Brock University

Department of Computer Science

A Novel Variation Operator for More Rapid Evolution of DNA Error Correcting Codes

D. Ashlock and S. Houghten
Technical Report # CS-05-09
August 2005

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

A Novel Variation Operator for More Rapid Evolution of DNA Error Correcting Codes.

Daniel Ashlock
Mathematics and Statistics
University of Guelph
Guelph, Ontario
Canada N1G 2W1
dashlock@uoguelph.ca

Sheridan K. Houghten
Computer Science
Brock University
St. Catharines, Ontario
Canada L2S 3A1
houghten@brocku.ca

***Abstract*— Error correcting codes over the edit metric have been used as embedded DNA markers in at least one sequencing project. The algorithm used to construct those codes was an evolutionary algorithm with a fitness function with exponential time complexity. Presented here is an substantially faster evolutionary algorithm for locating error correcting codes over the edit metric that exhibits either equivalent or only slightly inferior performance on test cases. The new algorithm can produce codes for parameters where the run-time of the earlier algorithm was prohibitive. The new algorithm is a novel type of evolutionary algorithm using a greedy algorithm to implement a variation operator. This variation operator is the sole variation operator used and has unary, binary, and k-ary forms. The unary and binary forms are compared, with the binary form being found superior. Population size and the rate of introduction of random material by the variation operator are also studied. A high rate of introduction of random material and a small population size are found to be the best.**

I. Introduction

An *error correcting code* is a collection of strings over a given alphabet that are well-separated from one another. The separation property means that small numbers of errors in transmission of a code word can be detected (because the received word is not a code word) and corrected (by calculating which code word is closest to the received word). Typically the type of errors being corrected are single character substitutions which are counted by the *Hamming distance*. This study instead explores creation of codes relative to the *edit metric* or *Levenshtein distance*. The edit distance between two words is the smallest number of single character substitutions, insertions, or deletions that can transform one word into the other. This is, in fact, a metric in the mathematical sense [5]. Edit metric codes are startlingly unlike codes over the Hamming metric [4]. Very little of the beautiful algebraic theory of Hamming metric codes applies to edit codes. While the edit metric permits code words with variable length we will, for reasons of simplicity in both coding and biological applications, work with code words of a fixed length. An (n, d)*-code* is a code whose words are of length n with pairwise minimum distance between code words of d.

Edit metric error correcting codes, over the DNA alphabet, are used to track expressed sequence tags (expressed genes) in sequencing projects. An example of their use in corn genomics appears in [10]. Error correction over the edit metric is required because DNA sequencers make edit, rather than Hamming, style errors. In the cited research, the use of an error correcting code as embedded genetic tracking tags permitted recovery of 50% of the data that would otherwise have been lost. In this study *distance* means edit distance; other notions of distance will be named explicitly, e.g. "Hamming distance".

Thus far edit metric error correcting codes have been located with an evolutionary algorithm that uses a fitness function with exponential time complexity[1]. This high time complexity was acceptable only because a relatively short code (of length 6 and able to correct a single error) was required. The algorithm used to find these codes operates by evolving small collections of code words called *code seeds* that are completed to a full code with a greedy algorithm. If we view the greedy algorithm as a type of optimization then this algorithm is a Baldwinian one. The technical details of this initial algorithm for finding edit metric codes appears in [1]. The fitness of a code seed is the size of the code located by the greedy algorithm starting with the seed and this definition of fitness is retained in the current study. In general, once the length of the code words and the number of errors that must be corrected are fixed, codes are better if they are bigger (possess more code words). To understand why bigger codes are better, consider the application of tagging biological constructs. A larger code provides more available labels at a given level of error correction and (word length based) difficulty of synthesis.

Conway's lexicode algorithm, Algorithm 1, is the key to both the old and new evolutionary algorithms. The form given here is modified from the original to operate on general input sets.

Algorithm 1: **Conway's Lexicode Algorithm**
Input: An alphabet $\mathcal{A}$, a minimum distance d and and ordered subset of $S \subset \mathcal{A}^n$.

Output: $CLA(S)$, a subset of S that has pairwise minimum distance d.

Details:
Initialize an empty set R of words.

Traverse the members $s \in S$ in order

If s is at least distance d from every member of R,
add s to R

Return R as $CLA(S)$.

The original lexicode algorithm, defined by Conway for theoretical reasons[3], ran with $S = \mathcal{A}^n$ in lexicographical order. We call the code thus obtained the (n, d)-*Conway code*.

In the remainder of the paper, section II will review some elementary theory of error correction and discuss the problem of representing error correcting codes for evolutionary search. Section III will give the design of the evolutionary algorithm used in this study, including the novel variation operator. Section IV gives the main results of the paper, the parameter setting studies that begin to explain how the new variation operator differs from standard variation operators like crossover and mutation. This section also presents results on the degree to which the parameter setting study generalizes. Section V discusses these results. Section VI gives potential variations of the algorithm design not tested in the current study. Tables of best-known code sizes are given in an appendix.

II. Error Correcting Codes and Their Representation

We begin by reviewing some elementary properties of error correcting codes and the Hamming and edit metrics. First notice that the edit distance between two words can be no more than the Hamming distance between those words. This follows from the fact that substitutions, which generate Hamming distance, also generate edit distance. The opposite does not hold: consider two words of length n constructed by alternating the same two characters. If their initial characters differ then they are at Hamming distance n. Deleting the terminal character of one and inserting a copy of that character at the beginning of the same word shows the words are at edit distance two. Hamming distance can be computed in time $O(L)$ for length L words where edit distance, computed by dynamic programming [9], requires time $O(L^2)$. The Hamming distance between words can thus be used to speed up computations that check for a minimum edit distance: if the cheaper-to-compute Hamming distance is already too small then the more expensive edit distance need not be computed. In practice first checking the Hamming distance and then checking the edit distance only when the Hamming distance is acceptable yields a substantial reduction in runtime for both the seed based algorithm and the algorithm described in this study.

Error correcting codes are characterized by their word length n and their minimum distance d. The number of errors that they can correct can be computed from d. A code word is processed in some manner that created errors. In standard error correcting codes this is a noisy channel such as an acoustic modem or radio transmitter and receiver. For edit metric codes the typical source of error is a DNA sequencer. If the number of errors is less than half of d the processed word is closer to the code word that spawned it than to any other code word. The errors are corrected by polling the words of the code to see which is closest. Thus if $d = 2r + 1$ an error correcting code can correct r errors.

An error correcting code, viewed as a data structure, is simply a subset of $\mathcal{A}^n$ for some alphabet $\mathcal{A}$. The application of edit metric codes thus far has been to DNA meaning that $\mathcal{A} = \{C, G, A, T\}$, an alphabet of size 4. Table IV, which gives the sizes of the (n, d)-Conway codes, serves as a lower bound on the size of the sets of words we will need to specify. A length 7, distance 3 code would thus be a set of over 300 seven character strings. Stored as a bit string this would require over 42000 binary variables. In addition such a representation would be hard put to avoid epistasis. No code word is good in isolation: its quality depends on the presence of absence of many other words in the code. All of this strongly suggests that a direct representation is not a good idea.

In the initial study partial codes with three code words, the previously mentioned *code seeds*, were used as the representation. Fitness was computed by initializing Conway's lexicode algorithm with the code seed instead of the empty set and then applying the algorithm to the set of all words of length n. The exponential size of the set of all words is thus the source of the exponential time complexity of the original algorithm. In spite of its high time complexity, this algorithm achieved a compact representation for error correcting codes. This representation had variation operators that modified the code seed, discarding it if the three words forming the seed violated the minimum distance condition for the code. The representation may also be *incomplete* in the sense that there is no reason to think that all codes can be reconstructed from three of their words by Conway's algorithm. It may be the case that codes of optimal size sometimes cannot be located by this algorithm.

Another possible approach is to store the code directly, as a list of words that are a code with the minimum specified distance. The normal kinds of variation operators cannot be applied in any natural way to this representation. If one has a good code then almost all of the space $\mathcal{A}^n$ is made of strings either in the code or within distance d of a member of the code. This means that "mutating" a code word is very likely to violate the minimum distance condition. Likewise, any "crossover" that mixes and matches words from two codes is very likely to bring incompatible words (at distance less than d) together. The direct representation of codes is also less compact than the code seeds. To be useful it must be combined with a variation operator that solves the problem of generating new incompatible words or of uniting incompatible words when combining two or more codes. This new variation operator must also avoid examining all of $\mathcal{A}^n$. Such a variation operator is described in the next section.

III. Evolutionary Algorithm Design

In the evolutionary algorithm defined in this study, codes are represented as ordered lists of words of length n that satisfy the minimum distance condition, that all pairs of words are at edit distance d from one another. The algorithm is, in most respects, a standard evolutionary algorithm *except* that it uses only one variation operator, described as Algorithm 2.

Algorithm 2: **Conway Variation Operator**
Input: One or more (n, d)-codes $C_1, \ldots C_k$, a random material number $R \geq 0$ (an integer), and a minimum distance d.

Output: An $(n, d) - code$.

Details:
Let Q be the union of $C_1, \ldots, C_k$.
Generate R random words and add them to Q.
Shuffle the set Q into a random order.
Apply Algorithm 1 to Q, returning $CLA(Q)$.

With the Conway variation operator defined the remainder of the evolutionary algorithm may be specified. The random material number plays a role similar to mutation, in that new words can only enter the population via the addition of the random codewords associated with R. The algorithm is steady-state[12], proceeding by mating events. In a mating event a tournament of three codes are chosen at random from the population. The one or two best codes are subjected to the Conway variation operator and the resulting code replaces the worst code in the tournament. The population size and random variation number are the principal objects of the parameter setting study and are specified on an experiment-by-experiment basis. For each set of parameters 100 evolutionary runs are performed, lasting for 100,000 or 400,000 mating events.

The fitness of a code is simply its size. Initial populations are created by generating random lists of W words, one for each member of the population, and applying Algorithm 1 to obtain initial codes, $CLA(W)$. The size of W is usually 80 but varies and is specified in each experiment. We conclude this section by demonstrating that the algorithm specified is complete, in the sense that it can find any optimal (maximum size) error correcting code.

Theorem 1: If $R \geq 1$ and $C = \{w_1, w_2, \ldots, w_m\}$ is an optimal code then the evolutionary algorithm described in this section can locate C.
Proof:
The Conway variation operator blends one or more codes with random material in a random order. If, in each application of the variation operator to a particular lineage of codes, R words of C are introduced in the correct order and position, prefixes of C of size $R \cdot k$ will result after the kth application of the variation operator. The algorithm thus has a positive probability of reproducing code C. □.

While the positive probability of producing a given optimal code must be very small, this is a stronger result than any for the evolutionary algorithm described in [1].

IV. Setting Algorithm Parameters

For the initial study of parameter sizes $(7, 5)$-codes were used with the size of the random set used to initialize members of the population set at $W = 80$. The best known $(7, 5)$-code has 18 members. For this code we will study the best random material number R, survey several population sizes, and test for one case if unary or binary variation operators

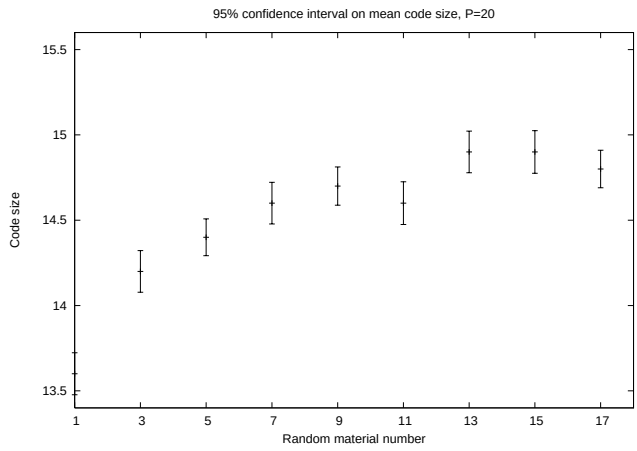

Fig. 1. For populations of size 20 this plot shows a 95% confi dence interval on the mean size of best codes located in 100 simulations as a function of R, the random material number. All runs shown here used 100,000 mating events and $W = 80$.

yield better performance. The effect of population seeding by placing the $(7, 5)$-Conway code into the initial population and of using a form of Lamarkian mutation at a low rate (0.5%) are reported. The utility of running the algorithm for a larger number of mating events is checked. Finally, experiments on the problem of locating a $(6, 3)$-code are reported. All results other than the binary-unary comparison use the binary form of the variation operator.

A. Random Material Inclusion Rate

Figure 1 shows the impact on code size of different random material numbers when the population size is 20. Random material rates of 13 and 15 are significantly better than all those below 9, as well as 11. This shows that the number of random words added during variation is more than half the size of an optimal structure. The upper half of Figure 2 also shows that large values of R are good when the population size is 5, though 11 is the winner.

The high values for R that yield good performance suggest that while R plays the role in the evolutionary algorithm as a mutation rate its "units" are very different.

B. Population Size

Comparing the upper half of Figure 2 with Figure 1 we see that populations of size 5 substantially outperform populations of size 20. A lateral comparison for $R = 9$ of population sizes 5, 20, 60, and 100 appears in the lower half of Figure 2. These results show substantially superior performance for the population of size 5.

C. Use of the Unary Variation Operator

Table I summarizes the outcome of a comparison of the unary and binary versions of the variation operator. The experiments were run with populations of five codes for a variety of random material rates, R. The runs proceeded for 100,000 mating events each. Other than using the unary or binary form of the variation operator and varying R the experiments were

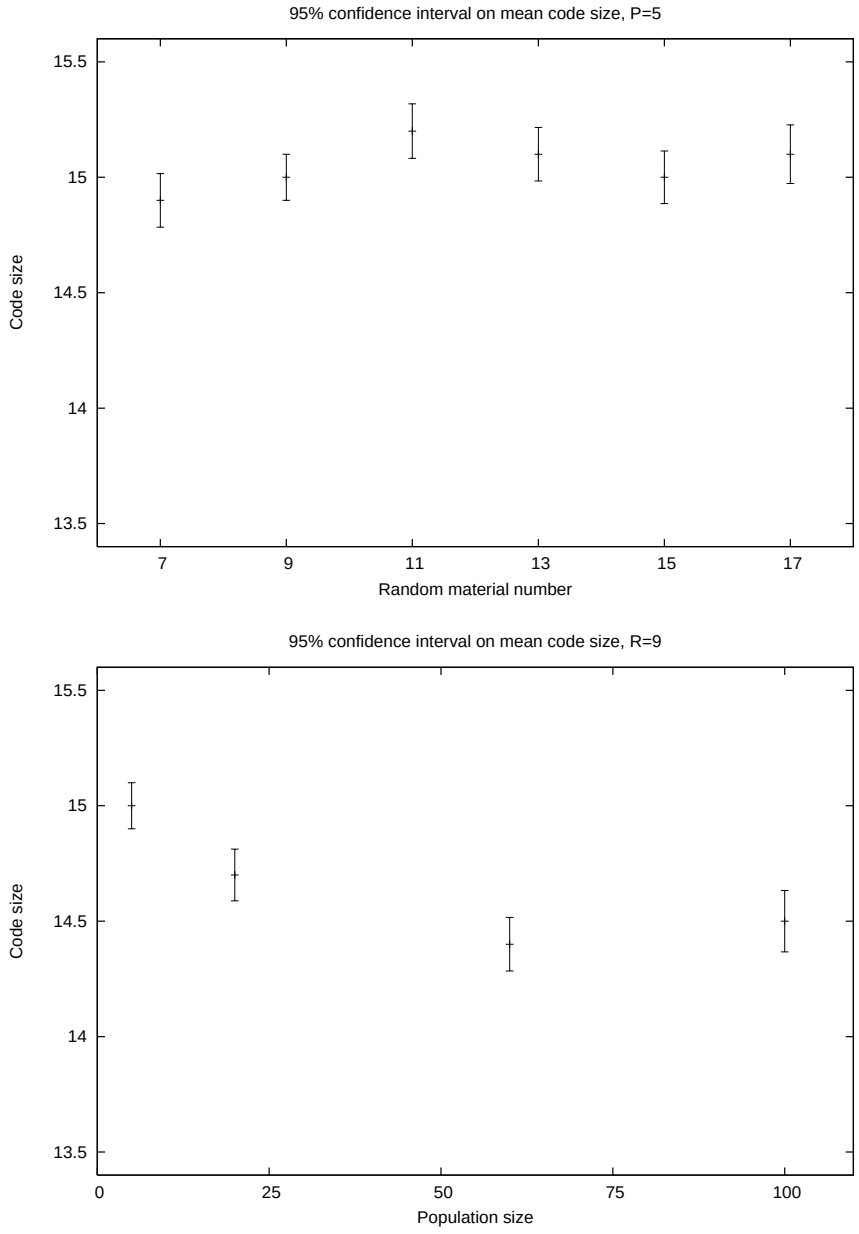

Fig. 2. For populations of size 5 the upper plot shows a 95% confi dence interval on the mean size of best codes located in 100 simulations as a function of R, the random material number. The lower plot shows the same statistic as a function of population size with $R = 9$. All runs shown here used 100,000 mating events.

run with identical parameters. All four pairs of experiments show a significant advantage to using the binary form of the variation operator.

The results in Table I represent an incomplete test of the utility of the unary version of the variation operator. It may be that very large values of R might enhance its performance and this is a matter for subsequent explorations.

D. Population Seeding: Initialization with the Conway Code

Population seeding is the placement of high-quality individuals into a random population in hopes of jump-starting evolution and obtaining better results. The Conway codes are substantially better than those in a typical initial random population and so are a natural choice for use in population seeding. A single copy of the $(7, 5)$-Conway code was placed into the initial population. The algorithm parameters were set to: population size 5, $R = 13$, and $W = 80$. A set of 100 runs was performed with lackluster results.

The size of the best code located dropped from 17 to 16 while the 95% confidence interval on the size of the best code located rose from (15.0,15.2) to (15.1,15.3). Variance in the size of the best code was 0.59 in the runs without population seeding and 0.52 for those with population seeding. The improvement in the confidence interval for best codes

	Variation operator	
R	Unary	Binary
7	(14.3,14.4)	(14.8,15.0)
9	(14.1,14.4)	(14.9,15.1)
11	(14.1,14.3)	(15.0,15.3)
13	(13.9,14.1)	(15.0,15.2)

TABLE I

THE ABOVE TABLE GIVES 95% CONFIDENCE INTERVALS ON THE SIZE OF THE BEST CODE FOUND IN 100 EVOLUTIONARY RUNS FOR FOUR VALUES OF R USING UNARY AND BINARY VERSIONS OF THE CONWAY VARIATION OPERATOR.

R	Lamarck 0%	Lamarck 5%
9	(14.9,15.1)	(15.6,15.8)
11	(15.0,15.3)	(15.7,15.9)
13	(15.0,15.2)	(15.7,15.9)
15	(14.9,15.2)	(15.6,15.8)

TABLE II

THE ABOVE TABLE GIVES 95% CONFIDENCE INTERVALS ON THE SIZE OF THE BEST CODE FOUND IN 100 EVOLUTIONARY RUNS FOR FOUR VALUES OF R USING THE LAMARKIAN MUTATION 0% OR 0.5% OF THE TIME. THE RUNS SUMMARIZED HERE USE A POPULATION SIZE OF 5, $W = 80$, AND THE BINARY VERSION OF THE VARIATION OPERATOR.

found seems to be offset by a decrease in the breadth of search.

E. Lamarkian Mutation

A Lamarkian version of the variation operator was implemented as follows. After applying the binary Conway operator, the resulting code C was used as if it were a code seed. The entire space $\mathcal{A}^n$ was enumerated in alphabetical order and each word was added to C if this did not violate the minimum distance condition relative to the words initially in C or added to it thus far by the Lamarkian mutation operator. This Lamarkian mutation was compared to the standard algorithm for a variety of values of R as shown in Table II.

The Lamarkian mutation operator helped substantially, significantly improving the mean size of the best code found. For $R = 9, 11$, the non-Lamarkian runs found codes of size 16 as their best-of-the-best while all other runs found codes of size 17. The number of codes of size 17 found was always enhanced by using the Lamarkian variation of the variation operator.

The Lamarkian mutation operator is very similar to the fitness function used in the code-seed algorithm and retains the undesirable feature of having exponential time complexity. The way it was applied, at a rate of half of one percent, still slowed the algorithm noticeably. Applying this operator at a low rate represents a point in a speed/quality trade-off curve that may merit additional study.

R	95% C.I.	Best size
10	(101.4,102.0)	92
30	(99.7,100.4)	105
90	(88.6,89.0)	105

TABLE III

THE ABOVE TABLE GIVES 95% CONFIDENCE INTERVALS ON THE SIZE OF THE BEST CODE FOUND IN 100 EVOLUTIONARY RUNS FOR THREE VALUES OF R, TOGETHER WITH THE BEST CODE SIZE FOUND IN ANY RUN. THE RUNS SUMMARIZED HERE USE A POPULATION SIZE OF 5, $W = 320$, AND THE BINARY VERSION OF THE VARIATION OPERATOR.

F. Increasing Algorithm Runtime

A collection of 100 runs was performed in which the number of mating events was increased from 100,000 to 400,000. The best code located in any of the runs improved from 17 (at 100,000) to 18 (at 400,000). The 95% confidence interval improved from (15.0,15.2) to (15.7,15.9). This indicates that additional runtime is substantially helpful.

G. Scaling for R on the $(6, 3)$-codes.

The results of the parameter study on R for $(7, 5)$-codes show that the correct value for R on $(7, 5)$-codes is larger than half the number of words in the best known $(7, 5)$-code. It is hard to deduce a pattern from one data point, and so this result was checked for $(6, 3)$-codes. The best-known $(6, 3)$-code has a size of 106. The results are summarized in Table III.

The results for the $(6, 3)$-codes do not have the same character as those for $(7, 5)$-codes: that good values for R are large relative to the size of an optimal result. In fact the good value for R, based on the evidence available, seems quite similar for the $(6, 3)$ and $(7, 5)$ codes.

V. DISCUSSION

The Conway variation operator has aspects of both crossover and mutation about it. It plays the role of crossover by combining words from two different codes. The random material introduced pulls in new material in a fashion similar to mutation. The combination of these two traditionally distinct functions is a choice resulting from experimentation; shuffling new words in with the words from two codes and then applying Conway's algorithm yields better performance than shuffling together two codes, applying Conway's algorithm, then shuffling in new words, and applying Conway's algorithm again. The former procedure also requires fewer calls to Conway's algorithm and less total random number for shuffling. The work in [1] demonstrates that the order in which words are considered matters. The discovery of new codes (not necessarily better) is substantially enhanced by shuffling a code together with new material, be it another code or random material.

The current set of experiments show that the parameter R, while controlling an analog of mutation rate, does not have the

same intuitive character as mutation in, say, the simple genetic algorithm. The values of R that yield the best performance seem absurdly large. For $(7, 5)$-codes the number of random words added yields best performance at levels above half the size of the current optimum code. For $(6, 3)$-codes the best value of R among those tested was 10; still pretty large. This is probably because of the greedy algorithm at the heart of the variation operator. A greedy algorithm is a very simple form of heuristic but it results in a more directed incorporation of the random material than is usual for the mutation operators in other evolutionary algorithms.

It is worth noting that a substantial fraction of all crossover events for the algorithm reported in [1] resulted in three word code seeds that were closer than the minimum allowed distance d. The algorithm handled this by assigning such code seeds a fitness of zero and allowing selection to remove them. The Conway variation operator cannot produce a code that violates its minimum distance constraint. This avoidance of completely non-viable codes is another feature in which the new algorithm improves on the old.

There are multiple views of the role of crossover. At one extreme it is assumed to assemble building blocks while at the other it is supposed to provide potentially innovative but essentially random leaps into new parts of the space (macro-mutations). For locating error correcting codes, Conway variation must act more in the latter than the former manner, simply because there are unlikely to be building blocks in the problem in its current representation. Code words are neither good nor bad by themselves, rather they achieve quality only in their geometric relationship under the relevant distance metric as members of codes. This situation makes building blocks that make independent contributions to fitness quite rare.

The order of the random material fed into algorithm 1 as part of the Conway variation operator is important. Figure 3 shows the result of applying Algorithm 2 to 1000 different orderings of the same set of 100 words of length 7 with the required minimum edit distance set to $d = 5$. Except for the fact the same 80 words were used each time this is the same procedure used to create initial population members. These initial population members are small relative to the size of an optimal code; increasing W improves them only slowly as W increases.

The time to generate a new code and compute its fitness drops substantially with the new code-search evolutionary algorithm when compared to the seed-based algorithm reported in [1]. The bottleneck in the older algorithm was the need to enumerate $\mathcal{A}^n$ when computing fitness. This is replaced, in the new algorithm, by a bottleneck in applying the variation operator which requires running algorithm 1, in a manner comparable to that used in the enumeration of $\mathcal{A}^n$, on a set of size at most $2m + R$, where m is the size of the largest code located thus far in a given run and R is the random material number. Empirically, the time to complete 100 runs searching for a good $(7, 5)$-code drops from overnight to about an hour; for $(6, 3)$-codes it drops from over a week to less than 8 hours. This confirms the theoretical expectation that the algorithm should be faster. The new algorithm gains the greatest advantage when $n - d$ is relatively small, making the best codes quite small relative to $\mathcal{A}^n$.

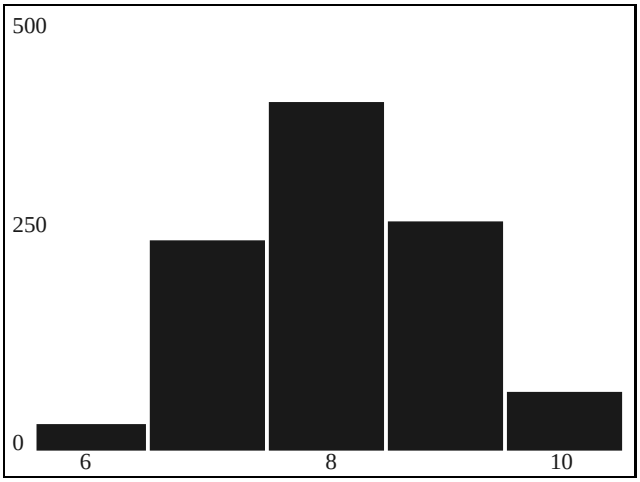

Fig. 3. A histogram of code sizes, 6-10, resulting from applying Algorithm 2 to 1000 shufflings of the same set of 80 random words of length 7 for minimum distance $d = 5$.

VI. Future Directions

The experiments reported here made a modest check of the relative utility of the unary and binary versions of the crossover operator. While the binary version of the variation operator was the clear winner, only a small number of tests were performed. The trinary and higher version of the operator have not been tested at all. Experiments to check the utility of the non-binary versions of the crossover operator are part of an ongoing project to understand what the best possible edit metric codes are. The most recent results from this project appear in [11].

The importance of the order in which words are presented to Algorithm 1, as demonstrated by the data in Figure 3, suggest that Algorithm 2 could be rewritten to exploit a technique similar to *brood recombination* in genetic programming[6], [7]. Brood recombination is used to compensate for the highly disruptive type of crossover used in genetic programming. A discussion of its merits appears on pages 158-161 of [2]. Rather than using a single application of crossover, several different crossovers are performed for a single set of parents and the best results are saved. The analog to this for Algorithm 2 would be to try shuffling the set Q from Algorithm 2 many times and saving the largest resulting code.

While this modification of the variation operator is likely to yield better results in the short term, a series of experiments would be required to make sure it does not act to place the search on a sub-optimal code more often than the current, softer, version of the search operator. The results on seeding the population with a single instance of the Conway code suggest this may be a danger. It seems unlikely that the Conway variation operator is as disruptive as the sub-tree crossover operation used in genetic programming. The use of a greedy algorithm gives the operator an intrinsic mechanism

for preserving substantial fitness. This means that if brood recombination based on shuffling order of the words does yield a benefit it will probably be for different reasons than those that cause brood recombination to be valuable in genetic programming.

At present there are many constructions for Hamming-distance codes. The definition of and constructions for cyclic codes, BCH codes, Goppa codes, Reed-Solomon codes, and convolutional codes can all be found in [8]. While it would be shocking if any of these Hamming-distance codes were also edit codes, they are sets of words already at large Hamming distance. There is good hope that they contain large subsets that are edit distance codes. Such *edit subcodes* may be good edit codes in their own right or may form good initial population members to enhance evolutionary search.

REFERENCES

[1] Daniel Ashlock, Ling Guo, and Fang Qiu. Greedy closure genetic algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1296–1301, Piscataway NJ, 2002. IEEE Press.

[2] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming : An Introduction*. Morgan Kaufmann, San Francisco, 1998.

[3] Richard A. Brualdi and Vera Pless. Greedy codes. *Journal of Combinatorial Theory(A)*, 64:10–30, 1993.

[4] Jessie K. Campbell. *Enumeration and Symmetry of Edit Metric Spaces*. PhD thesis, Iowa State University, 2005.

[5] Daniel Gusfi eld. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Boston, 1997.

[6] John R. Koza. *Genetic Programming*. The MIT Press, Cambridge, MA, 1992.

[7] John R. Koza. *Genetic Programming II*. The MIT Press, Cambridge, MA, 1994.

[8] Robet J. McEliece. *The Theory of Information and Coding*. Cambridge University Press, New York, NY, 1977.

[9] Joao Meidanis and Joao C. Setabal. *Introduction to Computational Molecular Biology*. Brooks-Cole, Pacifi c Grove, CA, 1997.

[10] F. Qiu, L. Guo, T.J. Wen, D.A. Ashlock, and P.S. Schnable. Dna sequence-based bar-codes for tracking the origins of ests from a maize cdna library constructed using multiple mrna sources. *Plant Physiology*, 133:475–481, 2003.

[11] S.K.Houghten, D.Ashlock, and J.Lenarz. Bounds on optimal edit metric codes. submitted to Discrete Mathematics, 2005.

[12] Gilbert Syswerda. A study of reproduction in generational and steady state genetic algorithms. In *Foundations of Genetic Algorithms*, pages 94–101. Morgan Kaufmann, 1991.

APPENDIX

Tables of Code Sizes.

This appendix summarizes, in three tables, the current state of knowledge about the size of DNA error-correcting codes for small code lengths. Table IV gives the size of Conway codes. Table V gives the performance of the seed-based and Conway variation operator based algorithms for those parameter values were at least 100 runs have been performed. For those parameter sets not used in tuning the algorithm, the algorithm was run with population size 5, random material inclusion rate $R = 13$, 0.2% use of the Lamarkian mutation operator, initial random list size $W = 320$s, run for 100,000 generations. Table VI gives the best possible size or, if this is not known, the range of possible sizes with the lower limit of the range being the best known thus far.

Table VI amalgamates results from all available sources. These include the size of the Conway code (which supplies the lower bound in many cases), exhaustive search for small lengths, both evolutionary algorithms mentioned in this paper, and three techniques for finding upper bounds. The first of these notes that the size of an optimal $(n+1, d)$-code cannot be more than $|\mathcal{A}|$ times the size of an (n, d)-code. If it were the $(n+1, d)$-code could be sorted by the first letter of all its members. The largest subset of the code with a uniform first letter would then be, by the pigeonhole principle, larger than the best (n, d)-code. Deleting that first letter from the subset would, however, yield an (n, d)-code, giving a proof of the upper bound by contradiction. The second upper bound technique simply notes that the size of an (n, d)-code over the edit metric cannot exceed the size of an (n, d)-code over the Hamming metric and so mines tables of known bounds on Hamming codes to obtain bounds on edit codes. The third technique is to perform an exhaustive search for codes with a given number of codewords. If no code with M codewords exists, then the upper bound is lowered to $M - 1$. Furthermore, if the best known code has $M - 1$ codewords, then we have shown that this is indeed the best possible. These exhaustive searches become significantly more difficult and time-consuming for every step closer one gets to the optimal value, therefore it is not always possible to obtain an upper bound close to the optimal value.

It is perhaps worth noting that the upper bounds in Table VI are much softer than the lower bounds. With the exception of exhaustive search, which can only be run for codes of length less than 7, they produce substantial overestimates of the true optimum code size. The lower bounds are obtained by producing examples of codes with heuristics that produce optimal codes for small word length where the optimal size is known.

Code Sizes	Minimum Distance						
Length	3	4	5	6	7	8	9
3	4	1	1	1	1	1	1
4	12	4	1	1	1	1	1
5	36	8	4	1	1	1	1
6	96	20	4	4	1	1	1
7	311	57	14	4	4	1	1
8	1025	164	34	12	4	4	1
9	3451	481	90	25	10	4	4
10	11743	1463	242	57	17	9	4
11	40604	4574	668	133	38	13	4

TABLE IV

SIZE OF (n, d)-CONWAY CODES FOR $n \leq 11$, $d \leq 9$.

Code Sizes	Minimum Distance						
Length	3	4	5	6	7	8	9
3	4	1	1	1	1	1	1
4	16	4	1	1	1	1	1
5	44-46	11	4	1	1	1	1
6	106-179	28-40	8	4	1	1	1
7	329-614	63-128	18-28	8	4	1	1
8	1025-2340	164-512	38-112	14-32	5	4	1
9	3451-9360	481-2048	90-448	26-128	11-20	4-5	4
10	1174-30417	1463-8192	242-1792	57-496	19-80	10-16	4-5

TABLE VI

BEST KNOWN CODE SIZES AND UPPER BOUNDS. DOUBLE ENTRIES ARE OF THE FORM BEST KNOWN-UPPER BOUND. SINGLE ENTRIES ARE KNOWN OPTIMAL CODE SIZES.

		Algorithm:		
Parameters:		Conway	Seed	Conway
Length	Distance	Code	Algorithm	Variation
5	3	36	41	42
5	4	8	11	10
6	3	96	106	105
6	4	20	25	24
6	5	4	8	8
7	3	311	329	*
7	4	57	63	*
7	5	14	18	18
7	6	4	7	*
9	7	10	*	11

* No results thus far

TABLE V

LARGEST CODE LOCATED USING THE SEED-BASED AND CONWAY VARIATION OPERATOR EVOLUTIONARY ALGORITHMS. THE SIZE OF THE CONWAY CODE INCLUDED AS A LOWER BOUND. THIS TABLE CORRECTS THE BEST-KNOWN VALUE FOR THE SIZE OF $(6, 5)$-CODES FROM AN EARLIER INCORRECT VALUE OF 9 TO THE CORRECT VALUE OF 8.