Evolved Digital Circuits and Genome Complexity

Morten Hartmann, Per Kristian Lehre and Pauline C. Haddow
Complex Adaptive Organically-Inspired Systems Group (CAOS)
Dept of Computer Science
The Norwegian University of Science and Technology (NTNU)
{mortehar,lehre,pauline}@idi.ntnu.no

Abstract

A major issue with evolutionary computation is the lack of scalability with regards to the problem size. In the field of digital circuit design this fact severely limits the size and complexity of the circuits that can be evolved. Developmental approaches are being suggested as a possible remedy to the scalability issue. Earlier work indicated a close relationship between complexity and genotype-phenotype mapping. We investigate the genotypic complexity of evolved digital circuits and the complexity distribution of the search space by using a measure based on Lempel-Ziv compression. Also addressed is the question whether this knowledge could potentially be used to shrink the evolutionary search space.

1 Introduction

Evolutionary Algorithms (EAs) do not scale well with increased problem size. Increasing the number of elements in the genotype drastically increases the dimensionality of the search space.

One approach to reducing the search space and thus potentially improving the scalability of EAs is artificial development (AD) [19]. Inspired by biological development and the fact that the human genome is exceedingly small compared to a grown human being, AD could allow EAs to operate using smaller genotypes. There are many issues that need to be addressed before artificial development can be effectively applied to EAs. The authors believe that the understanding of the relation between complexity and genotype-phenotype mapping is one of the key issues.

In the search for a better understanding of genotypic complexity, the approach taken herein is to evaluate the complexity of genotypes using the Lempel-Ziv compression algorithm, assuming that interesting genotypes have some form of regularity. Interesting genotypes herein is genotypes who's phenotypic function is correct with respect to the goal of the EA. The complexity value is the length of the compressed genotype. If the interesting genotypes can be compressed, one can potentially search in the smaller space of a size defined by the length of the compressed genotypes.

Earlier work [11, 12] identified a close relationship between complexity and AD. The investigation of development mapping in [12] showed that the mappings output distribution is concentrated in a range of medium complexity, whilst the distribution of direct encoding is strongly concentrated on the highly complex range.

To investigate the potential of putting those results to practical use, this paper addresses the field of Evolvable Hardware (EHW) in which EAs are employed in the design and optimization of hardware. Specifically, the complexity of specific evolved digital circuits under different encoding methods, are investigated and related to the complexity of random genotypes (effective search space) and random bitstrings. The evolved circuits are 2-bit multipliers and adders evolved using Cartesian genetic programming (GP) [16, 8].

Section 2 briefly outlines related work. The theoretical background and an explanation of our complexity measure is given in section 3. The measured complexity of the circuit genotypes is presented in section 4. We discuss the results in section 5, and draw conclusions as well as outline future work in section 6.

2 Related Work

Reducing the genotype size with developmental mappings was considered as a possible approach to solve the scalability problem in complex evolutionary design optimization tasks in [9, 3, 5]. At the same time, comparative studies between direct encoding and developmental mapping indicated that in those cases direct encodings were at least as efficient as developmental mappings [6, 4, 18]. Haddow, Tufte and van Remortel suggested the use of Lindenmayer systems [14] in order to shrink the genotype in the evolution of digital circuits [7]. Miller has used developmental mapping to evolve parity-circuits and bi-

nary adders [17]. Other recent work on artificial development includes the use of fractal proteins [1], biological approaches [10, 20] and the building of knowledge into development rules in circuit design [21].

3 Complexity

The complexity measure used herein is theoretically based on Kolmogorov complexity. Kolmogorov complexity is a quantitative measure of complexity based on Turing machines [13].

Definition 1. *The Kolmogorov complexity $C(x)$ of a natural number x (or the corresponding binary string) is defined as*

$$C(x \mid y) \quad = \quad \min_{p \in \mathcal{N}} \{ \ell(p) \mid \phi(< p, y >) = x \}, \quad (1)$$

where ϕ is the partial recursive function corresponding to a fixed universal Turing Machine.

Informally, the Kolmogorov complexity of a binary string is the length of the shortest program which outputs that string on a fixed universal Turing Machine and then halts.

Kolmogorov complexity is not computable. Yet there are several examples where theory of Kolmogorov has led to practical applications. In these cases, Kolmogorov complexity is often approximated with a compression algorithm like Lempel-Ziv [22], see e.g. [2] where this replacement is justified for their application. Lempel-Ziv is also used herein as the practical measure of complexity and we refer to it as LZ-complexity.

3.1 Incompressibility lemma

Before looking at the specific case of genotype complexity, let us first consider complexities of bitstrings in general.

Certain bitstrings are highly compressible. For example a repeating bitstring of 101010..10 can be easily described. These highly compressible bitstring contain what we informally call regular patterns.

On the other hand no compression algorithm can compress every bitstring, so certain bitstring must be incompressible. As we will see, most random bitstrings are incompressible or close to incompressible. In terms of Kolmogorov complexity, an upper bound on the complexity of an incompressible string, is the length of a program which contains a copy of the string in the program, and a routine to print that string. This program will be slightly longer than the bitstring, yielding a Kolmogorov complexity slightly higher than the length of the bitstring.

If we consider the set of bitstrings of a given length n, we know that the complexity values of those bitstrings will approximately be in the range 0 to n. But how will they be distributed within this range?

Consider the example in figure 1. Among bitstrings of length 4, no more than 1 bitstring has complexity less than 1, no more than 3 bitstrings have complexity less than 2, no more than 7 bitstrings have complexity less than 3 and no more than 16 bitstrings have complexity less than 4. At least one bitstring has the maximum complexity of 4. The consequence is that the general compressibility of the bitstrings is very low. This argument generalizes for sets of bitstrings of any length n:

Theorem 1 (The Incompressibility Lemma [13]). *Let c be a positive integer. For each fixed y, every finite set A of cardinality m has at least $m(1 - \frac{1}{2^c}) + 1$ elements x with $C(x|y) \geq \log m - c$.*

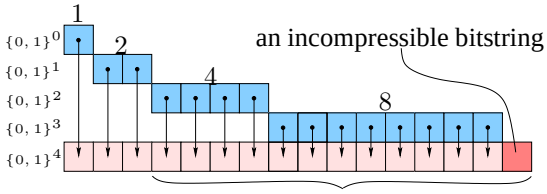


Figure 1. The Incompressibility Lemma

3.2 Implications

Let us now consider the incompressibility lemma of Kolmogorov complexity and its implications for the complexity of genotype strings. Informally, Kolmogorov complexity quantifies the complexity of all bitstrings.

For a given bitstring length, at least $\frac{1}{2}$ of bitstrings of that length cannot be compressed more than one bit. Furthermore, at least $\frac{3}{4}$ of bitstrings of that length cannot be compressed more than two bits, $\frac{7}{8}$ of bitstrings of that length cannot be compressed more than three bits and so on (see Theorem 1). The number of bitstrings with a given complexity value will be exponentially reduced with the value of complexity until finally only at most one single bitstring has the lowest complexity value.

From the incompressibility lemma we know that for a given bit string length, most of the bit strings are of high complexity. Assuming that interesting genotypes are of some regularity, and thus can be compressed, we can search in the space corresponding to the same length as the compressed genotypes. The size of this search space is reduced exponentially with the compressibility of the interesting genotypes. Considering a case where the interesting genotypes are of complexity $\frac{n}{2}$, one could potentially reduce the size of the search space with a factor of $2^{\frac{n}{2}}$.

3.3 Digital circuit evolution

The circuits evolved for the experiments herein are combinatorial 2-bit multipliers and adders. A Cartesian GP system was used [8]. The genotype is a symbolic netlist, as shown in one column in example in table 1. Each row represents a gate, with its type and from where to connect its two inputs. Connections can only be made to lower number gates (or the circuit inputs). This simple scheme assures a pure feed-forward network and thus combinatorial circuits. The last gates in the netlist are connected to the circuit output. The available gate types are NOT, AND, OR, $NAND$ and NOR as well as connections to GND and VCC.

Mutation is applied at gate level. A mutated gate either has one of its input connections rewired or its type is changed.

The goal of evolution is to generate a circuit that successfully produces the correct mapping between input and output vectors. The target behavior (multiplier or adder) is specified by a truth table. A candidate circuit is tested by applying the complete set of input vectors and comparing produced output to the target truth-table. The fitness of a candidate is equal to its hamming distance to the complete set of correct output vectors, with zero fitness being a fully correct circuit.

This approach, and many others, will relatively efficiently discover correct combinatorial circuits. However, a major issue arises as we increase genotype size in order to evolve larger circuits. As the genotype size is increased with more elements, so is the dimensionality of the search space, and the EA run time will usually increase in a seemingly exponential fashion. As such, larger circuits are therefore much harder to evolve [8]. This phenomena is dominant for the field of EHW in general, and is the main driving force of the earlier mentioned developmental approaches.

3.4 Encoding of circuits

Cartesian GP employs a symbolic netlist representation. In order to measure the LZ-complexity of genotypes we need an encoding scheme. Three encoding schemes were used. An example showing the encoding of a genotype using the different schemes is given in table 1.

3.4.1 Straight-forward encoding

The straight forward encoding scheme enumerate all the gate types using three bits. All connections are simply converted to binary numbers, using as few bits as possible (e.g. gate 7 can only have connections to lower number gates, and thus each connection can be encoded using 3 bits). Gates that do not utilize both inputs (GND and VCC have no inputs, NOT has one) are encoded with zeros for their unused connections. This preserves a constant genome length. Using this encoding schemes the circuits are encoded into a 504 bit string.

3.4.2 One-hot encoding

A closer look at the straight-forward scheme reveals a variable complexity of the enumerations of each circuit type. The one-hot encoding scheme remedies this by employing a brute force approach that enforces all gate types to be assigned binary codes of more similar complexity. The type is encoded using 8 bits, where only one single bit is set to one at the time (one-hot). With this exception, the encoding scheme is equal to the straight forward scheme. This scheme will encode the circuits into a string of 664 bits.

3.4.3 Unused-to-zeros encoding

In the symbolic representation many gates are present which are inactive in the corresponding phenotype. Only gates who's output is propagated to the output of the circuit are considered active. These inactive gates have no influence on circuit functionality, and while the active part of a multiplier circuits may be assumed to be of rather low complexity, the inactive parts may not be a part of the regular structure and as such may contribute to a higher measured complexity of the genotype. Circuits with few active gates (simpler structure) may thus have a genotype which appears rather complex. To get a picture of the complexity of only the active part of the genome, the unused-to-zeros encoding scheme encodes all inactive gates and their connections as zero. Using this scheme the circuits are encoded into a bitstring of equal length as when using straight-forward encoding, 504 bits.

4 Experiments

4.1 Measuring complexity

In the experiments described, we used the standard compression library `zlib` [15] as the Lempel-Ziv implementation. Two approaches to measurement was used. The first approach compresses the binary string, and the length of the compressed string is denoted as $LZ_b(x)$. The second approach encodes the binary string as ASCII-characters 0 or 1 and compresses this buffer, denoting the length of the compressed string as $LZ_a(x)$. The second approach is used since `zlib` has a hard time compressing short strings, and the use of ASCII characters may somewhat remedy this issue.

It should be noted that `zlib` produces some overhead data in the compressed string, and an incompressible string

		SYMBOLIC			STRAIGHT			UNUSED-ZERO				ONE-HOT		
what	**label**	**type**	**in A**	**in B**	**type**	**in A**	**in B**	**type**	**active**	**in A**	**in B**	**type**	**in A**	**in B**
input	0													
input	1													
input	2													
input	3													
gate	4	AND	3	1	011	11	011	011	y	11	01	0001000	11	011
gate	5	NOT	4		010	100	000	000	n	000	000	0010000	100	00000
gate	6	OR	0	2	101	000	010	101	y	000	010	0000010	000	010
gate	7	NOR	2	1	110	010	001	110	y	010	001	0000001	010	001
.														
.														
.														
gate	38	VCC			001	000000	000000	001	y	000000	000000	0100000	000000	000000
gate	39	GND			000	000000	000000	000	n	000000	000000	1000000	000000	000000
gate	40	NAND	30	22	100	011110	010110	000	n	000000	000000	0000100	011110	010110
gate + output	41	AND	15	27	011	001111	011011	011	y	001111	011011	0001000	001111	011011
gate + output	42	OR	37	34	101	100101	100010	101	y	100101	100010	0000010	100101	100010
gate + output	43	NAND	38	24	100	100110	011000	100	y	100110	011000	0000100	100110	011000

Table 1. Example genotype encoding of a 4-inputs, 3-outputs circuit

will compress to a longer string then the original. In addition the compressed string is output in bytes, thus the measurements will have a resolution of 8 bits.

4.2 Evolution of circuits

Using the described Cartesian GP system a set of 200 multipliers and 200 adders was evolved using the parameters shown in table 2. The evolved circuits were then encoding using the three different encoding schemes, and the Lempel-Ziv complexity of the encoded versions of all circuits was measured. This should give some picture as to how complex the encoded circuits are in general. In order to be able to compare these values to the complete search space, a large number (10k) of random valid circuits were generated. These circuits are not necessarily functionally correct multipliers or adders (probably none of them are), but the symbolic genotypic representation force them to be valid (no feedback, only allowed gates etc.). Again, we encoded the circuits and measured the Lempel-Ziv complexity. This should give a picture of the complexity distribution of the search space. For reference a large number (10k) of random strings were generated of length 504 and 664 bits, and their complexity was measured as well.

5 Results and Discussion

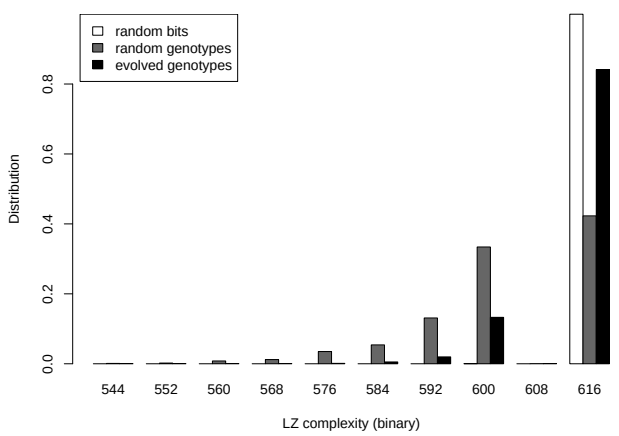
The measured complexity of evolved multipliers and random genotypes encoded using the straight-forward scheme as well as random bitstrings of length 504 is shown in figure 2. The bars show the distribution of complexity values for each group of measurements. It should be noted that the vertical axis is scaled differently on each plot in order to maximize visibility.

Parameter	**Setting**
Genotype size	40 gates
Phenotype target	2-bit ADD & MUL
Population size	20
Elitism (neutral)	1
Max generations	100k
Mutation rate	5% pr. gate
Crossover rate	20%, single-point
Selection method	tournament selection
Group size	3
Selection prob.	70%

Table 2. Cartesian GP settings

A first observation is that for random bitstrings all compressed strings are of length 616 using the binary-based complexity measure LZ_b. As we shall see later in figure 5, a single sample is of length 608. From the incompressibility lemma tells us that there are few bitstrings with lower complexity, and combined with an 8-bit resolution it is to be expected that very few random bitstrings have less than maximum complexity. The fact that the compressed strings are longer than the original (compressed from 504 bits to 616 bits) is due to the `zlib` overhead.

Moreover we can observe that both random and evolved genotypes have samples that are less complex than random bitstrings. This is to be expected, as genotypes are bound to have some regularity in their structure. Perhaps surprisingly, evolved genotypes seem more complex than random ones. A reason for the low observed complexity of random genotypes is probably the fact that $\frac{3}{7}$ of the gates on average will be of type GND, VCC and NOT. Those gate types encode empty connection fields as zeros, as mentioned in section 3.3, and are therefore less complex than evolved genotypes who's use of those gate types is probably less



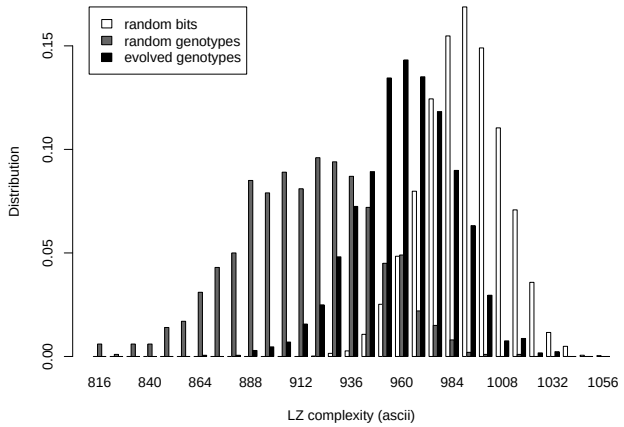
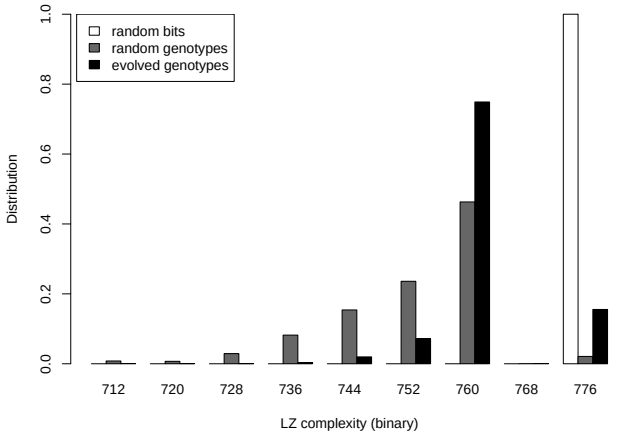


Figure 2. Multipliers using straight-forward encoding



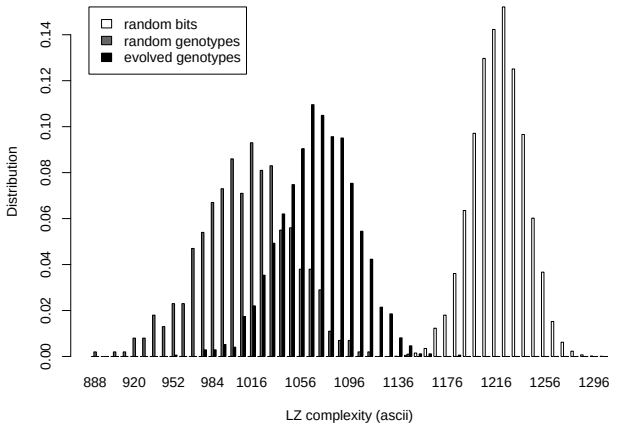


Figure 3. Multipliers using one-hot encoding

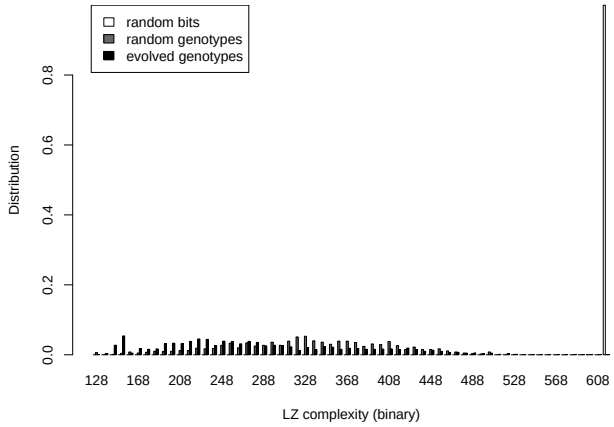
common.

The same measurements using the one-hot and unused-to-zero encoding schemes are shown in figures 3 and 4 respectively. The one-hot encoding seems to produce the same trend as the straight-forward scheme, but is slightly better at separating the three categories. Not that the random bitstrings shown in figure 3 shows random bitstrings of length 664, as these are the relevant ones in comparison with the one-hot encoding scheme (which encodes genotypes into bitstrings of length 664).

The measurements using the unused-to-zero encoding scheme shown in figure 4 are quite different. In the evolved circuits, less than half of the gates are typically active. It is therefore not surprising to see the low complexity values when the inactive parts of the genome are encoded as zero before measuring the LZ-complexity. The random genomes have very few active elements, and are of little relevance but shown for reference.

If we compare ASCII and binary complexity measures, we see that the complexity values are higher when using ASCII. The bitstring to be compressed using the ASCII approach is 8 times longer, and seem to compress to 2-3 times the size as would be the case with the binary compression approach. Also, the ASCII approach seem to yield a "smoother" visualization. Other than this, the relative characteristics of the measurements are the same when comparing ASCII and binary complexity measures.

To get a clearer picture of the relationship between the results box-plots were generated and are shown in figures 5, 7, 6 and 8. A box shows the interquartile range of the measured values, with a line showing the median, whiskers showing extreme values and points showing the most extreme values individually. The labels along the horizontal axis represents, in the following order; random bitstrings of



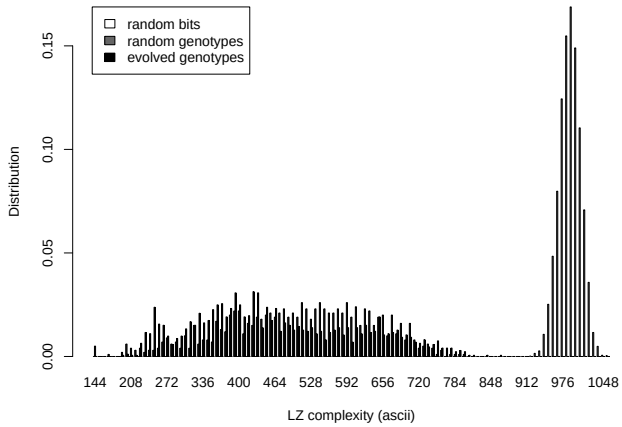


Figure 4. Multipliers using unused-to-zero encoding

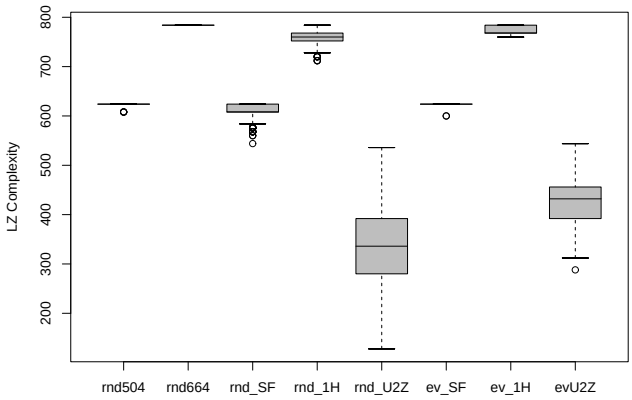


Figure 5. Box-plot multipliers (binary compression)

length 504 and 664 respectively, random genotypes encoded using the straight-forward, one-hot and unused-to-zero encoding and evolved genotypes using the same three encoding schemes.

Figures 5 and 6 shows the box-plot for multipliers using the binary complexity measure and ASCII complexity measure respectively. The box-plots summarizes what we know from figures 5, 3 and 4, and gives a clearer picture of the relationship between the different measurements.

So far we have only considered the case of 2-bit multipliers. The box-plots for the case of 2-bit adders are shown in figures 7 and 8. The differences between the complexities measured for multipliers and adders are marginal.

The main question is whether the genotypes evolved to be multipiers or adders can be compressed. Looking at the results, the answer depends on several choices, such as encoding method, measurement method and how we choose to interpret the experimental data. Perhaps counterintuitive, evolved genotypes can be measured as more complex than random genotypes. But, as mentioned, the purely practical reason for this may be the high amount of low-complexity gate types (VCC, GND and NOT) in random genotypes. If, however, we compare the complexity of evolved genotypes encoded using the unused-to-zero approach to that of random genotypes using straight-forward encoding, we see the strong opposite where evolved genotypes are much less complex than random ones. This case may be exploitable, as the standard EA searches for the former evolved genotypes in the latter space of random genotypes. For instance, we know that the majority of evolved adders are in the complexity range 400-450 if we consider only the active part of the genome (unused-to-zero) and measure complexity using binary `zlib` compression. On the other hand, the distribution of complexities complexity over the search space considered by looking at the complexity of most random genotypes, is in the range 590-620. In terms of pure search space size, we could potentially search in a reduced space of 2^{450} instead of a search space of size 2^{620} (including the `zlib` overhead) by employing reverse Lempel-Ziv as a development mapping. The practical feasability of actually doing this does ofcourse require substansial investigation.

6 Conclusion and Future Work

Development mapping and how it relates to genotypic complexity is a difficult issue. We have approached the issue by performing actual measurements of genotypic complexity using Lempel-Ziv compression. This approach

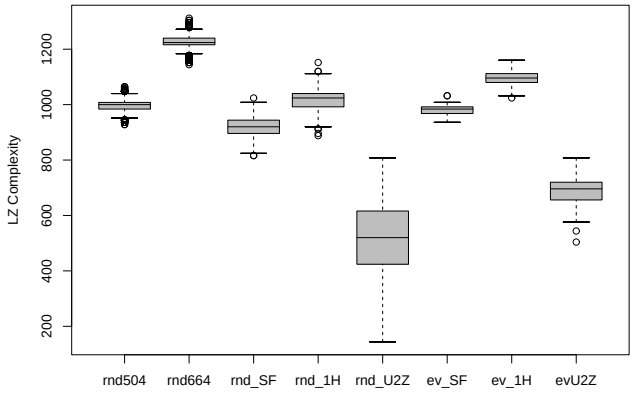


Figure 6. Box-plot multipliers (ascii compression)

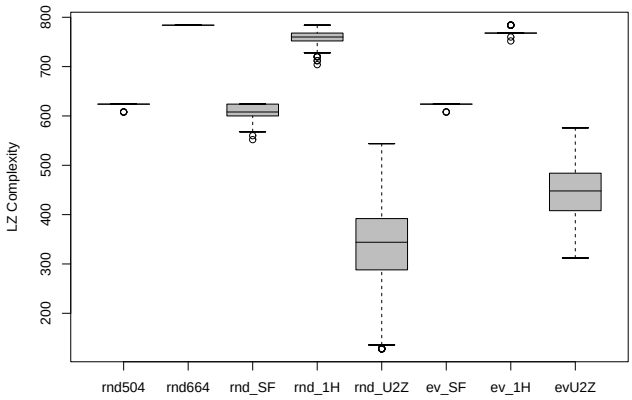


Figure 7. Box-plot adders (binary compression)

forces one to make practical choices that may affect the result, and is also limited by the complexity measure itself. On the other hand, there is a benefit to the practical approach. For instance, the use of Lempel-Ziv as a complexity measure means that the compressed version of the genotype can actually be used in practice since a Lempel-Ziv decompression could act as a development mapping.

The results show that if one concider only the active part of a genotype, the complexity distribution of evolved genotypes is much less than that of all genotypes (the search space). This could potentially be exploited to improve the scalability of the EA. It should be noted that we have considered only the complexity and size domains of the search space. For evolution to benefit from a search space size reduction, we need to investigate other important features such as search space structure, neutrality and genotype-phenotype mapping and distance correlation.

7 Acknowledgments

Thanks for valuable thoughts to all participants at the seminar at Fefor, Norway, arranged by Keith Downing of the Complex Adaptive Organically-Inspired Systems Group (CAOS).

References

[1] P. J. Bentley. Evolving fractal proteins. In A. M. Tyrell, P. C. Haddow, and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Fifth Int. Conf., ICES 2003*, volume 2606 of *Lecture Notes in Computer Science*, pages 81–92. Springer, 2003.

[2] R. Cilibrasi and P. Vitanyi. Clustering by compression. *IEEE Trans. Information Theory*, 51(4), 2005. To appear.

[3] P. Eggenberger. Evolving morphologies of simulated 3d organisms based on differential gene expression. In P. Husbands and I. Harvey, editors, *Proceedings of the 4th European Conference on Artificial Life (ECAL97)*. Cambridge: MIT Press, 1997.

[4] T. Gordon and P. Bentley. Towards development in evolvable hardware. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, *The 2002 NASA/DoD Conference on Evolvable Hardware*, Alexandria, Virginia, 15-18 July 2002. Jet Propulsion Laboratory, California Institute of Technology, IEEE Computer Society.

[5] F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis, France, 1994.

[6] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Stanford University, CA, USA, 1996. MIT Press.

[7] P. C. Haddow, G. Tufte, and P. van Remortel. Shrinking the genotype: L-systems for ehw? In Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, and M. Yasunaga, editors, *Evolvable Systems: From Biology to Hardware. Proceedings of the 4th International Conference on Evolvable Systems, ICES'2001, Tokyo, October 3-5, 2001*, volume 2210 of *Lecture Notes in Computer Science*, pages 112–127. Springer-Verlag, Berlin, Heidelberg, 2001.

[8] M. Hartmann and P. C. Haddow. Evolution of fault-tolerant and noise-robust digital designs. *IEE Proc. -Comput. Digit. Tech.*, 151(4):287–294, 2004.

[9] H. Kitano. Designing neural networks using genetic algorithm with graph generation system. *Complex Systems*, 4:461–476, 1990.

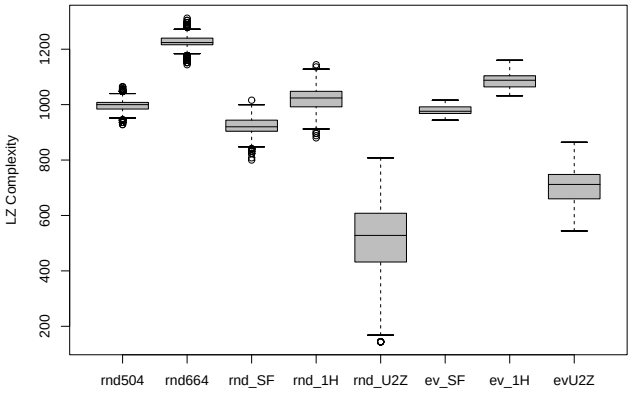


Figure 8. Box-plot adders (ascii compression)

[10] S. Kumar and P. J. Bentley. Biologically inspired evolutionary development. In A. M. Tyrell, P. C. Haddow, and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Fifth Int. Conf., ICES 2003*, volume 2606 of *Lecture Notes in Computer Science*, pages 57–68. Springer, 2003.

[11] P. K. Lehre and P. C. Haddow. Developmental mappings and phenotypic complexity. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 62–68. IEEE Press, 2003.

[12] P. K. Lehre and M. Hartmann. Complexity-based fitness function modifiers. In *Proc. of Workshop on Learning and Regeneration in Developmental Systems (part of GECCO2004)*, 2004.

[13] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, New York, 2nd edition, 1997.

[14] A. Lindenmayer. Developmental Systems without Cellular Interactions, their Languages and Grammars. *Journal of Theoretical Biology*, 1971.

[15] J. loup Gailly and M. Adler. *zlib general purpose compression library version 1.1.4*, March 2002.

[16] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the evolutionary design of digital circuits – part i. *Journal of Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[17] J. F. Miller and P. Thomson. A developmental methos for growing graphs and circuits. In A. M. Tyrell, P. C. Haddow, and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Fifth Int. Conf., ICES 2003*, volume 2606 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2003.

[18] A. Siddiqi and S. Lucas. A comparison of matrix rewriting versus direct encoding for evolving neural networks. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 392–397. IEEE Press, 1998.

[19] K. O. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

[20] G. Tempesti, D. Mange, E. Petraglio, A. Stauffer, and Y. Thoma. Developmental processes in silicon: An engineering perspective. In J. Lohn, R. Zebulum, J. Steincamp, D. Keymeulen, A. Stoica, and M. I. Ferguson, editors, *Proc. The 2003 NASA/DoD Conference on Evolvable Hardware*, pages 255–264. IEEE Computer Society, 2003.

[21] G. Tufte and P. Haddow. Building knowledge into developmental rules for circuit design. In A. M. Tyrell, P. C. Haddow, and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Fifth Int. Conf., ICES 2003*, volume 2606 of *Lecture Notes in Computer Science*, pages 69–80. Springer, 2003.

[22] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–342, 1977.