In practice this is most simply done by implicitly coding the structure and using the genotype to specify only the function type and the input reference(s) for each unit.

```
select a random expression length L

set the output of the expression to:
   the output of the unit L.

for each unit up to L, do
   set the type of the unit to:
       a random gate type
   set one of the input connections of the unit to:
       the previous gate (make this immutable)
   set the other input connection to
       a randomly selected input line
done
```

Notice that this sampling method is also not 'fair'—it over samples short functions as well, however, it allocates approximately the same number of samples to each of the possible function lengths, and as a consequence has a much better success rate, we recorded 0.020683% success over 100 million trials. This is equivalent to expecting a solution to be located every 5000 trials.

9.5. Uniformly sampling the formula space

For formulas using up to 100 operators of one type the actual solution density is approximately 0.003756%. For each sample of 40,000 independently selected formulas this gives an expected solution occurence of 1.502 per 40000 samples. Thus an algorithm which samples uniformly from the space of formulas and stops when it finds a solution has an expectation of success of 0.778. The simple sampling strategy has an expected success rate of around 0.9997 under the same conditions.

The apparent ‘overperformance’ of the random sampling is entirely due to the distribution of the samples it uses, in particular the manner in which it avoids sampling the very largest formulas in favour of shorter formulas. Uniform sampling of the space would result in a large proportion of the test cases being drawn from the set of formulas of length 100 operators, since so much of the formula space is composed of these formulas. All formulas which reference an odd number of inputs (such as those with 100 operators) are not solutions to the reduced Boolean 12 parity problem. Hence the vast majority of the search space is occupied by formulas which are not solutions. Algorithms which are capable of sampling long formula lengths less than a uniform sampling method but still maintain sufficient sampling of all areas to detect solutions are likely to yield better than random results.

10. CGP on this domain

This paper concentrates on the various forms of CGP on the reduced Boolean parity problem. This problem domain has the peculiar property of having no fitness gradient. Consequently studying the CGP algorithm on this landscape gives insight into the uninfluenced behaviour of the algorithm. The algorithm naturally biases towards short samples. Without further influence the alogrithm will not explore the majority of the space.

The bias in the placement of the initial CGP samples strongly prejudices against creating a function of more than a certain length. If the fitness gradient is missing or begins with functions larger than this size the CGP algorithm has a low probability of reaching the fitness gradient. This creates a class of functions for which the CGP algorithm has a vanishingly low probability of success, irrespective of the density of solutions in the space: under such circumstances CGP can not be considered to be a general problem solver.

At the same time CGP is also reported to have much better performance than other genetic programming techniques [4]. These results are direct comparisons against published benchmark performances. This reinforces the difference between the main objective of this paper and its superficial interpretation: This paper exposes the existance of sample bias in the CGP algorithm, which for the reduced boolean parity problem reduces the algorithms effectiveness. However it is important to realise the same bias could be beneficial on different problems.

11. CGP in other domains: A hypothesis

It is hypothesised the very same bias which caused CGP to underperform on the reduced Boolean parity problem may be responsible for the increased performance reported in [4].

CGP behaves differently in a landscape with fitness: a fit candidate with a certain number of units active will persist (and be subjected to mutations) until replaced by a better fitness candidate. These mutations may lengthen the chain of active units, which will then persist if there is a corresponding improvement in fitness: It is then feasible for a CGP candidate to have the active proportion of the representation extended by the process of evolution if there is a continuous fitness gradient through which such movement could be rewarded.

The same reasons that explain why CGP underperforms on the reduced Boolean parity problem define a class of functions on which CGP would be expected to perform well: Those functions with fitness gradients which are solved by locating and combining small fragments. For a given size of function, a particular function arrangement has a certain probability of being found in a fair sample of the formula space, all unbiased search methods are expected to sample the arrangement at the same rate. A CGP sample on the other hand, due to its more frequent sampling of shorter formula lengths is much more likely to saturate the sampling of certain length functions.

Thus the sample bias can be of benefit if it allows the earlier discovery of basic function arrangements which are on a climbable fitness gradient. Under these circumstances the CGP algorithm would be well placed, purely as a by-product of its biased initialisation. For the CGP to successfully complete the search, the fragments would have to be chained into longer active lengths, which would have to be maintained against the sample bias. This requires a sequence of lengthening operations each with an associated fitness improvement. Due to the constraints on this work, this hypothesis has not been investigated in this extension.

12. Discussion

A brief review of the work presented: Random sampling the CGP representation fails to locate any solution to the 12 parity problem [8], in the same work it is reported that search using the $(1 + 4)$ CGP mechanism has obtained 55% success in 40,000 samples. Under the same circumstances an unbiased sample from the space of possible formulas is expected to have a 77% success rate, and a simple (but also biased) random sampling from a non-intron representation has a 99% success rate. In this instance the CGP representation is biased against a successful search.

Recall Fig. 4, which shows the length of the chain of active units. The initial CGP sample is composed of randomly created CGP candidates and expresses shorter formulas than those sampled by an iterated CGP mechanism. When iterated without a fitness gradient CGP performs mutation and random selection of these formula chains, candidates are exposed to random chain growth and destruction. This process tends to an equilibrium, the chains reach a meta-stable point when chain lengthening caused by randomised growth processes is balanced by the increased probability of longer chains being broken.

The experiments show that the chain lengths are tightly clustered near the meta-stable point, concentrating the sampling effort. The vast majority of samples are considerably shorter than the range of possible chain lengths, long chain lengths have very low probabilities of being generated. Successful use of CGP search is dependent upon the coincidence of this frequently sampled range of formula lengths and the lengths of the solutions. Without a mechanism to encourage the generation of long chains, CGP would not be expected to succeed in locating a solution to any problem that requires a long chain length, no matter how trivial the actual solution.

The differences between this work and that reported in [8] are consequences of the CGP representation. We have the ability to map and count the solutions to the parity problem. By recording the progress of the search we can show the CGP sampling is biased against the search succeeding. It appears that the Yu and Miller work reported in [8] was performed entirely within the CGP representation space. The work reported here uses a comparison against either a map of the space of solutions or against a non-intron representation. Without the benefit of this comparison, the bias in sample distribution which is inherent when using the CGP representation may not have been visible to the experimenter.

13. Conclusion

One of the key features of the neutrality argument is that a representation which permits neutrality in genotype changes does not hinder nor worsen performance on the problems where neutrality is not of benefit. One of the findings of the Yu and Miller paper studying the reduced Boolean parity probem is that they found neutrality did not impair performance in those cases where it did not give improvement. The performance of the CGP algorithm on the reduced Boolean 12 parity problem seems to indicate that the asymmetry in the intron mappings does in fact cause damage to the ability of the algorithm to sample the space effectively.

In this work we have presented evidence that the CGP results as reported in [8] are actually worse than true random search. We have shown that the cause of this performance deficit is present in all common forms of the CGP algorithm. We have also hypothesised that the improved performance of the CGP algorithm on other domains is in part due to the same sampling bias and not necessarilly due to the role of neutral gene sequences. This serves to highlight that the effect of neutrality in evolution is not as benign as it might first appear, and reinforces the case for a full understanding of the test problems on which algorithms are evaluated.

This work is not intended to be read as a critique of the work of Yu and Miller, but as a warning using CGP as an example: Neutrality in a representation introduces search bias, and the use of a biased search mechanism must be justified.

Acknowledgments For their invaluable contributions during my work, my gratitude to: Richard Carter, Jacques Fleuriot, Michelle Galea, John Levine, Julian Miller, Dave Robertson and Henrik Westerberg. Sincere thanks are also due to the reviewers

References

1. M. Collins, “Counting solutions in reduced boolean parity,” in *GECCO 2004 Workshop Proceedings*, R. Poli et al. (eds.), Seattle, Washington, 26–30 June 2004.
2. M. Collins, “Monte carlo sampling and counting solutions in reduced boolean parity,” Technical Report, November 2004. EDI-INF-RR-0240.
3. W.B. Langdon and R. Poli, “Boolean functions fitness spaces,” in *Late Breaking Papers at the Genetic Programming 1998 Conference*, J. R. Kozai, (ed.), University of Wisconsin, Madison, Wisconsin, USA, 22–25 July 1998. Stanford University Bookstore.
4. J.F. Miller, “An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Wolfgang Banzhaf et al. (eds.), Morgan Kaufmann, 1999, vol. 2, pp. 1135–1142.
5. J.F. Miller and P. Thomson, “Cartesian genetic programming,” in *Genetic Programming, Proceedings of EuroGP'2000*, vol. 1802 of LNCS, Riccardo Poli et al. (ed.), Springer-Verlag, Edinburgh: 15–16 April 2000, pp. 121–132.

6. J.A. Walker and J.F. Miller, “Evolution and acquisition of modules in cartesian genetic programming,” in *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, Maarten Keijzer et al. (eds.), vol. 3003 of LNCS, Springer-Verlag: Coimbra, Portugal, 5–7 April 2004, pp. 187–197.
7. T. Yu and J. Miller, “Neutrality and the evolvability of boolean function landscape,” in *Genetic Programming, Proceedings of EuroGP’2001*, J.F. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A.G.B. Tettamanzi, and W.B. Langdon (eds.), vol. 2038 of LNCS, Springer-Verlag, Lake Como, Italy, 18–20 April 2001, pp. 204–217.
8. T. Yu and J.F. Miller, “Finding needles in haystacks is not hard with neutrality,” in *Genetic Programming, Proceedings of the 5th European Conference*, J.A. Foster et al. (eds.), EuroGP 2002, vol. 2278 of LNCS, Springer-Verlag: Kinsale, Ireland, 3–5 April 2002, pp. 13–25.