Edit Metric Decoding: Return of the Side Effect Machines

Sheridan Houghten
Dept. of Computer Science
Brock University
St. Catharines, ON, Canada
Email: shoughten@brocku.ca

Tyler K. Collins
Dept. of Computer Science
Brock University
St. Catharines, ON, Canada
Email: tk11br@brocku.ca

James Alexander Hughes
Computer Science Dept.
University of Western Ontario
London, ON, Canada
Email: jhughe54@uwo.ca

Joseph Alexander Brown
AI in Games Development Lab
Innopolis University
Innopolis, Tatarstan, Russia
Email: j.brown@innopolis.ru

***Abstract*—Side Effect Machines (SEMs) are an extension of finite state machines which place a counter on each node that is incremented when that node is visited. Previous studies examined a genetic algorithm to discover node connections in SEMs for edit metric decoding for biological applications, namely to handle sequencing errors. Edit metric codes, while useful for decoding such biologically created errors, have a structure which significantly differentiates them from other codes based on Hamming distance. Further, the inclusion of biologically-motivated restrictions on allowed words makes development of decoders a bespoke process based on the exact code used. This study examines the use of evolutionary programming for the creation of such decoders, thus allowing for the number of states to be evolved directly, not witnessed in previous approaches which used genetic algorithms. Both direct and fuzzy decoding are used, obtaining correct decoding rates of up to 95% in some SEMs.**

I. Introduction

In this study *side effect machines*, an offshoot of finite automata, are employed for efficient decoding of tags that have been incorporated into genetic constructs. The tags in question are designed to be resilient to errors that may be introduced during sequencing. Evolutionary programming is used to create side effect machines that are able to recover a high proportion of the original data.

A. Background

Error-correcting codes are used to retain information that may otherwise be lost during transmission over a noisy channel. Redundant information is added to the original message to form a *codeword*. The set of all allowed codewords forms a *code*. The codeword is transmitted in the place of the original message, and if noise occurs during transmission that changes the word transmitted, these errors are *detected* if the received word is not one of the allowed codewords. If the exact changes to the transmitted word can be identified then the errors can be *corrected*. Codes are designed so that the codewords are well separated – the further they are separated from one another, the higher the number of errors that can be corrected.

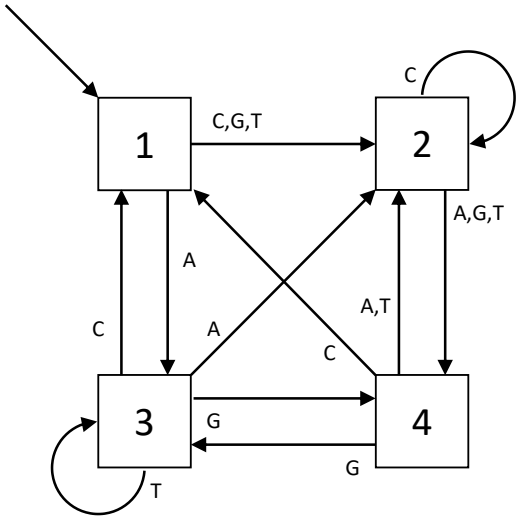
Traditionally, error-correcting codes were defined using *Hamming distance*. This measures the number of *substitution errors*, in which individual symbols are independently changed. For example, the words 101010 and 010110 are Hamming distance 4 from each other, as the first four symbols all differ between the two words, while the remaining two symbols are the same.

In some applications it is useful for a genetic construct to contain *tags*, which are short sequences incorporated into the genetic construct that provide identifying information. When this genetic construct is sequenced, errors may occur. The sequencer may misread a base, leading to a substitution error. It may also skip a base, causing a deletion, or it may read one that is not there, causing an insertion. The tags must be resilient to such errors, and so should be designed as codewords. In this case, however, it is inappropriate to use Hamming distance because it does not take into consideration insertions and deletions of symbols. Instead, *edit distance*, also known as *Levenshtein distance* [17], is the appropriate choice. The edit distance between two words is defined as the minimum number of insertions, deletions and substitutions required to transform one word into the other. If the fourth symbol ('0') in the word 101010 is deleted and then a '0' is inserted at the start, the word 010110 is created, and so these two words are at edit distance 2 from each other.

As the language of DNA is constructed from 4 possible symbols ('A', 'C', 'G', and 'T'), the tags are codewords in which the alphabet consists of these same 4 symbols. The code has a specified *minimum distance*, which is the smallest pairwise edit distance between codewords. A $(n, M, d)_q$ code is a code with M codewords of length n and with minimum distance d, where each codeword is constructed using an alphabet of q symbols. A code with minimum distance d is able to correct $\lfloor (d-1)/2 \rfloor$ errors. A code with a greater number of codewords has a greater number of possible tags and thus has more available labels to represent information. Meanwhile, the practicality of incorporating tags is dependent on their length. For a given length, as minimum distance (and hence level of error correction) increases, the maximum possible number of codewords decreases.

A number of previous studies have considered the problem of finding codes with as many codewords as possible, given length and minimum distance. Techniques include exhaustive search [15] along with various computational intelligence techniques (e.g. [3], [2]).

It should be noted that, along with the length, minimum distance and number of codewords, there may also be further



Received Pattern	c_1	c_2	c_3	c_4
ACCGTCAGTCTT	2	4	3	3
CTGCGTACGTCT	1	6	1	4
TCTAAAGCTGGC	2	5	1	4

Fig. 1. Example Four State SEM with Example Patterns and Output Vectors

considerations as to the suitability of a given codeword for a particular application. For example, there may be restrictions on the GC content of the codewords or on the existence of certain substrings, and in some cases the minimum distance also applies between codewords and reverse complements of codewords. Examples of studies that consider such further restrictions include [6], [19] and [20].

The process of finding and correcting errors is called *decoding*. *Maximum-likelihood decoding* is based on the assumption that the original (uncorrupted) word is the codeword closest to the received word. Some codes defined using Hamming distance have a mathematical structure allowing for particularly efficient decoding algorithms, making these codes very useful in practice. However, this is not even a remote possibility when considering edit-metric codes for biological applications: the codes are simply sets of codewords with little in the way of mathematical structure connecting them, even before considering any additional biological restrictions such as those described above.

As a result, previous work in [8] [9] [16] employed *side effect machines* for efficient decoding of such codes.

II. Side Effect Machines

Side Effect Machines (SEMs) extend finite state machines by placing a counter on each node. Each time a state is entered, its counter is incremented.

Figure 1 shows a four state side effect machine. As a convention the SEM always begins on a set state, usually state 1. The counters are represented by the classification vector $c = (c_1, c_2, c_3, c_4)$ where c_i, $1 \leq i \leq 4$, records the number of times that state i has been entered. For example, an input of ACCGTCAGTCTT gives a path through the states of 312422433124, which yields the classifying vector $c = (2, 4, 3, 3)$, since state 1 is visited 2 times, state 2 is visited 4 times, and states 3 and 4 are each visited 3 times.

Side Effect Machines act as a transducer between an input language, Σ, onto a $\mathbb{N}^S$ string, where S is the maximum number of states in the SEM representation. Brown [7] examines their placement into the Chomsky Hierarchy and determines that the key to their recognizing abilities is based upon the power of the examination of the output vector, as the machine is limited by its finite number of states; however, it is more powerful than finite state recognizers as it is also able to count.

Genetic algorithms have been used to evolve SEMs in a variety of different ways. Those evolved SEMs have been used to classify DNA sequences (see e.g. [4] [18]) and for motif discovery [1]. In a study examining the classification of transposable elements [5], it was revealed that biological meaning could be extracted from SEMs.

A. Decoding using Side Effect Machines

As stated earlier, SEMs have also been used for efficient decoding of edit metric codes. Decoding a received word w involves finding the codeword that is closest (i.e. has the smallest edit distance) to w. For words of length n, edit distance is calculated using a $O(n^2)$ dynamic programming algorithm. To find the closest codeword, therefore, requires this operation to be performed for all M codewords.

To avoid this heavy cost, a SEM could be used, with the SEM designed to minimize the Euclidean distance between the classifying vector of the received word w and that of the correct codeword. Specifically, in earlier studies genetic algorithms were used with a goal of evolving such SEMs.

In the first study [8] genetic algorithms were used to evolve SEMs for decoding of $(12, 55, 7)_4$ edit metric codes. In the direct approach, the classifying vector of the received word w was compared to the classifying vectors of all codewords, with w decoded as the codeword v with the closest classifying vector. Note that as SEMs are probabilistic, additional verification can be provided by calculating the edit distance between w and v. If this distance is within the error-correcting capacity of the code then the SEM has decoded correctly; otherwise, a response such as "unable to decode" should be provided. In the study the SEM was also used for fuzzy classification: the classifying vectors of all codewords were ranked in order of their Euclidean distance to the classifying vector of the received word, and all those within a given tolerance were checked, in order, using edit distance. The best SEM was able to correct 81% of words with 1 or 2 errors, in both training and testing data; when extended to fuzzy classification, this improved to over 90% for a tolerance level of 3.

The second study [9] expanded upon the first by studying five different codes, all of length 12 and minimum distance 7, and with between 54 and 56 codewords. Performance was

found to improve rapidly as the number of states in the SEMs increased, however with negligible improvement after the number of states reached 12. The study also considered a hierarchical classification of multiple SEMs.

The third study [16] used three different approaches. The first was a basic GA with a direct representation in which the SEMs were the chromosomes. The other two both used the recentering-restarting evolutionary algorithm, one with a direct representation and the other with an indirect representation based upon transpositions. Three codes were studied, all of length 12 and minimum distance 7, and with between 55 and 60 codewords. As in [8], both direct and fuzzy classification were considered. For direct classification, performance for the direct representation again improved significantly as the number of states increased, up to about 12. For direct classification, the indirect representation did not compete well with the direct representation, although again there was a general improvement when the number of states increased. However, the indirect representation performed very well when using fuzzy classification; in this case it appeared to be better able to generalize, as with four states some SEMs were able to correctly decode over 99% of words with 1 or 2 errors.

All of the above clearly demonstrates the importance of carefully studying the number of states in SEMs used for decoding. Our approach in the current study is to allow the number of states to vary during evolution. This is accomplished using Evolutionary Programming.

III. Evolutionary Programming

Evolutionary Programming (EP) was developed by L. Fogel [14] to act as method of developing finite state automata from a given input string to match with an expected output. It is considered one of the foundational algorithms in the field of Evolutionary Algorithms (EAs) due to its contemporary development to genetic algorithms, genetic programming, and evolutionary strategies, but currently it is perhaps one of the least frequently used. The original formulation of the algorithm used a population of finite state machines, which were utilized on an online controller to the problem with a constant flow of input and outputs. Later extensions [13] would use different representations, and a system of controlling the selection pressure by adding a "bout" system, which is similar to tournament selection in genetic algorithms.

More recently, a modified EP method was used in mixed wireless controllers in an online setting in order to control the direction of transmission between nodes in a mixed basis network. These controllers have been found to provide coverage which meets or exceeds the 802.11 DCF standard [12][11][10].

The automata used in these systems are Finite State Machines (FSM).

A. Representation

As mentioned above, side effect machines are utilized as representations. The machine is stored as a transition matrix, with size equal to the number of states. Each entry corresponds to a particular state, with a state number and four output transitions (A, C, G and T). There is also a designated initial state. When the machine is given an input string, a pointer is placed on the initial state, the string is parsed and run through the machine, and an output vector is generated. This output vector has length equal to the size of the machine, and is used to collect the counters.

Note that while the upper and lower bounds on the number of states provide only the available states, the SEM might not have all states fully connected as the mutations do not ensure connectivity. This allows for null mutations, allowing the machine to transition over a number of additions. This however does allow for a potential use of these states as bloat, protecting the heritability, though requiring a post processing of the machines to discover the simplified machine.

B. Mutations

Mutation is a unary operation selecting from the following uniformly at random as long as the operation is not deemed illegal based on the current parent:

1) Change Initial State: Select a state uniformly at random, and make this selected state the initial state of the machine.

2) Add a State: Place a new state into the machine, by first connecting all outgoing transitions, and then selecting a transition uniformly at random in the machine. This later action is to attempt to provide some connection of this new state into the machine, although as stated above we do not require the machine to be connected.

This operation is illegal if the parent is at the upper bound of the number of allowed states.

3) Delete a State: Select a state uniformly at random in the machine. The input edges to the state are then passed through to the outputs of that state: e.g. if state 1 transitions to state 2 by C and state 2 transitions to state 3 by C, then when deleting state 2, state 1 will now transition to state 3 by C.

If the deleted state is the initial state, then the initial state is changed to the next state numerically.

This operation is illegal if the parent is at the lower bound of the number of allowed states.

4) Change a Transition: Take a transition in the machine and change its ending state to another available state, chosen uniformly at random.

Note that in other versions of EP there is also an option to change the output. However, since SEMs are extensions which work on increasing a counter, no equivalent mutation is utilized in the current study.

IV. Methodology

Three codes were used in the experimental analysis. For comparison with earlier work, these are the same codes used in [16]. The first code, which we label Code55, is a $(12, 55, 7)_4$ code, i.e. a quaternary code (for symbols 'A', 'C', 'G' and 'T') consisting of 55 codewords each of length 12 and with minimum edit distance 7. The other two codes, which we label Code60-1 and Code60-2, are both $(12, 60, 7)_4$ codes. For all of these codes, the maximum number of errors that can be corrected is $\lfloor (7-1)/2 \rfloor = 3$.

TABLE I
PARAMETER VALUES FOR INITIAL SETS OF EXPERIMENTS

	Experiment 1	Experiment 2
Population Size	300	300
Number of Generations	1250	1250
Bout Size	10	10
Minimum Number of States	4	4
Maximum Number of States	18	18
Probability of Changing a Transition	0.60	0.75
Probability of Changing Initial State	0.10	0.05
Probability of Adding a State	0.15	0.10
Probability of Removing a State	0.15	0.10

Two sets of words (error patterns) were produced for each of the codes, one used strictly for training and one used strictly for verification. These sets each consist of, for each codeword, 12 words with a single error, 12 with two errors, and 12 with three errors, where the errors are a combination of insertions, deletions and substitutions of symbols. Thus the training set and verification set for Code55 both have a size of $55 \times 12 = 660$ for each of distances 1, 2 and 3, while the training set and verification set for the other two codes both have a size of $60 \times 12 = 720$ for each of distances 1, 2 and 3. Each of these words will be run through the SEMs produced. If error pattern x was generated from codeword y by 1, 2 or 3 errors (any combination of insertions, deletions and substitutions) then no other codeword can be closer than y because the minimum distance between codewords is 7. Therefore upon running x through the SEM, if the SEM returns y as the classification then it has correctly decoded x.

A perfect score is obtained by a given SEM if it correctly decodes all error patterns. Thus for Code55 a perfect score, whether for the training set or the verification set, is $660 \times 3 = 1980$ while for both Code60-1 and Code 60-2 a perfect score is $720 \times 3 = 2160$.

A. Initial Experiments — Direct Classification

Two initial sets of experiments were performed, with the parameters determined empirically. These parameters are summarized in Table I. Further, to ensure that the maximum number of states was not too restrictive, two matching sets of experiments were later performed with the maximum number of states increased to 24.

B. Fuzzy Experiments

Fuzzy classification is explained in Section II-A. For comparison with earlier work, the same tolerance value, i.e. a Euclidean distance of 3, was used in the current study. The same parameter values were used as in the initial experiments.

V. RESULTS

The summary statistics for Experiment 1, for both direct and fuzzy classification, are presented in Table II. The median number of states, along with the interquartile range for the 100 runs for each code are also presented. For the rows labelled *All*, the values are the median total fitness, i.e. the number of correctly decoded error patterns of the 1980 in Code55, and of the 2160 for each of codes 60-1 and 60-2, along with the interquartile range of these values. This is also broken up according to distances 1, 2, and 3. For these, the maximum partial fitness is 660 per distance for Code55 and 720 per distance for Code60-1 and Code60-2. Table III contains the equivalent summary statistics for Experiment 2.

As can be seen from these tables, the results were very similar in terms of fitness for both experiments, when considering both direct and fuzzy classification. Violin plots showing the distribution of fitness for training and verification, for both direct and fuzzy classification, are shown in Figure 2. In this figure, the y-axis is the total number of error patterns that the machines were able to decode for each distance. Recall that this is a maximum of 660 for Code55 and 720 for both Code60-1 and Code60-2. In each plot, the training and verification distributions for both direct and fuzzy classification are presented for the 3 distances. As one would expect, the ability of the machines to decode decreases as the distance increases. As one would also expect, the machines were able to correct more errors with the fuzzy decoding, and this is most significant for distance 3. It is thus easy to argue that the very small increase in runtime required by fuzzy classification is worthwhile in this case. In both experiments, the best machines corrected $90 - 91\%$ of distance 1 error patterns using direct classification, and $94 - 95\%$ using fuzzy classification.

Between the two experiments there is a small difference in terms of the median number of states. It is important to note that any given machine may not actually use all of its states, i.e. some may be unreachable from the start state. It was found that in all experiments, the mean number of states tended to slowly increase throughout evolution, but then flatten out. The slight propensity towards larger machines could be a manifestation of *bloat*.

The distribution of final machine sizes is shown in Figure 3. This shows some inconsistency in the distributions within the 100 runs for each of the experiments. Because it was observed that in some cases there were a significant number of larger machines, a brief investigation was performed of experiments in which the number of allowed states ranged from 8 to 24. However, these performed relatively poorly and so no further investigation along these lines was performed.

Figure 4 shows a comparison of the number of states in the machine to their accuracy (error or fitness) for each of the 100 runs for each experiment set. This figure shows that machines with more states typically perform better than those with fewer; however, after a certain point (typically 14–16 states) the difference is negligible. In Experiment 1, as the number of states increases, there appears to be more variance in the results. As one would expect, the training fitness was slightly better than the verification fitness.

Figures 5 and 6 show the training vs. verification fitness for direct and fuzzy classification respectively. In the direct classification, it can be seen that the larger machines (red) performed the best and the smaller machines (blue) performed the worst. It is interesting to ask whether larger or smaller machines tend to more greatly overfit their training data. To

TABLE II
SUMMARY STATISTICS FOR EXPERIMENT 1

		Direct								Fuzzy					
		Training			Verification					Training			Verification		
	Dist.	Max	Median	IQR	Max	Median	IQR	States	IQR	Max	Median	IQR	Max	Median	IQR
Code55	All	1477	1390.0	± 33.75	1437	1337.5	± 31.0	13.5	± 3.0	1704	1615.5	± 38.0	1698	1594.5	± 38.125
	1	597	570.0	± 9.0	590	554.5	± 11.5			623	598.0	± 9.0	624	588.5	± 10.5
	2	525	487.0	± 10.5	512	476.0	± 10.0			582	553.0	± 12.125	587	552.0	± 12.5
	3	378	336.5	± 15.125	366	303.0	± 15.125			513	468.5	± 21.0	504	457.0	± 15.875
Code60-1	All	1576	1497.5	± 34.125	1532	1428.0	± 32.25	13.0	± 2.5	1852	1744.0	± 34.0	1823	1711.5	± 36.625
	1	652	613.0	± 10.625	644	607.0	± 11.625			678	641.5	± 9.5	670	638.0	± 11.25
	2	561	520.0	± 12.125	542	501.0	± 12.125			639	596.0	± 11.0	636	584.5	± 11.375
	3	410	358.0	± 19.125	365	317.0	± 19.125			539	502.0	± 14.125	547	485.0	± 18.25
Code60-2	All	1561	1465.0	± 38.0	1497	1401.0	± 35.25	13.0	± 3.0	1863	1737.5	± 38.75	1854	1712.0	± 42.25
	1	647	607.0	± 15.125	635	599.5	± 14.25			681	646.0	± 12.125	679	640.0	± 12.5
	2	553	518.5	± 15.125	535	495.5	± 12.125			644	600.5	± 13.0	641	588.5	± 12.625
	3	383	339.5	± 19.0	351	309.5	± 19.0			543	490.5	± 16.625	553	478.5	± 21.25

TABLE III
SUMMARY STATISTICS FOR EXPERIMENT 2

		Direct								Fuzzy					
		Training			Verification					Training			Verification		
	Dist.	Max	Median	IQR	Max	Median	IQR	States	IQR	Max	Median	IQR	Max	Median	IQR
Code55	All	1475	1374.5	± 34.125	1425	1333.5	± 31.625	11.0	± 2.0	1704	1606.0	± 40.125	1690	1593.0	± 38.75
	1	593	568.5	± 10.125	593	553.0	± 9.625			623	594.0	± 11.125	621	586.0	± 10.0
	2	524	478.0	± 11.75	510	476.5	± 9.625			583	548.0	± 12.75	578	551.0	± 13.625
	3	372	329.0	± 14.625	354	305.0	± 14.625			513	466.5	± 14.875	493	453.0	± 13.75
Code60-1	All	1593	1474.0	± 41.0	1531	1422.0	± 34.5	11.0	± 2.0	1874	1751.0	± 37.0	1847	1726.0	± 39.5
	1	646	614.0	± 10.0	643	610.0	± 10.5			680	642.0	± 10.0	674	644.0	± 12.5
	2	558	515.0	± 14.0	544	498.0	± 12.5			640	601.0	± 12.0	637	587.0	± 13.5
	3	416	347.0	± 23.5	374	317.0	± 23.5			559	505.0	± 17.5	545	491.0	± 18.5
Code60-2	All	1585	1456.0	± 44.5	1495	1401.0	± 39.0	12.0	± 2.5	1853	1719.0	± 46.0	1856	1694.0	± 49.0
	1	639	606.0	± 14.5	635	597.0	± 17.0			672	640.0	± 12.0	672	637.0	± 13.5
	2	563	509.0	± 15.5	526	494.0	± 13.0			642	597.0	± 15.0	633	588.0	± 15.0
	3	391	341.0	± 18.0	353	309.0	± 18.0			550	485.0	± 20.0	562	466.0	± 26.0

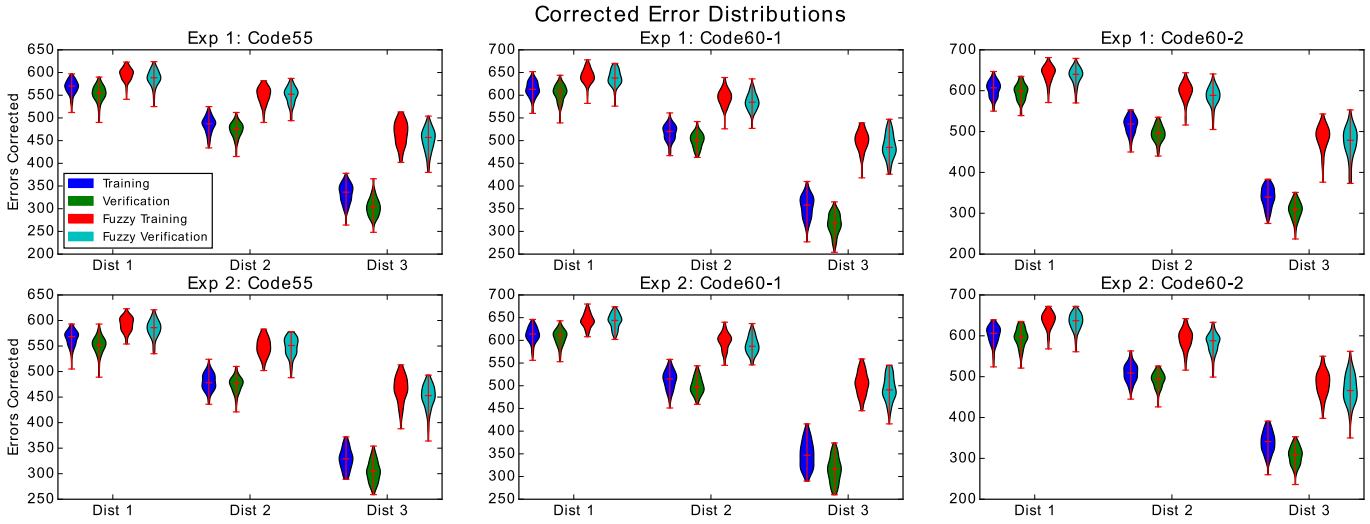


Fig. 2. Violin plots showing the distribution of correctly decoded error patterns for the 100 runs for each set of experiments

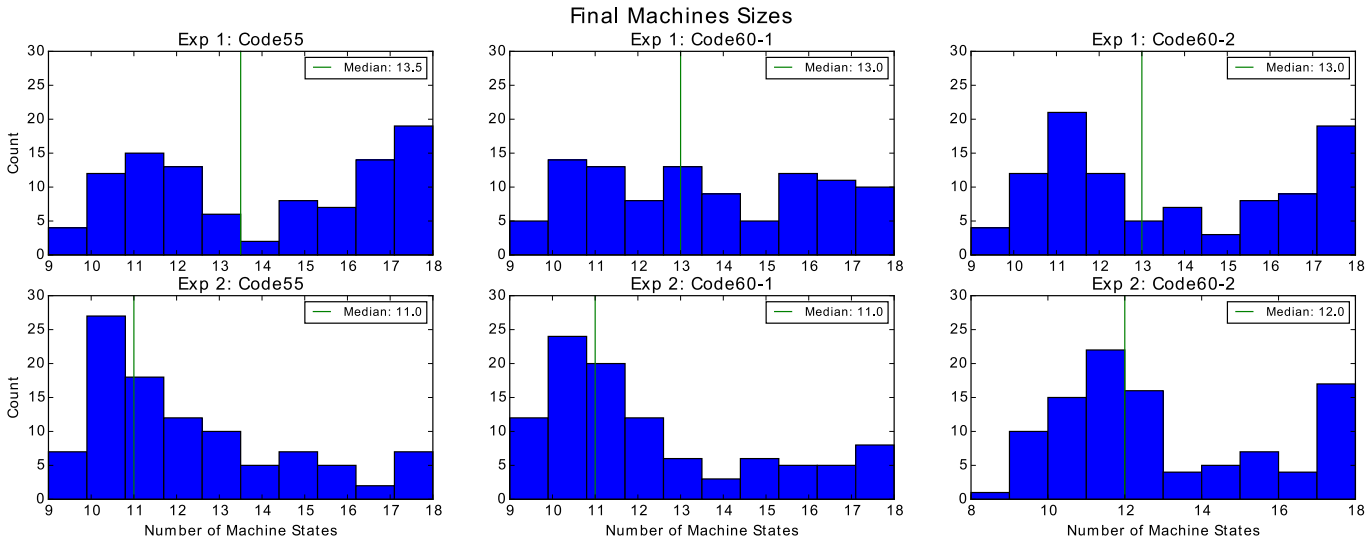


Fig. 3. Distribution of Machine Sizes and Median of the 100 Runs

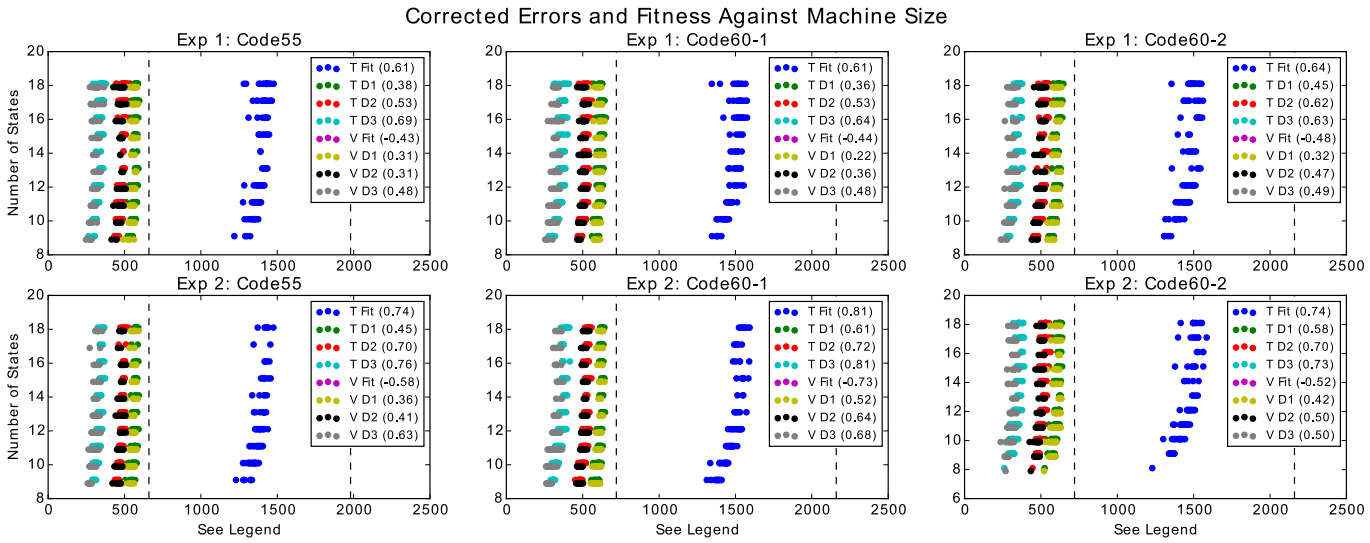


Fig. 4. A comparison of the number of states in the machine to their accuracy (error or fitness) for each of the 100 runs for each experiment set. For the training and verification fitness, the x-axis corresponds to overall fitness (higher is better), and for the training and verification distances (D1, D2, D3), the x-axis is the number of incorrectly classified error patterns (closer to zero is better).

answer this question, the slope of the line of best fit (noting that the data appears to be linear) was found. For Experiment 1 the slopes were 0.87, 0.77, and 0.84, and for Experiment 2 the slopes were $0,84$, 0.81, and 0.78. The fact that these slopes are all less than 1 indicates that the larger machines had a higher difference between training and verification fitness relative to the smaller machines. One should be careful using this as an indication of overfitting however, since the larger machines typically still have the best verification fitness.

With the fuzzy analysis, the machine sizes seem to have little impact on the quality of results and the difference between the training and verification errors is much lower. The slopes of the lines of best fit are 1.01, 1.01, and 1.10 for Experiment 1, and 0.98, 0.97, and 1.02 for Experiment 2. This provides an indication that the best performing machines are not overfitting, although it should be noted that the evolution was based on the direct decoding.

Table IV presents the p-values obtained by comparing the results from Experiment 1 with those from Experiment 2, for both the direct and fuzzy classification, and for training and verification. These p-values were calculated with Mann-Whitney U tests. Those values less than 0.05, indicating a significant difference, are highlighted in bold. As can be seen from this table, in general there is no significant difference between Experiment 1 and Experiment 2. The main exception to this is in the training dataset for the direct classification, most notably for Code55.

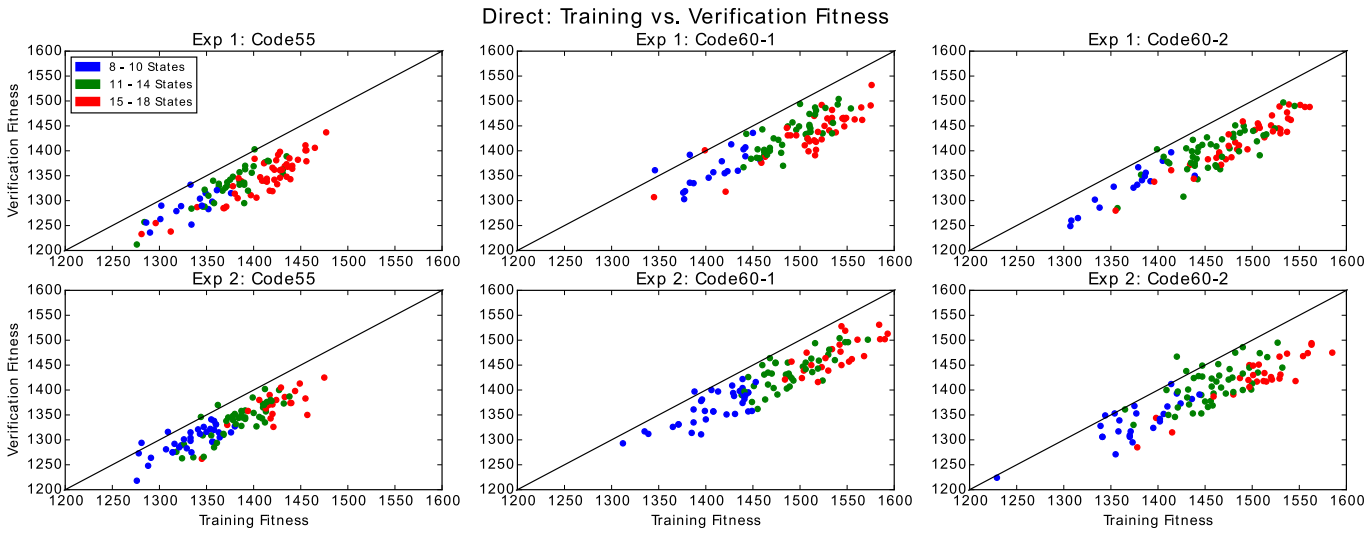


Fig. 5. Training vs. verification fitness for direct classification, for each of the 100 runs for each experiment. The points are colored to show small, medium, and large sized machines. Any point below the $x = y$ line means that it had a better final training fitness than verification fitness, and conversely for points above the line.

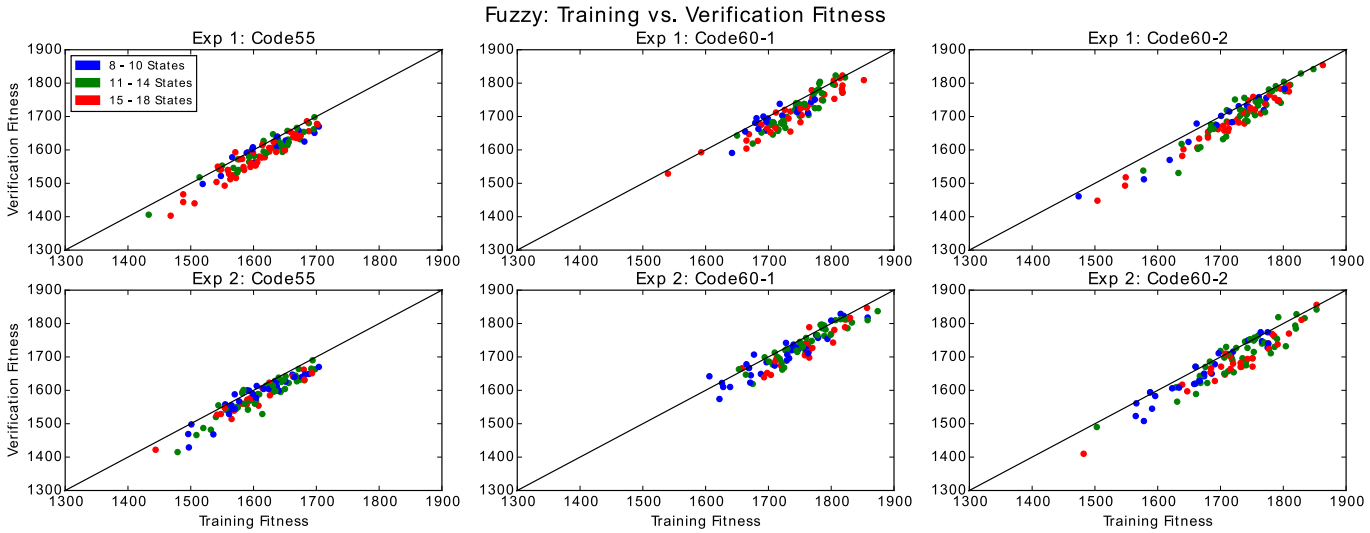


Fig. 6. Training vs. verification fitness for fuzzy classification, for each of the 100 runs for each experiment. The points are colored to show small, medium, and large sized machines. Any point below the $x = y$ line means that it had a better final training fitness than verification fitness, and conversely for points above the line.

TABLE IV
P-VALUE TABLE COMPARING DISTRIBUTION OF RESULTS FROM EXPERIMENT 1 AND EXPERIMENT 2.

		Direct		Fuzzy	
	Dist.	Training	Verification	Training	Verification
Code55	All	**2.05e-02**	4.20e-01	2.51e-01	2.85e-01
	1	1.38e-01	1.80e-01	**4.95e-02**	5.01e-02
	2	**1.48e-02**	4.01e-01	1.82e-01	3.80e-01
	3	**3.98e-02**	4.03e-01	4.50e-01	4.83e-01
Code60-1	All	7.89e-02	3.94e-01	1.00e-01	**4.82e-02**
	1	3.72e-01	2.87e-01	2.51e-01	6.58e-02
	2	7.88e-02	2.71e-01	1.16e-01	1.21e-01
	3	**3.50e-02**	3.05e-01	8.28e-02	**2.97e-02**
Code60-2	All	1.62e-01	3.57e-01	9.70e-02	9.91e-02
	1	3.85e-01	4.53e-01	7.93e-02	1.80e-01
	2	**1.36e-02**	2.34e-01	5.54e-02	8.19e-02
	3	4.91e-01	4.06e-01	1.80e-01	1.08e-01

VI. CONCLUSIONS AND FURTHER WORK

This is, to the best of our knowledge, the first use of evolutionary programming in evolving SEMs. The results are very competitive with earlier studies. In addition, the value of being able to easily modify the number of states during the evolution cannot be overstated.

The results indicate a preference for larger machines, with the increase in fitness being negligible after approximately 14–16 states. This is similar to [9], which obtained its best results with 12 states. With that being said, however, it is important to recognize that the current study is the *only* one which does not force an exact number of states, and furthermore there is no guarantee that all of the states are actually reachable

from the start state. An outlier is the work described in [16], which obtained very good results with only 4 states. Although both initial experiments in the current study do have a lower bound of 4 states, the allowed range in the number of states is possibly too large to allow for good small machines to be evolved.

Important future work includes analyzing the best machines generated to determine the exact count of states that are actually reachable from the start state. This could be used to then simplify the resulting machines.

In [5], it was shown that biological meaning could be extracted from SEMs. This is also an interesting idea to apply here, with the possibility of identifying some underlying structure of the codes and/or the words that are most easily decoded.

The decoders still show difficulty on error patterns that are higher distances from codewords. This is of course not surprising, given that these are the exact words that are closest to not only the correct codewords, but also other codewords. In the current study, in some cases these error patterns are distance three from the correct codeword and may be only distance 4 from other codeword(s). Although the fuzzy decoders greatly improve results for these error patterns, it is a definite future goal to improve the rate of correct decoding for such words.

Finally, the codes used in the current study were chosen because they are the same as those used in earlier work. It is important to expand the analysis to include other codes, in particular for different lengths, numbers of codewords and minimum distance.

ACKNOWLEDGEMENTS

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada.

This research was also enabled in part by support provided by WestGrid https://www.westgrid.ca/ and Compute Canada www.computecanada.ca.

REFERENCES

[1] Farhad Alizadeh Noori and Sheridan Houghten. A multi-objective genetic algorithm with side effect machines for motif discovery. In *2012 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pages 275–282. IEEE, 2012.

[2] Daniel Ashlock and Sheridan Houghten. Hybridization and ring optimization for larger sets of embeddable biomarkers. In *2017 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–8. IEEE, 2017.

[3] Daniel Ashlock, Sheridan K. Houghten, Joseph Alexander Brown, and John Orth. On the synthesis of dna error correcting codes. *Biosystems*, 110(1):1–8, 2012.

[4] Daniel Ashlock and Elizabeth Warner. Side effect machines for sequence classification. In *2008 Canadian Conference on Electrical and Computer Engineering*, pages 001453–001456. IEEE, 2008.

[5] Wendy Ashlock and Suprakash Datta. Distinguishing endogenous retroviral ltrs from sine elements using features extracted from evolved side effect machines. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 9(6):1676–1689, 2012.

[6] Nabil Bennenni, Kenza Guenda, and T Aaron Gulliver. Greedy construction of dna codes and new bounds. *arXiv preprint arXiv:1505.06262*, 2015.

[7] Joseph A. Brown. On side effect machines as a representation for evolutionary algorithms. In *2015 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–8, Aug 2015.

[8] Joseph A. Brown, Sheridan K. Houghten, and Daniel Ashlock. Edit metric decoding: A new hope. In *Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering*, C3S2E '09, pages 233–242, New York, NY, USA, 2009. ACM.

[9] Joseph A. Brown, Sheridan K. Houghten, and Daniel Ashlock. Side effect machines for quaternary edit metric decoding. In *2010 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–8, May 2010.

[10] Jason B. Ernst and Joseph A. Brown. An online evolutionary programming method for parameters of wireless networks. In *2011 International Conference on Broadband and Wireless Computing, Communication and Applications*, pages 515–520, Oct 2011.

[11] Jason B. Ernst and Joseph A. Brown. Co-existence of evolutionary mixed-bias scheduling with quiescence and ieee 802.11 dcf for wireless mesh networks. In *2012 26th International Conference on Advanced Information Networking and Applications Workshops*, pages 678–683, March 2012.

[12] Jason B. Ernst and Joseph A. Brown. Performance evaluation of mixed-bias scheduling schemes for wireless mesh networks. *International Journal of Space-Based and Situated Computing*, 3(1):22–34, 2013. PMID: 51984.

[13] Lawrence J. Fogel. The future of evolutionary programming. In *1990 Conference Record Twenty-Fourth Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1036–, Nov 1990.

[14] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence though Simulated Evolution*. John Wiley & Sons, New York, 1966.

[15] Sheridan K Houghten, Daniel Ashlock, and Jessie Lenarz. Construction of optimal edit metric codes. In *Information Theory Workshop, 2006. ITW'06 Chengdu. IEEE*, pages 259–263. IEEE, 2006.

[16] James A. Hughes, Joseph A. Brown, Sheridan Houghten, and Daniel Ashlock. Edit metric decoding: Representation strikes back. In *2013 IEEE Congress on Evolutionary Computation*, pages 229–236, June 2013.

[17] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[18] Andrew McEachern, Daniel Ashlock, and Justin Schonfeld. Sequence classification with side effect machines evolved via ring optimization. *Biosystems*, 113(1):9–27, 2013.

[19] Jing Sun, Sheridan Houghten, and Jonathan Ross. Bounds on edit metric codes with combinatorial dna constraints. *Congressus Numerantium*, 204:65–92, 2010.

[20] Bin Wang, Xiaopeng Wei, Jing Dong, and Qiang Zhang. Improved lower bounds of dna tags based on a modified genetic algorithm. *PloS one*, 10(2):e0110640, 2015.