Embracing Plagiarism: Theoretical, Biological and Empirical Justification for Copy Operators in Genetic Optimisation

S. McGregor (sm66@sussex.ac.uk) and I. Harvey (inmanh@sussex)
Centre for Research in Cognitive Science, University of Sussex, UK

Abstract. A novel genetic operator, the *plagiarism operator*, is introduced for evolutionary design and optimisation. This operator is analogous in some respects to crossover and to biological transposition. Plagiarism is shown to be theoretically superior to uniform mutation for generalised counting-ones problems, and also to outperform uniform mutation on certain classes of random fitness landscapes. Experimental results are presented showing that plagiarism speeds up the artificial evolution of certain digital logic circuits. The performance of this operator is interpreted in terms of the non-uniform distribution of genetic primitives in good solutions for certain problems.

Keywords: Genetic operators, Boolean logic, evolutionary design, translocation, transposition

1. Introduction

This paper discusses some findings in relation to the identification of useful primitives during evolution. The use of repeated structures in artificial evolution has proved valuable in a number of domains (e.g. ADFs, CGP-ADFs, symmetrical robot controllers). The n-bit even parity problem in digital logic synthesis is a commonly-used test problem for novel evolutionary methodologies. However, little investigation of the reusable-component principle has been done at the single-primitive level. We describe a simple, novel, genetic *plagiarism* operator similar to non-homologous crossover between a genome and itself. The plagiarism operator simply copies a genetic primitive from one locus to another. There are some analogies for this process in biology.

The paper is organised as follows: a background section summarises work on reusable components in GP, transposition and translocation in nature and artificial evolution, and the discovery of useful gate-level primitives in digital logic evolution. This is followed by a methodology section describing the genetic encoding, evolutionary algorithm and target problems used in this study. A section on *post hoc* and *ad hoc* discovery of useful primitives introduces the plagiarism operator. Empirical results are presented indicating that this operator is beneficial in the evolution of digital logic circuits for binary arithmetic. The statistical properties of the plagiarism operator are analysed in terms

of its exploration of the *distribution of primitives* space. This analysis is supported by experiments using an unbalanced plagiarism operator. Plagiarism is compared analytically to uniform mutation for the counting-ones problem and some generalised variants of that problem, and shown to be superior. Finally, the search performances of idealised hill-climbing and random walk algorithms using both plagiarism and mutation are compared on random fitness landscapes.

The paper's purpose is to illustrate the practical value of using copy operators in relevant evolutionary domains and to analyse this phenomenon in terms of copy operators genetically exploring a different space (distribution-of-primitives space) than conventional mutation does. We see the use of copy operators as complementary to other evolutionary techniques rather than constituting an alternative.

2. Background

A number of researchers e.g. [5, 8, 19] have applied artificial evolution to the design of Boolean logic circuits. These are circuits made out of binary logic gates such as AND, OR or NOT gates. Such research is often regarded as belonging to the field of *evolvable hardware*, but in fact Boolean circuits can also be regarded as purely idealised mathematical entities. Unlike many other sorts of electronic circuit [17, 16], the physical characteristics of the medium can be effectively ignored for Boolean circuits.

2.1. Transposition & Gene Duplication

In biological evolution, copying errors can result in genetic duplication, inversion or transposition. Some transposons are prone to being copied into other locations as well as merely transferred. Although transposons in some respects function like genetic parasites, they are thought to have played an important role in human genetic evolution [2]. Gene duplication occurs in nature and is postulated to be a major evolutionary mechanism [10], though of course there are important relevant differences between artificial and biological genomes: for instance, the significance of gene location on the genome.

Some evolutionary computing studies already use translocation and/or transposition as a genetic operator [13, 14, 12, 20]. These studies have focused on transfer rather than copying but of relevance here is Urbanek's [18] finding that translocation was ineffective on an analogue circuit design problem in which genetic loci were semantically unrelated; we would expect the same to be true of the plagiarism operator.

2.2. Gene Reuse in Artificial Evolution

The principle of *reuse*, i.e. repeating the use of valuable structures, seems to occur both in human endeavour (e.g. *subroutines* or *classes* in programming) and in nature (e.g. *vertebrae* in the vertebrate spine or indeed *cells* in multicellular organisms). One common method for implementing this principle in artificial evolution is an *automatically defined function* (ADF) or similar scheme (see, e.g. [5, 15, 11, 9, 1, 21]). However, these schemes are not geared towards the reuse of single primitives; indeed, many of them explicitly prohibit automatically-discovered subroutines from consisting of a single primitive.

2.3. Finding Primitives

In [8], Miller et al. evolved functionally correct circuits and then analysed them *post hoc* both automatically and by eye to try to discover useful design principles. Of relevance here is their method for evolving a 3-bit multiplier circuit. They began by evolving 2-bit multiplier circuits with an unconstrained set of gates, and then observed that one of the evolved solutions was rather elegant and used only 3 different types of gate: AND, XOR, and AND with one input inverted. They then evolved a 3-bit multiplier using only these 3 types of gate as primitives.

3. Methodology

3.1. Genetic Encoding

We used a very simple *Cartesian Genetic Programming* encoding following Miller [8]. This encoding, or one very much like it, has been extensively used in previous work [6, 7, 19]. The genome encodes a fixed array of gates indexed by integers. Each gate comprises a *gate function* (one of 16 possible) and two *input indices*. The input indices specify which gates and/or circuit inputs are inputs to a given gate. The genome also encodes a fixed number of *output connection indices* which specify gates to draw the entire circuit's outputs from. Because the circuit is feed-forward, gate x's input indices must refer to a gate with a lower index than x, or to a circuit input. Typically, many gates specified in the genome do not connect to the circuit's outputs either directly or indirectly, and these gates are not expressed in the circuit's phenotype. See figure 1 for an example.

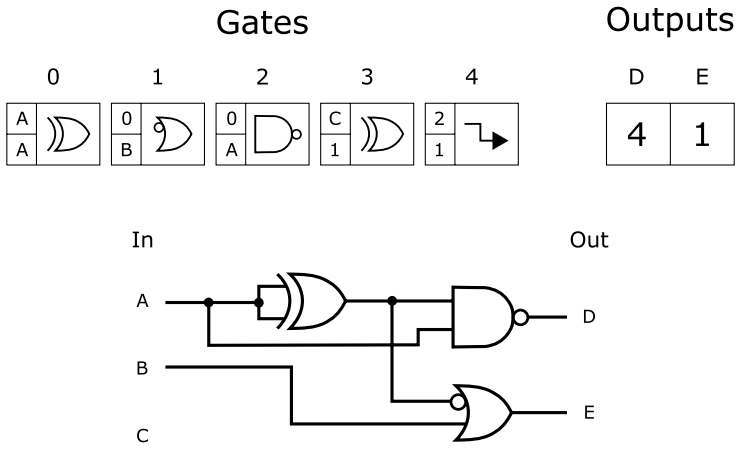

Figure 1. The genetic encoding used, on an array of only 5 gates. In actual evolution the array would be of length 30 or more. The genome is shown above the circuit it encodes. Gate functions are in effect characters in an alphabet. Note that gate 4 is a trivial gate which copies one of its inputs to its output. Gate 4 is still considered *expressed* in the phenotype, unlike gate 3 which is *unexpressed*.

3.2. Genetic Algorithm

The algorithm used was a simple 1+1 mutation-based generational one with no crossover, effectively equivalent to hill-climbing with neutral exploration. Previous work [7] had indicated that narrow, deep searches are effective in digital logic evolution. Each generation consisted of the elite individual and one mutated offspring. If the offspring's fitness was less than the parent's, it was discarded; otherwise, it replaced the elite individual (parents were replaced with equally fit offspring to facilitate neutral exploration). Evolutionary runs were terminated when a functionally correct circuit was found, or after a maximum number of generations (i.e. fitness evaluations) was reached. This maximum number varied with the test problem. The mutation operator operated as follows: - the genome was divided into locus types depending on the locus's interpretation in the phenotype. *Gate function* loci were mutated by setting the locus randomly to one of the 16 possible binary gate functions. *Input index* loci were mutated by setting the locus randomly to a legal input (a circuit input or previous gate output). *Output connection index* loci were mutated by setting the locus randomly to any gate output.

Evolutionary fitness was equal to the number of correct bits in the evolved circuit's truth table. Mutation rate was set to an average of 5 loci per genome, following preliminary experiments on the 1-bit adder.

3.3. Target Problems

Circuits were evolved to perform basic binary arithmetic functions: the inputs to a circuit are interpreted as two binary numbers, and the outputs are interpreted as one or more binary numbers which are some arithmetic function of the input. For example, a *2-bit multiplier* is a circuit with 4 binary inputs (representing two 2-bit binary numbers in the range 0-3) and 4 binary outputs (representing one 4-bit binary number in the range 0-16). Circuits were also evolved to be *even n-parity* calculators. This type of circuit has n binary inputs and 1 output; the output is *true* if the total number of *true* inputs is even, and *false* otherwise.

A feed-forward binary logic circuit can be described by a *truth table*, which exhaustively lists the desired circuit outputs for each possible combination of inputs. Hence, a truth table has 2^n rows where n is the number of circuit inputs. The truth tables used were as shown in table I.

Table I. Truth tables used as target functions for digital logic circuit evolution, along with the CGP geometry and maximum number of generations during evolution.

Table	Geometry	Max Gens	#In	#Out	Description
1-Bit Adder	1 x 30	20K	3	2	1-Bit plus 1-Bit Binary Adder with Carry
2-Bit Multiplier	1 x 40	20K	4	4	2-Bit by 2-Bit Binary Multiplier
2-Bit Adder	1 x 40	20K	5	3	2-Bit plus 2-Bit Binary Adder with Carry
2.5-Bit Multiplier	1 x 50	200K	5	5	3-Bit by 2-Bit Binary Multiplier
Even 6-Bit Parity	1 x 30	30K	6	1	Even 6-Bit Parity

N.B. Most studies on the even n-parity problem explicitly disallow the use of XOR or XNOR primitive gates to force evolution to build these gates out of other gates. We were interested in the ability of evolution to exploit useful existing genetic primitives, so in the experiments reported here evolution was free to use any 2-input gate (including XOR and XNOR) in evolving even 6-parity.

4. Gate Selection in Binary Logic Circuits

4.1. *Post Hoc* Selection

We began by trying to automate the *post hoc* discovery of useful primitives following Miller et al. [8] as described in section 2.3. It turned out that simply counting gate frequencies in evolved solutions was not the best guide to choosing a primitive set. For instance, consider the evolved solutions for the 1-bit adder (table II).

Table II. Proportion of total gates accounted for by each primitive gate, found by amalgamating the gates in 1000 functionally correct 1-bit adder circuits evolved using uniform gate mutation.

Gate	Function	Proportion	Gate	Function	Proportion
0	False	1.60%	8	A AND (NOT B)	6.03%
1	True	1.64%	9	A NOR B	6.69%
2	A	4.63%	10	A XOR B	15.11%
3	B	4.64%	11	A XNOR B	16.11%
4	NOT A	5.79%	12	A OR B	7.23%
5	NOT B	4.81%	13	A OR (NOT B)	5.83%
6	A AND B	7.13%	14	(NOT A) OR B	5.88%
7	(NOT A) AND B	5.79%	15	A NAND B	7.01%

The four most frequently-used gates in functionally correct solutions, from most frequent to least frequent, are 11 (XNOR), 10 (XOR), 6 (AND), and 15 (NAND). Evolving the one-bit adder using only these four gates as primitives was definitely faster (taking only 43% of the time on average) than using all 16 possible gates. However, cutting the primitive set down to only XNOR and XOR is disastrous, since a functional adder cannot be constructed from only XNOR and XOR gates! In fact, experiments indicated that the best post hoc selection of primitives came from looking at the functionally correct circuits which used the *least diversity* of primitives.

4.2. Plagiarism

Post hoc discovery of useful primitives is often of limited use. The experimenter wants to find a useful set of primitives *during* evolution if possible. One way to do this is to change the mutation operator so that a point mutation is more likely to result in a primitive which is already used by the better genomes in the population. A very simple method for achieving such a result in 1+1 evolution is as follows: -

1. Randomly choose a genetic locus to mutate.

2. Randomly choose a second locus on the same genome. *Copy* the contents of this locus to the first locus (see figure 2).

We call this operator, which functions by copying material from one genetic locus to another, the *plagiarism* genetic operator. It is similar in some respects to the *non-homologous crossover* used in standard GP. However, non-homologous crossover in GP is thought to be detrimental to evolution [4] due to its propensity to cause bloat, whereas we will show results indicating that plagiarism is beneficial for evolution in the digital logic circuit domain.

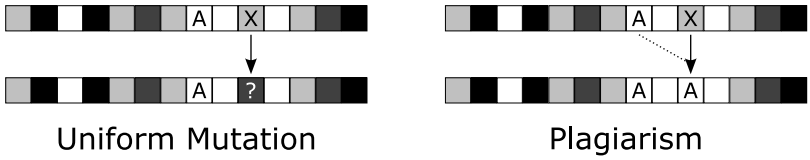

Figure 2. The plagiarism operator. Left: conventional uniform mutation changes the value of the locus marked X to an arbitrary value. Right: plagiarism changes the value of the locus marked X to the value of an arbitrary other locus.

Since we were primarily interested in identifying useful gates, we restricted the action of this operator to the parts of the genome which encoded *gate function*. The genetic *input indices* and *output connection indices* loci were mutated as before. To preserve diversity in gate function, only 50% of *gate function* point mutations were plagiarism events; the remaining 50% were ordinary uniform mutations. The reason for this should become clear later in the document. Obviously, when choosing genetic material to copy for gate function plagiarism, only *gate function* loci were considered. Furthermore, to take maximal advantage of available information, only those loci which were actually expressed in the elite phenotype were potential targets for plagiarism.

So the actual genetic modification procedure for *gate function* loci was as follows: - 50% of the time, perform a uniform mutation to any one of 16 possible primitives. Otherwise, choose a genetically active gate at random from the elite genotype and copy its gate function primitive into the locus being modified.

This operator was very successful. See table III for a comparison of performance on various different binary arithmetic problems. The plagiarism operator reduced solution time for all of the problems[1]; except

[1] Experiments with an adaptive mutation rate similarly reduced solution time, with an (as yet) unexplained anomaly in the 2-bit adder.

for the 6-bit parity problem, the difference was statistically significant at the $p < 0.05$ level or better. When the 6-bit parity problem was re-run with a sample size of 200, the difference was statistically significant ($p < 10^{-6}$).

Table III. Evolutionary performance using 50% plagiarism and 50% uniform mutation vs. 100% uniform mutation for 5 different problems. Figures given are median number of fitness evaluations ($N = 100$) to functionally correct solutions or run termination. Wilcoxon rank sum test p values are shown.

Table	Plagiarism	Uniform	p
1-Bit Adder	2,154	2,968	2.05%
2-Bit Adder	44,060	55,379	2.40%
2-Bit Multiplier	11,271	14,683	2.25%
2.5-Bit Multiplier	151,842	200,001	0.92%
Even 6-Bit Parity	2,651	3,242	18.18%

5. Analysis: Difference Between Operators

The plagiarism operator has an interesting statistical property. Consider the *distribution of primitives* in a genome. Suppose a genome G of length n primitives is composed of primitives $a_1, a_2, a_3, \ldots$ in an alphabet A. Then we count the number of occurrences of each primitive a_r in G giving us an absolute count f_r and a relative frequency $q_r = \frac{f_r}{n}$ for each primitive a_r. Clearly

$$\sum_r f_r = n \sum_r q_r = 1$$

Now, perform a plagiarism operation on G to produce a new genome G' (still of length n). What is the expectation of f'_r for each r? The plagiarism operator chooses a locus on G with uniform probability and replaces it with the primitive at another uniformly chosen locus. So, with probability

$$\alpha = q_r(1 - q_r)$$

a primitive a_r will be chosen and replaced with a different primitive, resulting in $f'_r = f_r - 1$. Similarly, with probability

$$q_r(1 - q_r) = \alpha$$

a primitive which is not a_r will be chosen and replaced with a copy of a_r. The remainder of the time, the number of a_r in G will not be changed. So the expected value of f'_r is

$$E(f'_r) = \alpha(f_r - 1) + \alpha(f_r + 1) + (1 - 2\alpha)(f_r) = f_r$$

In other words, the effect of the plagiarism operator on a genome is neutral with respect to that genome's distribution of primitives. On binary genomes, plagiarism performs a random walk (sticking at boundaries) in distribution-of-primitives space. Applying the plagiarism operator repeatedly in the absence of selection, eventually the genome will consist of n copies of the same primitive. If the operator is "tweaked" to allow it to leave the boundaries in its random walk (e.g. by maintaining an explicit count of primitive frequency and making the boundaries special cases), it will drift uniformly through the entire distribution space.

This is not true of conventional uniform mutation. Applying uniform mutation repeatedly in the absence of selection, the distribution of primitives rapidly converges to a uniform distribution, where there is an equal number of each type of primitive, and stays there. The expected effect of applying it just once is to push the distribution in the direction of a uniform distribution.

Seen from this point of view, uniform mutation is a biased operator, which has an opinion about the correct distribution of primitives in an optimal genome. How can that be the case? The reason is that genomes with a near-uniform distribution of primitives vastly outnumber genomes with a non-uniform distribution of primitives. Uniform mutation performs an unbiased random walk in genome space, with the consequence that its walk in distribution-of-primitives space is highly biased.

When a genome has more than two possible primitives, plagiarism does not perform a true random walk in distribution-of-primitives space. However, it still explores that space more evenly than uniform mutation. Figure 3 clearly illustrates this. Simulated plagiarism and mutation operators are applied to a random genome of 1000 loci, each of which can be one of 4 possible genetic primitives.

5.1. Plagiarising Too Enthusiastically

We tested evolutionary performance under *softmax* plagiarism. This operator counts the frequency of each genetic primitive, and then chooses a primitive with probability based on the *softmax* (*multiple logistic*) function

$$p(a_n) = \frac{e^{q_n}}{\sum_{a_r \in A} e^{q_r}}$$

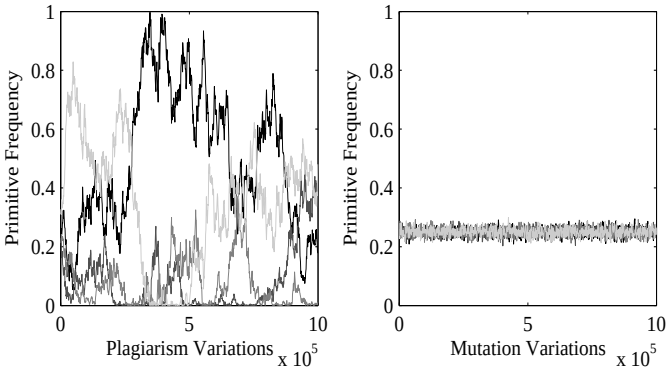

Figure 3. The frequency of each of 4 possible genetic primitives in a 1000-locus genome over time under plagiarism or mutation in the absence of selection. The x-axis counts variations, i.e. operations which actually alter the genome.

where $p(a_n)$ is the probability of picking primitive a_n from the primitive alphabet A and q_n is the frequency of the primitive a_n in the genome.

In contrast to original plagiarism, the softmax function causes the most common primitives to be selected disproportionately, resulting in a tendency to rapidly fill the genome up quickly with copies of the same primitive.

Using 50% softmax plagiarism instead of 50% plagiarism resulted in very slow evolution (mean evaluations to solution or termination increased 186%, Wilcoxon rank sum $p < 10^{-10}$). This suggests that the finely balanced nature of the plagiarism operator is an advantage in exploring genetic space.

Interestingly, evolution with softmax plagiarism sometimes produced solutions consisting of only one type of gate (e.g. all NAND gates, or all NOR gates), which balanced plagiarism or uniform mutation never did.

6. Analysis: Toy Problems

6.1. Counting-Ones

The simplest of toy problems in the artificial evolution world is the *counting-ones* problem. In this problem, the genotype is represented as a binary string and its fitness is directly proportional to the number of ones in it. Intuitively, this is a problem on which the plagiarism operator should be provably superior to uniform mutation, since the value of a genetic primitive (a one or zero) is completely independent of its position in the genome.

In fact, the plagiarism operator cannot be used alone because of its inability to introduce primitives which are not already present in the genome. In other words, it cannot escape from the genome consisting of all zeroes. Therefore, a certain proportion of the time it is necessary to use conventional mutation. We will analyse the situation in which the evolutionary algorithm uses a random mixture of plagiarism and conventional mutation in proportions λ and $1-\lambda$ respectively.

For this toy case, consider a 1+1 hill-climber which performs exactly one genetic operation at each generation. A mutation which turns a 0 into a 1 will be preserved, and any other mutation will be discarded. The plagiarism operator picks a single locus at random and replaces its contents with the contents of a second randomly chosen locus, while the uniform mutation operator picks a single locus at random and replaces its contents with 1 or 0 with uniform probability. So both operators have a chance of leaving the genome unchanged. It is assumed that the genome is initialised by setting each bit to 1 or 0 with uniform probability.

Suppose the genome is of length n, and f_1 bits in the genome are already 1s. Then the probability of turning a 0 into a 1 is: -

$$p(f_1 \to f_1+1) = \frac{n-f_1}{n}\Big(\lambda\frac{f_1}{n} + (1-\lambda)\frac{1}{2}\Big)$$

and the expected number of generations $g(f_1, f_1+1)$ to go from f_1 ones to f_1+1 (assuming $f_1 < n$) is the reciprocal of $p(f_1 \to f_1+1)$

$$g(f_1, f_1+1) = \frac{2n^2}{(n-f_1)(2\lambda f_1 + n - \lambda n)}$$

so the expected number of generations $t(f_1)$ to go from f_1 ones to the solution containing n ones is

$$g(f_1, n) = \sum_{r=f_1}^{n-1} g(r, r+1) = 2n^2 \sum_{r=f_1}^{n-1} \frac{1}{(n-r)(2r\lambda + n - \lambda n)}$$

and, factoring in the relative likelihood of the initial genome having f_1 ones in it, the overall expected number of generations to solution $G(n,\lambda)$ is

$$G(n,\lambda) = 2^{n-1}n^2 \sum_{f_1=0}^{f_1-1} \binom{n}{r} \sum_{r=f_1}^{n-1} \frac{1}{(n-r)(2r\lambda + n - \lambda n)}$$

where $\binom{n}{r}$ is the binomial coefficient, i.e. the number of ways of choosing r unordered objects from n. In the range $0 \le \lambda < 1$ and $n > 2$, this

is a curve which rises without bound as $\lambda \to 1$ (because of the risk of starting without any 1s in the genome) and has a single minimum value between 0 and 1. For instance, in the simple case where $n = 3$, G turns out to be

$$G(3, \lambda) = \frac{63 - 72\lambda + 11\lambda^2}{(\lambda - 1)(3 - \lambda)(3 + \lambda)}$$

Using the computer algebra package Maple, an exact optimum value for λ can be calculated by setting the derivative of $G(n, \lambda)$ to zero and solving numerically. Some example values are given in table IV.

Table IV. Optimal proportion of plagiarism vs. uniform mutation in counting-ones problems of different lengths.

Genome length	3	4	5	9	15
Optimum λ	0.4006	0.6714	0.8084	0.9743	0.9980

It can be seen that the optimum value for λ rapidly goes to 1 as the genome length increases; in other words, the uniform mutation operator contributes very little when the genome is large enough to guarantee a reasonable initial mixture of 0s and 1s.

6.2. Counting Ones, General Case

What if we consider a more general version of the counting-ones problem in which there is an optimal proportion μ of ones? Define the fitness of an individual to be

$$1 - \left(\frac{f_1}{n} - \mu\right)^2$$

where, as before, f_1 is the number of ones in the individual's genome and n is the length of the genome. The standard counting-ones problem is then the special case where $\mu = 1$.

Then analysis and numerical solution similar to that in the previous section provides performance comparisons for 99% plagiarism and 100% uniform mutation across a range of target proportions.

We can see from figure 4 that plagiarism outperforms uniform mutation when the optimal distribution of primitives is far from uniform. Close to the uniform distribution (0.5), uniform mutation outperforms plagiarism. The window in which uniform mutation outperforms plagiarism narrows as the genome length increases. For a genome of length

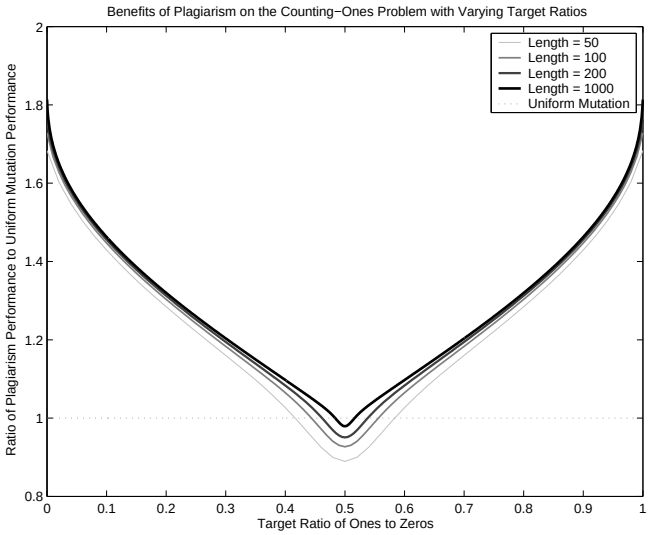

Figure 4. Analytically derived expected plagiarism speedup compared with uniform mutation (ratio of expected evaluations until solution) for different target ratios and genome lengths in the generalised counting-ones problem.

1000 bits, uniform mutation outperforms plagiarism only when the target ratio is within 1.6 percentage points of 50%.

7. Random Fitness Landscapes

It has been known for some time [22] that no optimisation or search algorithm can outperform any other algorithm, including random search, on average over all possible target functions. Consequently, when comparing the merits of any two genetic algorithms, one must characterise the particular types of fitness landscape they perform well on. We decided to perform the relevant analysis comparing the plagiarism operator to conventional uniform mutation. In order to better determine the effects of the two operators, we considered both hill-climbing search and simple random walk search. Our hypothesis was that the relative performance of plagiarism and random mutation would be independent of whether very simple evolution (i.e. hill-climbing) was used.

The formal definition [22] of a search algorithm for the No Free Lunch theorems is quite strict and in some ways does not correspond to evolutionary algorithms. In particular, unlike standard evolutionary search (or for that matter naive random search), a theoretical search algorithm is guaranteed to never revisit a point which has already been

searched. The search algorithms we investigated are described in figures 5 and 6.

1. Begin by testing a particular point, and set the "best-so-far" point to this point.
2. Conduct a random walk using the chosen genetic operator, *beginning at the best-so-far point*, until a point is reached which has not previously been tested.
3. Test the new point. If it is better than the best-so-far, it becomes the new best-so-far. If it is the global optimum, terminate. Otherwise, go to **2.**

Figure 5. Hill-climbing search algorithm.

1. Begin by testing a particular point.
2. Conduct a random walk using the chosen genetic operator, *beginning at the most recently tested point*, until a point is reached which has not previously been tested.
3. Test the new point. If it is the global optimum, terminate. Otherwise, go to **2.**

Figure 6. Random walk search algorithm.

The performance of both operators under hill-climbing and random walk search was tested on random permutation binary fitness landscapes of various sizes. The random fitness landscapes had the following properties: -

1. Every point in the landscape was given a value between 0 and $2^n - 1$, where n is the number of bits in a genome. No two points were assigned the same value.

2. The values were randomly permuted, i.e. shuffled using a fair shuffling algorithm.

Each landscape was tested with all 4 search algorithms. A single test of an algorithm consisted of running the algorithm 2^n times on a given fitness landscape, once for each possible starting position, and averaging the number of fitness evaluations needed to find the optimum. Each algorithm was tested this way 500 times on every landscape. In all, 5000 random landscapes were generated and tested for $n \in \{3, 4, 5\}$.

Bootstrap resampling tests[3] (resampling size 1000) were used to estimate 95% confidence intervals for the various means. Within landscapes of a given genome length, the mean number of evaluations to

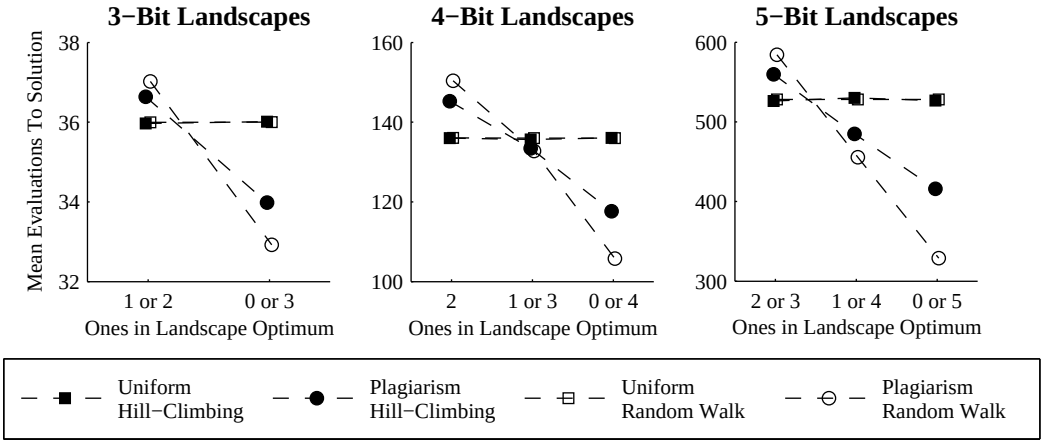

Figure 7. Mean evaluations to solution on 5000 random permutation binary fitness landscapes by mutation operator, search algorithm type and number of ones in the optimal solution. 95% confidence intervals, not shown, are comparable in size to data markers. X-axis values are staggered slightly to reveal overlapping markers.

solution over all the sampled landscapes was, as predicted by NFL, statistically indistinguishable for each of the 4 search algorithms. When landscapes were classified according to the proportion of ones in the optimal genome, the plagiarism searches performed for optimal genomes containing a non-uniform mixture of ones and zeros (see figure 7). This effect was somewhat greater in the random walk search as compared to the hill-climbing search condition. By contrast, the mean performance of searches using uniform mutation was unaffected by the number of ones in the landscape optimum.

8. Future Work

It is unclear to what extent the plagiarism operator is applicable outside of the Boolean circuit domain. Experiments on widely differing problems are clearly called for, although the genetic primitives must be similar in meaning across different loci. The basic plagiarism operator suggests a variety of extensions, such as an operator which copies several neighbouring loci at once.

We believe the idea of evenly exploring distribution-of-primitives space, and other relevant spaces of *a priori* interest, is worth pursuing. This line of research could be further developed by constructing mutation operators which perform an *explicit* random walk in a given space.

9. Conclusion

This paper describes a new genetic operator for evolutionary search and optimisation, the *plagiarism* operator, which involves the copying of genetic primitives between different loci. We have presented experimental results in which this operator facilitates digital logic circuit evolution. Analytically, it outperforms uniform mutation on far-from-uniform counting-ones tasks. On random fitness landscapes with far-from-uniform optima, it outperforms uniform mutation regardless of whether hill-climbing or random search is used.

This is yet another example of how prior knowledge is invaluable in constructing evolutionary operators. Our analysis provides theoretical validation for the practice of using copy operators in domains where fitter individuals should be expected to have a non-uniform distribution of genetic primitives. The plagiarism operator may well be valuable in practical optimisation or design problems, and is rather easy to implement. The analysis in terms of plagiarism's effectiveness in exploring the distribution-of-primitives space can, we hope, inspire the design of further ad-hoc operators based on prior domain knowledge.

References

1. Angeline, P. J. and J. B. Pollack: 1993, 'Evolutionary Module Acquisition'. In: D. Fogel and W. Atmar (eds.): *Proceedings of the Second Annual Conference on Evolutionary Programming*. La Jolla, CA, USA, pp. 154–163.
2. Bailey, J. A., G. Liu, and E. E. Eichler: 2003, 'An Alu Transposition Model for the Origin and Expansion of Human Segmental Duplications'. *American Journal of Human Genetics* **73**, 823–34.
3. Efron, B. and G. Gong: 1983, 'A leisurely look at the bootstrap, the jackknife, and cross-validation'. *The American Statistician* **37**, 36–48.
4. Hansen, J. V.: 2003, 'Genetic Programming Experiments with Standard and Homologous Crossover Methods'. *Genetic Programming and Evolvable Machines* **4**(1), 53–66.
5. Koza, J. R.: 1994, *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.
6. Miller, J.: 2001, 'What bloat? Cartesian Genetic Programming on Boolean problems'. In: *Late Breaking Papers, Proceedings of the 3rd Genetic and Evolutionary Computation Conference (GECCO'01)*. pp. 295 –302.
7. Miller, J. F., D. Job, and V. K. Vassilev: 2000, 'Principles in the Evolutionary Design of Digital Circuits – Part I'. *Journal of Genetic Programming and Evolvable Machines* **1**(1), 8–35.
8. Miller, J. F., T. Kalganova, N. Lipnitskaya, and D. Job: 1999, 'The Genetic Algorithm as a Discovery Engine: Strange Circuits and New Principles'. In: *Proceedings of the AISB Symposium on Creative Evolutionary Systems (CES'99)*. pp. 65–74.

9. Naemura, T., T. Hashiyama, and S. Okuma: 1998, 'Module Generation for Genetic Programming and Its Incremental Evolution'. In: C. Newton (ed.): *Second Asia-Pacific Conference on Simulated Evolution and Learning*. Australian Defence Force Academy, Canberra, Australia.
10. Ohno, S.: 1970, *Evolution by Gene Duplication*. Springer Verlag.
11. Rosca, J. P.: 1997, 'Hierarchical Learning with Procedural Abstraction Mechanisms'. Ph.D. thesis, Department of Computer Science, The College of Arts and Sciences, University of Rochester, Rochester, NY 14627, USA.
12. Simoes, A. and E. Costa: 1999a, 'Enhancing Transposition Performance'. In: *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 99)*. Washington, USA, pp. 1434–1441.
13. Simoes, A. and E. Costa: 2000, 'Using Genetic Algorithms with Asexual Transposition'. In: D. Whitley, D. Goldberg, L. S. E. Cant-Paz, I. Parmee, and H. Beyer (eds.): *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO' 2000)*. Las Vegas, USA, pp. 323–330.
14. Simoes, A. B. and E. Costa: 1999b, 'Transposition versus Crossover: An Empirical Study'. In: W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (eds.): *Proceedings of the Genetic and Evolutionary Computation Conference*, Vol. 1. Orlando, Florida, USA, pp. 612–619.
15. Spector, L.: 1995, 'Evolving Control Structures with Automatically Defined Macros'. In: E. V. Siegel and J. R. Koza (eds.): *Working Notes for the AAAI Symposium on Genetic Programming*. MIT, Cambridge, MA, USA, pp. 99–105.
16. Thompson, A.: 1997, 'An evolved circuit, intrinsic in silicon, entwined with physics'. In: T. Higuchi, M. Iwata, and L. Weixin (eds.): *Proc. 1st Int. Conf. on Evolvable Systems (ICES'96)*, Vol. 1259 of *LNCS*. pp. 390–405.
17. Thompson, A., P. Layzell, and R. S. Zebulum: 1999, 'Explorations in Design Space: Unconventional electronics design through artificial evolution'. *IEEE Trans. Evol. Comp.* **3**(3), 167–196.
18. Urbanek, G.: 2003, 'Factors having an effect on quality and cost of solution in identification of inverse models with the application of genetic algorithms'. In: *Presented at AI-METH 2003: Methods of Artificial Intelligence*.
19. Vassilev, V. K., T. C. Fogarty, and J. F. Miller: 2000, 'Smoothness, Ruggedness and Neutrality of Fitness Landscapes: from Theory to Application'. In: A. Ghosh and S. Tsutsui (eds.): *Theory and Application of Evolutionary Computation: Recent Trends*. Springer-Verlag.
20. Voss, M. S. and C. M. Foley: 1999, 'Evolutionary Algorithm For Structural Optimization'. In: W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (eds.): *Proceedings of the Genetic and Evolutionary Computation Conference*, Vol. 1. Orlando, Florida, USA, pp. 678–685.
21. Walker, J. A. and J. F. Miller: 2004, 'Evolution and Acquisition of Modules in Cartesian Genetic Programming'. In: M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule (eds.): *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, Vol. 3003 of *LNCS*. Coimbra, Portugal, pp. 187–197.
22. Wolpert, D. H. and W. G. Macready: 1995, 'No Free Lunch Theorems for Search'. Technical Report SFI-TR-95-02-010, Santa Fe, NM.