Available online at www.sciencedirect.com
ScienceDirect
BioSystems 87 (2007) 224–232
www.elsevier.com/locate/biosystems

The genotypic complexity of evolved fault-tolerant and noise-robust circuits

Morten Hartmann*, Pauline C. Haddow, Per Kristian Lehre

Complex Adaptive Organically-Inspired Systems Group (CAOS), Department of Computer Science, The Norwegian University of Science and Technology (NTNU), Norway

Received 28 February 2005; received in revised form 8 July 2006; accepted 15 July 2006

Abstract

Noise and component failure is an increasingly difficult problem in modern electronic design. Bio-inspired techniques is one approach that is applied in an effort to solve such issues, motivated by the strong robustness and adaptivity often observed in nature. Circuits investigated herein are designed to be tolerant to faults or robust to noise, using an evolutionary algorithm. A major challenge is to improve the scalability of the approach. Earlier results have indicated that the evolved circuits may be suited for the application of artificial development, an approach to indirect mapping from genotype to phenotype that may improve scalability. Those observations were based on the genotypic complexity of evolved circuits. Herein, we measure the genotypic complexity of circuits evolved for tolerance to faults or noise, in order to uncover how that tolerance affects the complexity of the circuits. The complexity is analysed and discussed with regards to how it relates to the potential benefits to the evolutionary process of introducing an indirect genotype–phenotype mapping such as artificial development.
© 2006 Elsevier Ireland Ltd. All rights reserved.

Keywords: Complexity; Fault-tolerance; EHW; Noise; Robust; Evolution

1. Introduction

The need for fault and noise tolerance is an important issue in modern electronic design. High density chips increase the possibility of failing components and the complexity of design increases the probability of human errors. Fault acceptance diminishes as the market demands increasingly more reliable systems. The need for fault-tolerant designs and management of noise are stated amongst the long term (2008–2016) grand challenges by the International Technology Roadmap for Semiconductors (ITRS, 2003).

Recently, a growing research field has started exploring new solutions to these problems, the field of evolvable hardware (EHW). The field seeks to apply evolution and other bio-inspired methods to hardware design exploration and optimization. Evolutionary Algorithms (EAs) is one such method, and was used by Hartmann and Haddow (2004) to generate circuits that to some extent are tolerant to faults and robust in handling noise. The approach suffers, like many EAs often do, from the fact that it does not scale well with increased problem size. Increasing the number of elements in the genotype increases the dimensionality of the search space, usually resulting in a problem which is harder to solve.

One approach to reducing the search space and thus potentially improving the scalability of EAs is artificial development (AD) (Stanley and Miikkulainen, 2003). Inspired by biological development and the fact that the human genome is exceedingly small compared to features of a grown human being such as the brain, AD could allow EAs to operate using smaller genotypes.

* Corresponding author.
E-mail addresses: mortehar@idi.ntnu.no (M. Hartmann), pauline@idi.ntnu.no (P.C. Haddow), lehre@idi.ntnu.no (P.K. Lehre).

0303-2647/$ – see front matter © 2006 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.biosystems.2006.09.017

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 relationship between complexity and the genotype–phenotype mapping is one of the key issues. Work by Lehre and Hartmann (2004) shows that Kitano mapping, a type of developmental mapping, achieves phenotypes that are much more compressible than random strings of the same length. Thus, using an indirect mapping such as Kitano mapping the search space is restricted to a region of specific complexities below the complexity of random strings of the same length. This property could be exploited. If one seeks to design phenotypes that reside in a region of complexities covered by the indirect mapping one could search in this smaller region. A reduced search space can yield a more efficient evolutionary search. If so, this could increase the size and complexity of circuits that can be designed using evolutionary approaches.

Hartmann et al. (2005) measured the complexity of evolved multipliers and adders using Lempel–Ziv compression (Ziv and Lempel, 1977). The complexity of these circuits was found to be in a significantly smaller region of complexities than that covered by the evolutionary search. To be able to exploit this, an indirect mapping must be able to achieve the relevant phenotypic complexity and phenotypes that perform correctly according to the fitness function. The advantage of using Lempel–Ziv as a complexity measure is that, in theory, Lempel–Ziv decompression could act as an indirect mapping. By measuring complexity of evolved circuits using compression, one knows that there exists at least one type of indirect mapping that can produce, not only the relevant phenotype complexities, but also phenotypes that perform correctly according to the fitness function.

Herein, the investigations are continued to include circuits evolved to be tolerant to faults or noise. The importance of handling faults and noise motivates further research with the goal of improving the scalability of evolutionary approaches applied to the automatic generation of circuits that can cope with faults and noise.

In the field of Bio-inspired hardware there are already several important contributions related to fault-tolerance. For instance, fault-tolerance and error detection for robots is explored by Jackson et al. (2003) using embryonic arrays. A decade ago, Thompson (1995) evolved fault-tolerant electronic control systems. Keymeulen et al. (2000) evolved field-programmable transistor arrays (FPTA) to obtain fault-tolerance. A developmental model for digital systems with fault-tolerant properties has been developed by Liu et al. (2004).

The reminder of this paper starts of with Section 2 giving an explanation of our interpretation of complexity and how it relates to the potential benefits to the evolutionary process of introducing an indirect genotype–phenotype mapping such as AD. Background on the simulation of faults and noise in evolved circuits is given in Section 3, while details on how the circuits are evolved are given in Section 4. Section 5 describes the encoding of circuits, how complexity is measured and the experiments executed. The results are shown and discussed in Section 6 and conclusions are given in Section 7.

2. Complexity

The complexity measure used herein is theoretically based on Kolmogorov complexity. Kolmogorov complexity is a quantitative measure of complexity based on Turing machines (Li and Vitányi, 1997).

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

$$C(x) = \min_{p \in \mathcal{N}} \{\ell(p) | \phi(p) = x\} \quad (1)$$

where ϕ is the partial recursive function corresponding to a fixed universal Turing machine, and $\ell(x)$ denotes the length of a number (in terms of bits).

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. Herein Lempel–Ziv compression (Ziv and Lempel, 1977) is used as the practical measure of complexity and is referred to as LZ-complexity. This yields an upper bound approximation of the Kolmogorov complexity. As a string compressed with LZ can also be decompressed, the LZ-complexity gives a correct measure of how small a genotype can be relative to the phenotype using an example indirect mapping, i.e. LZ decompression.

2.1. Incompressibility lemma

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

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

On the other hand no compression algorithm can compress every bitstring, so certain bitstrings must be incompressible. As we will see, most bitstrings are in-

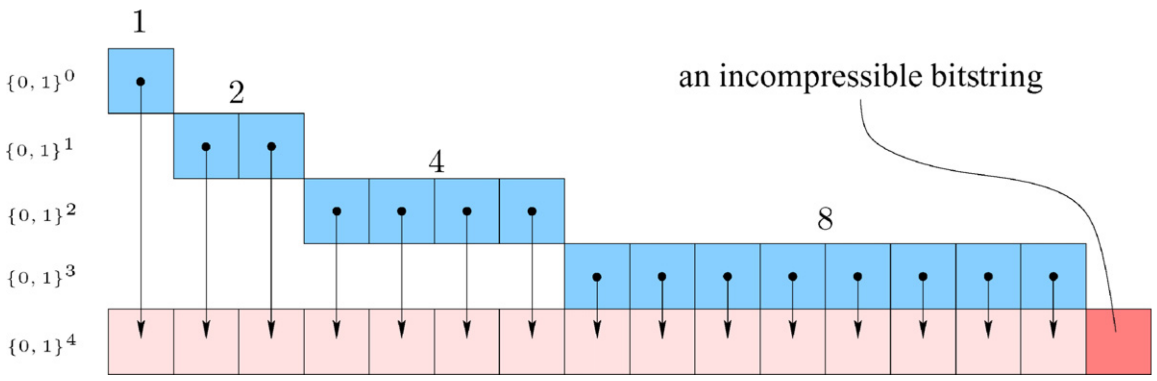

Fig. 1. The incompressibility lemma.

compressible 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 Fig. 1. Among bitstrings of length 4, maximum 1 bitstring has complexity less than 1, maximum 3 bitstrings have complexity less than 2, maximum 7 bitstrings have complexity less than 3 and maximum 16 bitstrings have complexity less than 4. At least one bitstring has the maximum complexity of at least 4. The consequence is that the general compressibility of the bitstrings is very low. This argument can be generalized for sets of bitstrings of any length n.

Theorem 1. (The incompressibility lemma (Li and Vitányi, 1997)). *Let c be a positive integer. Every finite set A of cardinality m has at least* $m(1-(1/2^c))+1$ *elements x with* $C(x) \geq \log m - c$.

2.2. 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 $(1/2)$ of bitstrings of that length cannot be compressed more than one bit. Furthermore, at least $(3/4)$ of bitstrings of that length cannot be compressed more than two bits, $(7/8)$ 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. The description of a circuit is bound to contain some regularity as gate types and connections are repeated several times in a circuit. This regularity means the description of a circuit can be compressed and one could potentially 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 genotypes that correspond to the phenotypes we search for. Considering a case where the interesting genotypes are of complexity $(n/2)$, one could potentially reduce the size of the search space with a factor of $2^{n/2}$.

3. Simulation

The feed-forward simulator (Hartmann and Haddow, 2004) designed allows simple, yet for our purposes, sufficiently realistic modelling of a digital circuit with the support for analogue noise and signal values. The simulator is combinatorial and there is no concept of time. The advantage of this approach is that many experiments can be run without requiring huge amounts of computational resources, which would be the case with a state of the art simulator, e.g. SPICE. Additionally, it allows tuning to different technologies.

A sigmoid approximation to 5 V CMOS technology is the core of the gate model. The output of a gate is calculated as a function of the sigmoid approximation. This function corresponds to F in the gate model shown in Fig. 2. E_1, E_2 and E_3 can generate one of the supported errors or let the signal propagate through without error. The probability of error is preset for each experiment,

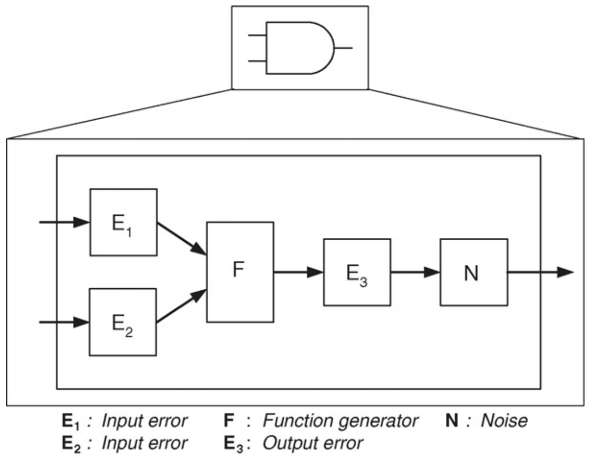

Fig. 2. Generic two-input gate. E_1: input error; F: function generator; N: noise; E_2: input error; E_3: output error.

while the type of error is random with equal probability for each of the three possible faults. The possible faults are of type stuck-at errors, floating outputs and partly random output. In addition, there is support for inducing signal noise at each gate output. Input or output stuck-at errors cover the case of short-circuit to power or ground, or the cases of inter-signal short-circuits that behave as stuck-at errors when observed in a combinatorial and timeless domain. Floating output errors cover cases where the output is completely random, while partly random output covers the case where the output is correct for one logical value while random for the other logical value, e.g. logical 1 is represented as 1 while logical 0 is represented as a random number from 0 to 1.

The output noise N in Fig. 1 is noise that is superimposed on the output signal to approximate errors that are not explicitly a part of the model, e.g. thermal noise, radiation, power supply noise, component variance and cross talk. Noise has no time-related effects but is simply implemented as random numbers within a specified range (in percent of the complete signal range).

4. Digital circuit evolution

The circuits evolved for the experiments herein are combinatorial multipliers and adders. A Cartesian genetic programming (GP) system was used, adopted from that of Miller et al. (2000). The genotype is a symbolic netlist as shown in one column in the example in Table 2. Each row represents a gate, with its type and from where to connect its two inputs. Gate input connections can only be made to lower numbered 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 the 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 output vectors for all input 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. When evolving circuits under fault or noise conditions, a circuit is subject to several noise or fault vectors. For each such vector the circuit is tested using the complete set of possible input vectors. The analogue output values of the circuit are rounded to their closest logical value and then compared to the target truth table.

The fitness function is expressed in Eq. (2). A circuit C (an individual) is tested against the target truth table (T) a number of times (TPI) under different environments. Noise and fault probabilities are used to generate the different environments m for each test. The average of all tests is computed to yield a penalty for the number of incorrect output bits. Thus, the fitness is effectively the negative of the average hamming distance between the measured function of the evolved circuit and the target function truth-table:

$$F = -\frac{\sum_{n=1}^{\mathrm{TPI}} \mathrm{diff}(C_m, T)}{\mathrm{TPI}} \tag{2}$$

where F is the fitness of individual, TPI the test per individual, diff(·) the number of incorrect output bits, C_m the circuit in environment m, and T is the target truth table.

5. Experiments

The following part describes the details of the experiments. First, a set of circuits were evolved as described in Section 5.1 and reported by Hartmann and Haddow (2004). Secondly, the evolved circuits were converted to a binary format as described in Section 5.2. Finally, the LZ-complexity of the circuits were measured as described in Section 5.3.

5.1. Evolution of circuits

Using the described Cartesian GP system a set of two-bit multipliers and adders was evolved with 20 circuits of each type for each value of fault probability or amount of noise. The GP parameters are shown in Table 1. Neutral elitism is used, allowing the one best individual to take part in the next generation (randomly chosen if several are equally good). The range of fault values was from 0% up to 14% chance of each gate failing (with steps of 1%), while the range of noise applied was from 0% up to 70% (with steps of 5%). The result was 4800

Table 1
Cartesian GP settings

Parameter	Setting
Genotype size	40 gates
Phenotype target	2-bit ADD and MUL
Population size	20
Elitism (neutral)	1
Max generations	100,000
Mutation rate	5% pr. gate
Crossover rate	20%, single-point
Selection method	Tournament selection
Group size	3
Selection prob.	70%

circuits evolved with simulations of different amounts of faults or noise.

Further details on the evolved circuits with regards to evolvability, degree of fault-tolerance and robustness to noise can be found in earlier work reported by Hartmann and Haddow (2004).

5.2. Encoding of circuits

The employed Cartesian GP system uses a symbolic netlist representation. In order to measure the LZ-complexity of genotypes we need an encoding scheme. An example showing the encoding of a genotype is given in Table 2.

This encoding scheme enumerates 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 of that gate's input connections can be encoded using 3 bits). Gates that do not utilise both inputs (GND and VCC have no inputs, NOT has one) are encoded with zeros for their unused connections. See for instance gate 38, a connection to VCC, in Table 2. In addition, the gates with output that does not propagate to the circuit output do not take part in the phenotypic function and are coded as all zeros. An example of such a gate is gate 40 in Table 2. This scheme preserves a constant genome length of 504 bits.

This encoding scheme was termed unused-to-zero in earlier work by Hartmann et al. (2005). All inactive parts of the genome are coded as zero. The result is that the measured LZ-complexity reflects the complexity of only the phenotypically active part of the genome.

5.3. Measuring complexity

In the experiments described, we used the standard compression library `zlib` (Gailly and Adler, 2002) as the Lempel–Ziv implementation. The complexity measure is obtained by compression of the binary encoded genotype. The complexity of the genotype is equal to the length of the compressed string and is denoted as LZ(x).

It should be noted that `zlib` produces some overhead data in the compressed string and an incompressible string will compress to a longer string than the original. In addition, the compressed string is output in bytes, thus the measurements will have a resolution of 8 bits.

Since the compression is reversible, one could in theory search in bitstrings equal to the length of a known compressed evolved multiplier or adder, employing an indirect genotype–phenotype mapping, e.g. LZ-decompression. This would reduce the size of the search space, but with unknown effects on evolvability and search space topology. By measuring the complexity of several evolved multipliers or adders, one can to some extent uncover the possibilities of such a compression of the search space. If most multipliers can be compressed from a length n to less than a length m, the search space can in theory be reduced from 2^n to 2^m, while still maintaining the possibility of discovering at least that same set of circuits.

Table 2
Example genotype encoding of a four-inputs, three-outputs circuit

What	Label	Symbolic			Binary			
		Type	In A	In B	Type	Active	In A	In B
Input	0							
Input	1							
Input	2							
Input	3							
Gate	4	AND	3	1	011	y	11	01
Gate	5	NOT	4		000	n	000	000
Gate	6	OR	0	2	101	y	000	010
Gate	7	NOR	2	1	110	y	010	001
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gate	38	VCC			001	y	000000	000000
Gate	39	GND			000	n	000000	000000
Gate	40	NAND	30	22	000	n	000000	000000
Gate + output	41	AND	15	27	011	y	001111	011011
Gate + output	42	OR	37	34	101	y	100101	100010
Gate + output	43	NAND	38	24	100	y	100110	011000

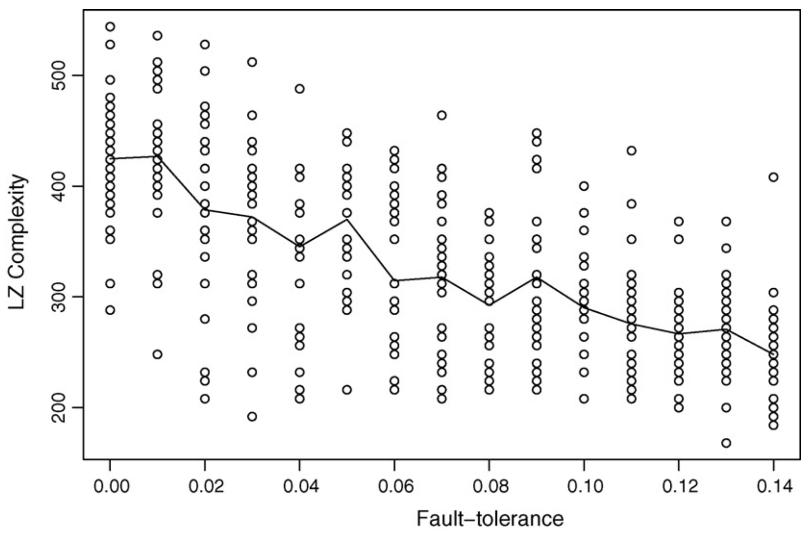

Fig. 3. Complexity of evolved multipliers.

6. Results and discussion

In the case of circuits evolved for fault-tolerance, the measured complexity is shown in Figs. 3 and 4, respectively. The graphs show the average complexity drawn as a solid black line, while the scattered circles are the individual measurements. Since some circuits may have the same LZ-complexity points may overlap. This can often be the case since the resolution of the LZ-complexity measure is limited to 8 bits. By chance, some circuits may even be functionally the same or, less likely, even have the same genotype.

From Figs. 3 and 4 we can clearly see that the evolved circuits, both multipliers and adders, are less complex the higher fault-tolerance they were evolved for. In fact, the multipliers evolved in an environment where 14% of the gates fail is, on average, of almost half the LZ-complexity of that of the standard multipliers (0% faults). For adders, the relative difference in the LZ-complexity between standard circuits and fault-tolerant ones does not seem as large as for multipliers. Comparing multipliers and adders we observe that adders are in general slightly more complex than multipliers, both for standard circuits and fault-tolerant ones. Before further dis-

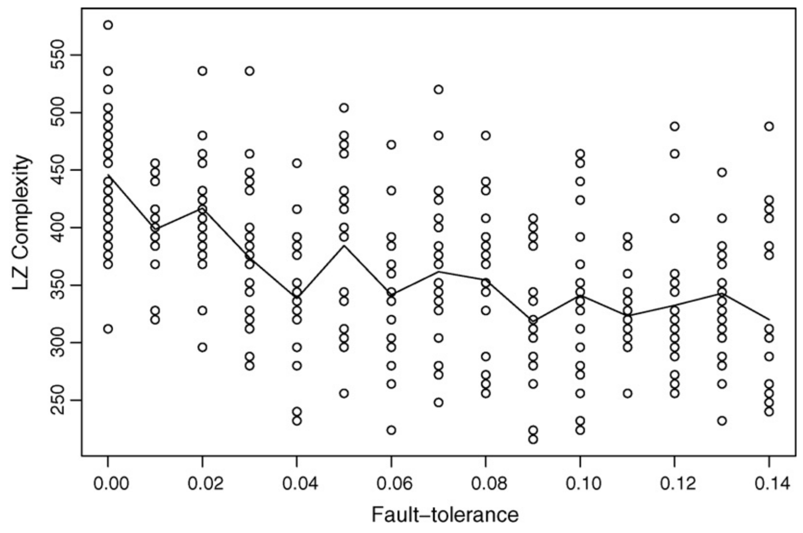

Fig. 4. Complexity of evolved adders.

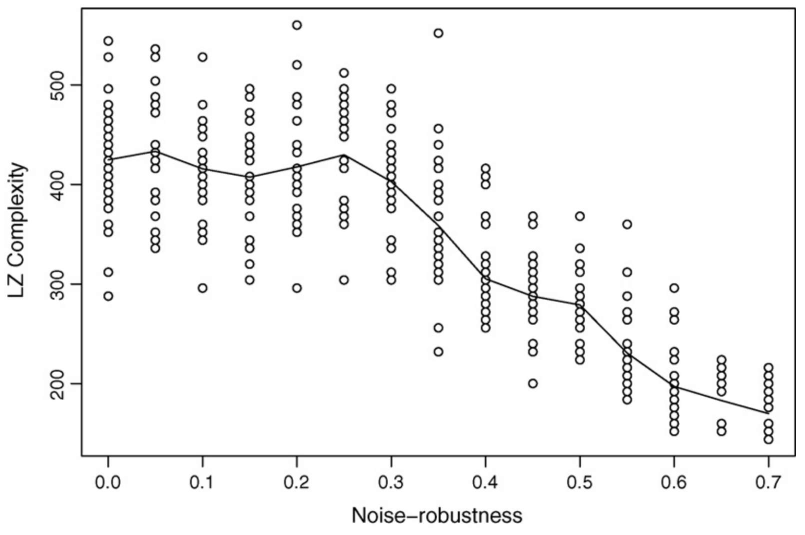

Fig. 5. Complexity of evolved multipliers.

cussions on this phenomena, let us have a look at the individual measurements themselves.

From the scattered circles we can see that there are relatively large differences in LZ-complexity even for circuits evolved under similar conditions, i.e. when the probability of faults or the amount of noise is the same. For instance, comparing extreme cases, some circuits evolved with requirements to fault-tolerance are more complex than certain circuits evolved without such requirements. The major trend is, however, the opposite. It does seem the highest complexity values are reserved for circuits evolved without requirements to tolerance. Conversely, none of the circuits without tolerance requirements are of very low complexity.

Similar graphs showing the case of increasing amounts of noise during simulation are shown in Figs. 5 and 6 for multipliers and adders, respectively. As in the case with fault-tolerant circuits, it is quite evident from the graphs that complexity decreases as the circuits are evolved in increasingly noisy environments. Multipliers evolved in a 70%-noise environment are on average of less than half the LZ-complexity of that of the standard circuits. Again, the case of adders is less extreme, but nevertheless show the same trend.

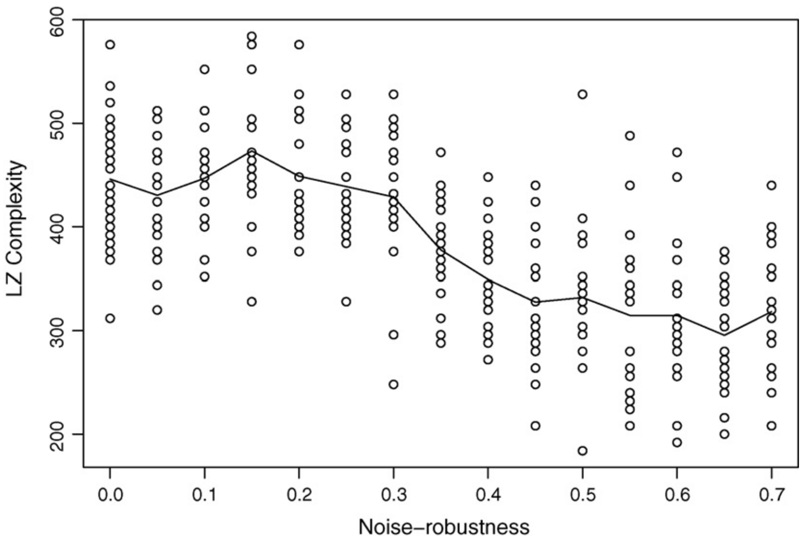

Fig. 6. Complexity of evolved adders.

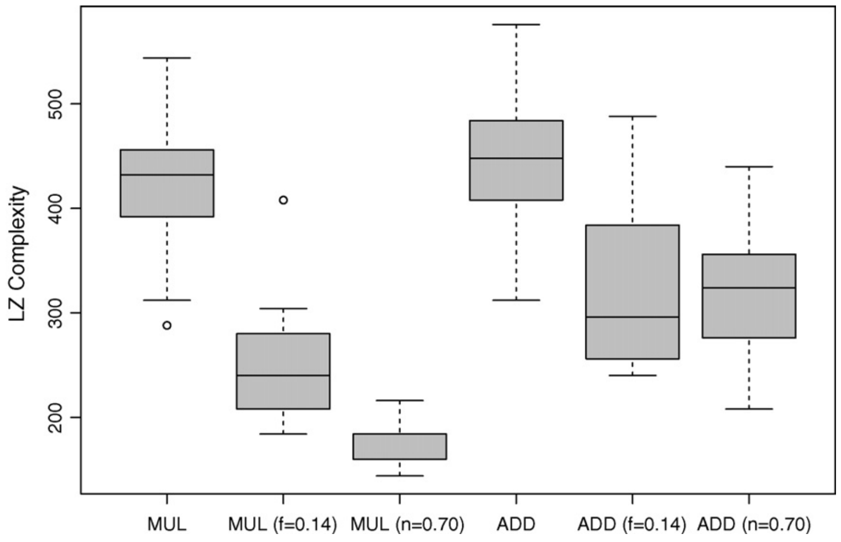

Fig. 7. Boxplot of multipliers and adders.

The results are shown in a more compact format in Fig. 7, showing box-plots for both multipliers and adders. A box shows the interquartile range of the measured values, with a line showing the median, whiskers showing extreme values and points showing outlying measurements individually. Boxes are shown for evolved multipliers and adders without special requirements and circuits performing the same functions in simulations of either faults (14% probability) or noise (70% noise).

The box-plots confirm the earlier observations in that the circuits with requirements on fault-tolerance or noise-robustness are of less LZ-complexity than the other circuits. The box-plot also shows that this trend is stronger with multipliers than adders. The reason for the latter is currently unknown and is probably a result of the difference in architectures evolved for multipliers and adders.

Earlier investigations of the same circuits, reported by Hartmann and Haddow (2004), show how the number of active gates in the evolved circuits decreases with an increase in the amount of faults and/or noise the circuits have to handle. It is assumed that this decrease in active gates is one way that evolution copes with faults and noise in the circuits. Since faults occur in each gate with a given probability and noise is applied at every gate output, reducing the number of gates also reduces the total amount of "uncertainty" in the circuit. We regard this phenomena as the major reason why the LZ-complexity of the evolved circuits decreases when more faults or noise is introduced.

The motivation for the experiments is the possibility of shrinking the search space using indirect genotype–phenotype mappings. Looking at the case of multipliers evolved without special requirements, the box-plot in Fig. 7 tells us that the average such multiplier is of LZ-complexity $\mathrm{LZ} = 430$. In contrast, the average multiplier evolved in with simulations of 70% noise is of LZ-complexity $\mathrm{LZ} = 180$. In fact, all multipliers evolved with that much noise are below complexity $\mathrm{LZ}(x) < 220$. Consequently, one could potentially search in the space of genotypes of length 220, employing an indirect mapping, e.g. LZ decompression. This yields a potential search space reduction factor of $2^{(430-220)}$ with regards to circuits evolved without simulated noise, and $2^{(620-220)}$ compared to direct mapping ($\mathrm{LZ} = 620$ is roughly the upper limit of the genotype LZ-complexity as reported by Hartmann et al. (2005)). The practical feasibility of actually doing this does of course require substantial investigation. Notably, this argument only considers the complexity domain involved in indirect mapping. Important features such as search space structure, neutrality, mapping schemes and distance correlation need to be considered as well.

7. Conclusion and future work

The question as to when and if developmental mappings can be applied to the benefit of an evolutionary system is to a large degree unanswered in the field in general. We believe complexity to be a major factor in answering such a question. Herein, we have investigated the complexity domain of evolved digital circuits, in particular circuits that are evolved with simulated faults or noise.

The experiments are based on an algorithmic complexity measure, namely the length of a string when compressed using Lempel–Ziv compression. Using this measure, it is clear that circuits evolved under conditions with many faults or much noise are of less complexity than those evolved without those conditions. The ordinary circuits have in earlier work already been found to reside in a much smaller space of complexities than the full space searched by an EA using a direct mapping.

In terms of artificial development, the measured LZ-complexities are promising for the evolved digital circuits and the tolerant and robust ones in particular, in that with regards to complexity the circuits are suited for indirect mappings. As such, there is potential of improvements in scalability, since employing an indirect mapping may narrow the search down to a particular space of relevant complexity.

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.

References

Gailly, J., Adler, M., 2002. zlib general purpose compression library version 1.1.4.

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

Hartmann, M., Lehre, P.K., Haddow, P.C., 2005. Evolved digital circuits and genome complexity. Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware, IEEE Computer Society, pp. 79–86.

ITRS, 2003. International technology roadmap for semiconductors, executive summary. http://public.itrs.net.

Jackson, A.H., Canham, R., Tyrrell, A.M., 2003. Robot fault-tolerance using an embryonic array. In: Lohn, J., Zebulum, R., Steincamp, J., Keymeulen, D., Stoica, A., Ferguson, M.I. (Eds.), Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware, IEEE Computer Society, pp. 91–100.

Keymeulen, D., Zebulum, R.S., Jin, Y., Stoica, A., 2000. Fault-tolerant evolvable hardware using field-programmable transistor arrays. IEEE Trans. Reliabil. 49 (3), 305–316.

Lehre, P.K., Hartmann, M., 2004. Complexity-based fitness function modifiers. Proceedings of the Workshop on Regeneration and Learning in Developmental Systems, GECCO'04.

Li, M., Vitányi, P.M.B., 1997. An Introduction to Kolmogorov Complexity and its Applications, 2nd ed. Springer-Verlag, New York.

Liu, H., Miller, J.F., Tyrell, A.M., 2004. An intrinsic robust transient fault-tolerant development model for digital systems. Proceedings of the Workshop on Regeneration and Learning in Developmental Systems, GECCO'04.

Miller, J.F., Job, D., Vassilev, V.K., 2000. Principles in the evolutionary design of digital circuits—part i. J. Genet. Program. Evolvable Mach. 1 (1), 8–35.

Stanley, K.O., Miikkulainen, R., 2003. A taxonomy for artificial embryogeny. Artific. Life 9 (2), 93–130.

Thompson, A., 1995. Evolving fault tolerant systems. Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95), IEE Conf. Publication No. 414, pp. 524–529.

Ziv, J., Lempel, A., 1977. A universal algorithm for sequential data compression. IEEE Trans. Inform. Theory 23, 337–342.