Development and complexity-based fitness function modifiers

Per Kristian Lehre and Morten Hartmann

The Norwegian University of Science and Technology
Department of Computer and Information Science
7491 Trondheim, Norway
{lehre, mortehar}@idi.ntnu.no

Abstract. Artificial Development is a promising approach to evolutionary design optimization inspired by biological development. However, there is still no consensus as to which problem classes this approach has a clear advantage over classical direct encodings. We attack this problem by introducing the concept of fitness function modifiers based on complexity. Our results indicate that using these modifiers, we are able to discriminate a developmental mapping from a direct encoding with respect to their efficiency at solving classes of problems defined by the fitness modifiers.

1 Introduction

Evolutionary computation (EC) is a popular field of research with many applications. Evolutionary algorithms often employ relatively direct genotype-phenotype mappings. Recently, there has been increased interest in employing mappings inspired by biological development [1–11]. Biological development is here considered as the growth process from the fertilized egg, to a grown-up multicellular organism. We will refer to this subfield of EC as Artificial Development (AD) but terms like Artificial Embryogeny or Computational Development also occur.

Reducing the genome size with developmental mappings was considered as a possible approach to the scalability problem in complex evolutionary design optimizing tasks [12, 5, 1]. However, this issue turned out to be more complex than expected [3]. Several comparative studies between direct encodings and developmental mappings indicated that direct encodings were at least as efficient as developmental mappings for the problems studied [13, 7, 14]. There is no consensus in the field as to when developmental mappings should be applied. We therefor believe the following question to be important for further progress in AD: *For which classes of problems is it beneficial to apply developmental mappings over more traditional direct encodings?*

If we are unable to answer this question it might remain impractical to systematically utilize the beneficial properties of artificial development. As such, we believe that even partial answers to this question would be interesting. We propose the following approach to partly answer the question above: Given an

evolutionary algorithm and a fitness function, try to find ways to modify the fitness function such that the problem gets harder for direct encoding, but does not change the difficulty for developmental mappings. And similarly, modify the fitness function such that the problem gets harder for developmental mappings but does not change the difficulty for direct encodings.

Motivated by previous results on developmental mappings and phenotypic complexity [15], we hypothesize that one can construct such fitness function modifiers based on phenotypic complexity. Our approach to complexity is based on Kolmogorov complexity [16].

2 Background

We will use the following notation. The set of binary strings of length n is denoted $\{0,1\}^n$. If x is a binary string, then x_i denotes the ith bit of x and $x[i...j]$ denotes the substring of x beginning with the ith bit of x and ending with the jth bit of x. The notation x^n denotes n repetitions (or concatenations) of the binary string x. If x and y are binary strings, then $\ell(x)$ denotes the length of x, and if $\ell(x) = \ell(y)$, then $x \oplus y$ denotes the binary XOR-operation on x and y. Recall that $x \oplus x = 00\cdots0$ and $x \oplus 00\cdots0 = x$ for all binary strings x. The set of natural numbers is denoted $\mathbb{N}$. Each binary string is associated with a natural number by the standard enumeration $\epsilon, 0, 1, 00, 01, ...$ The set of real numbers is denoted $\mathbb{R}$, and the set of non-negative real numbers is denoted $\mathbb{R}^+$.

2.1 Kolmogorov Complexity and Lempel-Ziv Compresssion

The fitness modifiers described later are based on complexity. Kolmogorov complexity is quantitative measure of complexity based on Turing machines [16]. (See eg. [17] to recall the definitions of TMs and partial recursive functions.)

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

$$C(x) = \min_{p \in \mathbb{N}} \{\, \ell(p) \mid \phi(p) = x \,\}, \tag{1}$$

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.

This complexity measure is not computable, but we use the Lempel-Ziv compression algorithm [18] to approximate the Kolmogorov complexity. In the experiments described below, we used the standard compression library `zlib` [19]. For practical purposes, each bit in a binary string is encoded as the ASCII-character 0 or 1. The resulting buffer of 0 and 1 characters is compressed using `lzip`. We denote the length of the compressed result as $C_{LZ}(x)$. *Example:* The Lempel-Ziv complexity of a binary string consisting of 100 zeros is $C_{LZ}(0^{100}) = 12$.

2.2 Developmental mappings and fitness functions

The literature on AD describes many developmental systems and mappings [10]. They vary considerably, but many can be classified as either a grammatical or a chemical approach. We decided to use the developmental mapping described by Kitano in [12]. This is an example of a grammatical approach to development. With this mapping, the genotype represents a matrix-rewriting grammar. The phenotype (a quadratic matrix) is obtained by iterated rewriting of a start symbol. We will refer to this mapping as the *Kitano mapping*.

The choice of mapping was motivated by its simple design, and our previous experience with the mapping. The literature often refers to this mapping. For example, Siddiqi and Lucas compared Kitano mapping to direct encoding for evolving neural networks [14]. In our particular implementation of Kitano mapping, the genotype is 357 bits long. See [15] for more details about our implementation of the mapping.

We will contrast Kitano mapping with a *direct encoding*. A direct encoding is here simply the identity function from genotype to phenotype. So the genospace is here identical to the phenospace in direct encodings.

While developmental mappings are traditionally applied to design optimization problems, they are here applied to *pseudo-boolean* function optimization [20]. These functions are more convenient for our purpose than design optimization problems. The pseduo-boolean fitness functions chosen were OneMax [21], DebDeceptive and RoyalRoad [22], because they are well-known, simple to implement, and has varying hardness. The function OneMax is relatively simple, while the function DebDeceptive seems harder. These functions are defined in Appendix B. All three fitness functions have an optimum at $x^* = 111\cdots1$.

2.3 Fitness function modifiers

The main approach in this paper is to find uniform ways of modifying fitness functions to make them either hard for developmental mappings and unchanged for direct encodings, or hard for direct encodings and unchanged for developmental mappings. This is achieved with the introduction of fitness function modifiers.

In this paper, a *fitness function modifier* is an operator which, when given a pseudo-boolean function, returns a new, modified pseudo-boolean function . We describe two fitness function modifiers, one which we will call *Complexity Filter* and the other which we call *Complexity Translate*. They are parametrized by positive real-valued parameters c_f and c_t respectively. A class of problems is defined by a fitness function and a fitness function modifier with an accompanying range of its parameter.

Definition 2 (Complexity Filter). *Given a pseudo-boolean fitness function $f : \{0,1\}^n \to \mathbb{R}^+$, and a positive, real value c_f, the Complexity Filter returns a new pseudo-boolean fitness function $f^\star : \{0,1\}^n \to \mathbb{R}^+$ defined as follows:*

$$f^\star(x) = \begin{cases} f(x) & \text{if } C_{LZ}(x) < c_f \cdot \ell(x), \text{ and} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where x^* is the optimal argument to the function f.

The modifier is parametrized by the value of c_f. The returned function $f^\star$ is equivalent to the original function f when the parameter c_f is much higher than 1.0. When the value of c_f is close to zero, the modified function $f^\star$ evaluates to 0 for almost all arguments, and becomes a needle-in-the haystack function. Informally, the Complexity Filter "removes information" from the fitness landscape depending on phenotypic complexity. If the optimal argument x^* to the function f has low complexity and the parameter c_f is above a certain threshold, then the original function f and the modified function $f^\star$ share the same optimal argument.

Definition 3 (Complexity Translate). *Given a pseudo-boolean fitness function $f : \{0,1\}^n \to \mathbb{R}$, and a binary string z of length n with $C_{LZ}(z) = c_t \cdot n$, the Complexity Translate modifier returns a new pseudo-boolean fitness function $f^\ddagger : \{0,1\}^n \to \mathbb{R}$ defined as follows:*

$$f^\ddagger(x) = f(x \oplus z). \tag{3}$$

We consider this modifier to be implicitly parametrized by the value of c_t. To apply the modifier for a certain parameter setting of c_t, we have to search for a binary string z with Lempel-Ziv complexity $C_{LZ}(z) = c_t \cdot \ell(z)$. It might not be possible to find such a binary string for all values of c_t, so the complexity translate modifier may be undefined for some values of c_t. In almost all cases, the modified fitness function $f^\ddagger$ will have a different optimal argument than the original fitness function f. In the general setting, if a fitness function f has an optimum value of $f(x^*)$ for the argument x^*, then the modified fitness function $f^\ddagger$ has optimum value $f^\ddagger(x^* \oplus z) = f((x^* \oplus z) \oplus z) = f(x^* \oplus (z \oplus z)) = f(x^*)$ for the argument $x^* \oplus z$. In the case where $z = 0^{\ell(x)}$ (corresponding to a low parameter setting of c_t), the functions f and $f^\ddagger$ are equal, and have the same optima. The parameter setting of c_t changes the complexity of the optimal argument of the modified fitness function $f^\ddagger$.

How does the Complexity Translate modifier alter the complexity of the optimal solution of the modified function? We can sandwich the Kolmogorov complexity of the new optimum $x^* \oplus z$ between two values as follows:

$$\max(C(x^*), C(z)) - \min(C(x^*), C(z)) - c \leq C(x^* \oplus z) \tag{4}$$

$$C(x^* \oplus z) \leq C(x^*) + C(z) + c, \tag{5}$$

where c is a small constant. A proof is given in Appendix A.

In our case, the optimal solution x^* has very low complexity, so if z has high complexity (i.e. high value of c_t), then by inequality (4), the optimum $x^* \oplus z$ of the modified function f' will also have a relatively high complexity. On the other hand, if z also has low complexity (low value of c_t), then by inequality (5), the optimal solution of the modified function will have a low complexity .

In contrast to the Complexity Filter modifier, the Complexity Translate modifier does not remove information from the fitness landscape. But the points in

the fitness landscape are translated, such that for high c_t-settings, the optimum is moved to a complex point in the fitness landscape. The Discussion section continues the description of the fitness modifiers.

3 Experiments

The objective of the experiments is to study how the fitness function modifiers change the behavior of a genetic algorithm using either a direct encoding, or a developmental mapping.

Density estimation of phenotypic complexity distribution From the definition of the fitness function modifiers, it is reasonable to assume that differences in the phenotypic complexity distribution of direct encoding and Kitano mapping may affect the outcome of the evolutionary experiments.

To get a clearer picture of this issue, we randomly sampled one million genotypes to the direct encoding, and one million genotypes to the Kitano mapping. Then, we mapped the genotypes into phenotypes using the direct encoding (i.e. uniformly distributed random strings) and the Kitano mapping respectively. Then we computed the Lempel-Ziv complexities of the phenotypes, and from these values estimated the distribution of phenotypic complexity in the direct encoding, and in the Kitano mapping.

GA settings The evolutionary experiments were implemented using the GAlib [23] genetic algorithm library. Linear roulette wheel selection was used with a mutation rate of 0.01 per bit in the genome, a crossover rate of 0.9 and a population size of 30. The maximum number of allowed generations was 50000. The fitness functions were instantiated with problem size $n = 100$.

Complexity Filter Experiments were carried out using the Complexity Filter with a parameter setting of c_f in the interval 0.1 to 1.0, with step size 0.05 and 50 repetitions for each parameter setting. The modifier was applied to OneMax , RoyalRoad and DebDeceptive first using direct encoding and then Kitano mapping.

Complexity Translate By decompressing binary strings of varying lengths using the Lempel-Ziv algorithm, we constructed a set Z of 901 binary strings with Lempel-Ziv complexities almost uniformly distributed on the interval 12 to 36. This corresponds to c_t values in the interval 0.12 to 0.36. (Notice that the parameters c_f and c_t are not directly comparable.)

For each binary string z in this set Z, one experiment was carried out for all three fitness functions using the Complexity Translate modifier with z as parameter. Again, the experiments were performed for both direct encoding and Kitano mapping.

4 Results

4.1 Density plot

The results from the estimation of complexity distributions are shown in Figure 3. The most notable difference between the two distributions, is that most

of the random strings (i.e. phenotypes of direct encoding) have a relative high complexity, and the variance in complexity is little. This result complies with the Incompressibility Lemma in Kolmogorov complexity. On the other hand, the Kitano mapping has a larger variance.

The mean complexity in the Kitano mapping distribution is not much lower than the mean complexity in the random string distribution.

4.2 Evolutionary runs

The results of the experiments are presented as scatter plots. Each point corresponds to the result of one evolutionary run. The plots show the number of generations ($\log_{10}$-scaled) used before either the optimal phenotype was discovered or the upper limit of 50000 generations was reached. This is plotted against the parameter of the employed fitness modifier. Runs using direct mapping are plotted as crosses while runs using Kitano mapping are plotted as circles.

Complexity Filter Figure 1 shows the results for OneMax and RoyalRoad modified with the Complexity Filter. The plot for DebDeceptive is not shown, but shows a similar trend as for the two other problems. Kitano mapping seems to be less affected than direct encoding by parameter settings of c_f above 0.1.

The performance of direct encoding increases suddenly as the value of c_f reaches 0.35. A general observation is that Kitano mapping is more efficient at solving the three problems. Even for very low values of c_f, Kitano mapping is able to generate optimal solutions.

Direct encoding however, when applied to OneMax or RoyalRoad , did not find any optimal solution in any of the runs for low c_f-values. Direct encoding did not find the optimal solution in any of the runs with DebDeceptive .

Complexity Translate Results for OneMax and RoyalRoad modified with the Complexity Translate modifier are shown in Figure 2. The plot for DebDeceptive is not shown, but follows a similar trend as the other two functions.

In these experiments, the direct encoding seems to be relatively insensitive to changes of the parameter c_t. The effectiveness of Kitano mapping is clearly reduced as c_t increases. Kitano mapping did seldom find the optimal solution in any runs with high c_t values. However, Kitano mapping seems to outperform the direct encoding for very low values of c_t.

5 Discussion

Except from extreme cases, Complexity Filter only affects the performance of direct encoding, and Complexity Translate only affects the performance of Kitano mapping. This result is interesting because it allows us to better understand and appreciate special complexity-related characteristics of Kitano mapping.

The density plot gives some clues about the results of the evolutionary runs. The density estimation plot shows that most of the random strings have a high Lempel-Ziv complexity. When we apply the Complexity Filter modifier with c_f

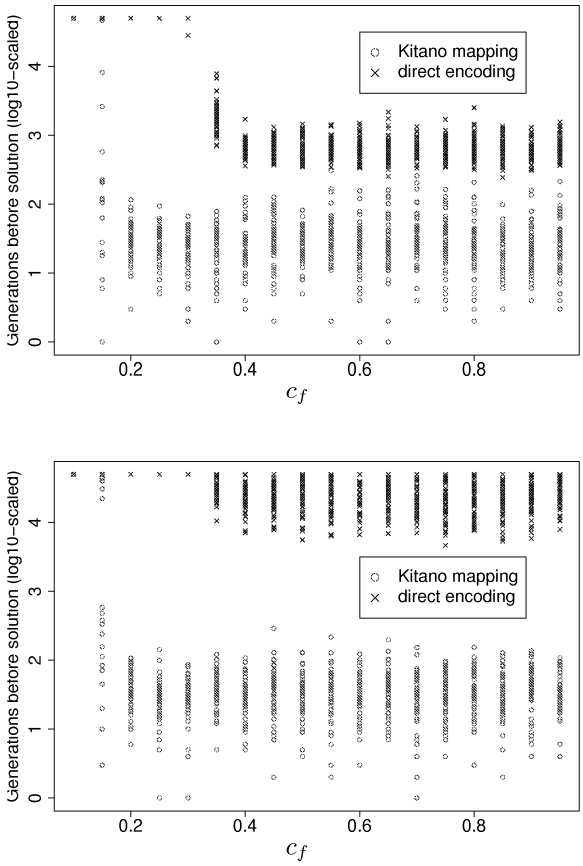

Fig. 1. Complexity Filtering of OneMax (top) and RoyalRoad (bottom).

lower than the main peak of the random strings - as indicated with the dashed vertical line in Figure 3 - the fitness of binary strings right of the line will be set to 0. In some of the experiments, the performance of direct encoding suddenly increased as c_f reached 0.35. This complies well with the density plot, where the main peak begins at around 0.35. The Kitano mapping distributes the phenotypes more evenly over possible ranges of complexity. This implies that when using a relatively low value of c_f, Kitano mapping still has a large portion of its fitness landscape preserved. With values of c_f below 0.15, the performance of the Kitano mapping suddenly decreases. This complies well with the distribution of complexities for the Kitano mapping. If we had replaced the evolutionary algorithm with a completely random search, then the density plot might have been sufficient to predict the performance of the two mappings on functions modified with Complexity Filter. However, when using an evolutionary algorithm, it

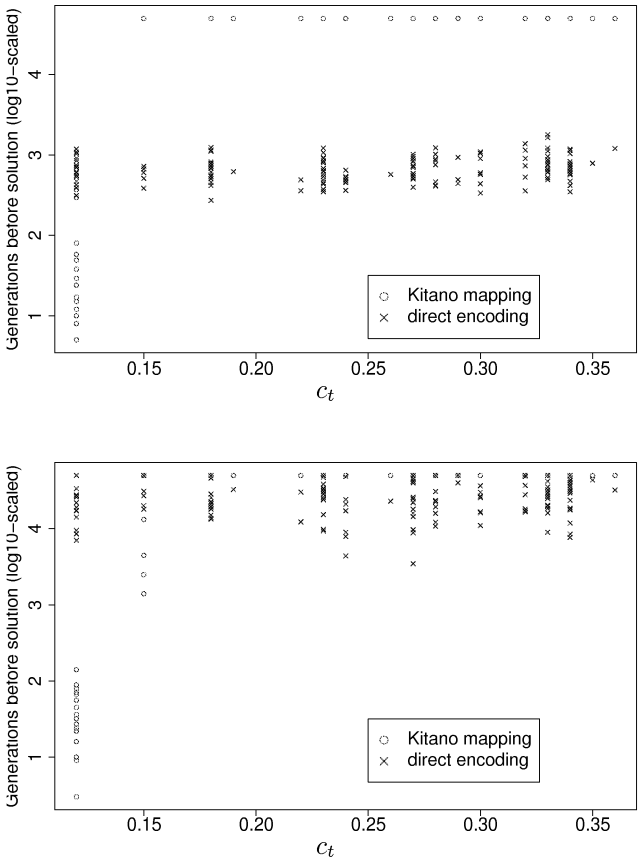

Fig. 2. Complexity Translate of OneMax (top) and RoyalRoad (bottom).

is unreasonable to claim that the density plot gives the full explanation of the results from the evolutionary runs.

To see this: Suppose that in phenospace of the Kitano mapping, the complex and regular phenotypes are mixed together uniformly. Then, when the complex phenotypes are given 0 fitness, the Complexity Filter would render the Kitano fitness landscape highly irregular.

Our results indicate that this interpretation is not correct because Kitano mapping is able to cope with the Complexity Filter. We suspect that the low complexity phenotypes are more tightly grouped in the Kitano search space. From the density plot, we know that low complexity phenotypes occupy a relatively larger part of the Kitano search space. When using the Complexity Filter, large regions of the Kitano search space will therefor remain relatively unaffected

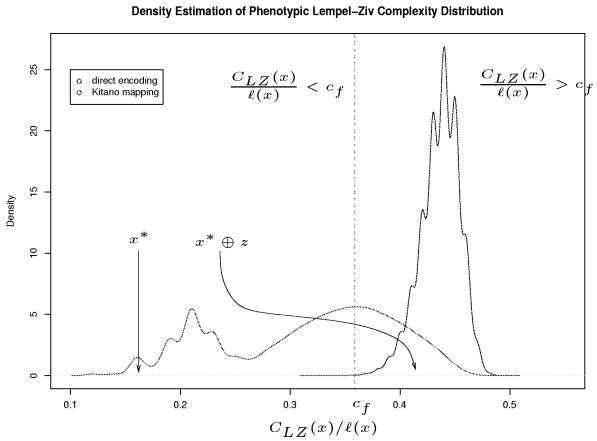

Fig. 3. Density Estimation of Phenotypic Lempel-Ziv Complexity Distribution.

by the Complexity Filter. This is not a claim, but a possible explanation of the results using the Complexity Filter.

As mentioned in the background section, when we apply the Complexity Translate modifier with a complex binary string z, the complexity of the optimum increases. This is illustrated in Figure 3 with a simple, original optimum x^*, and the modified optimum $x^* \oplus z$. Given a high parameter setting of c_t, the optimum of the modified fitness function is moved to the right along the horizontal axis (or to a more "complex point in the fitness landscape").

The density plot might trick us into believing that using direct encoding, it is much easier to find a specific complex phenotype, than to find a specific regular phenotype. However, under a totally uniform distribution, all binary strings are equally probable. Consequently, it is not surprising that the direct encoding is not affected by the setting of the c_t parameter in Complexity Translate. The results from OneMax modified with Complexity Translate is a good example of this. It is more challenging to explain why the Kitano mapping was affected by the c_t parameter of Complexity Translate. It is unclear whether the density plot can give us any hints about the reason for this.

The construction of the Complexity Translate modifier was motivated by results from earlier experiments with the Kitano mapping [15]. Results from that work indicate that the distance-preservation of the Kitano mapping from genotype to phenotype depends on phenotypic complexity. If the phenotype has high complexity we can expect a poor preservation of distance. One could conjecture that distance-preservation of the mapping might be a necessary condition for an evolutionary algorithm to find the optimal solution [24], and this might explain our results with Kitano mapping on Complexity Translate. Furthermore, Downing observed that it was harder to evolve irregular binary strings with his developmental mapping [25]. Furthermore, if Complexity Translate translates

the optimum to a bitstring outside the image of the Kitano mapping, then the optimum will of course not be found with Kitano mapping.

6 Conclusion

We have approached the problem of distinguishing for which problems a developmental mapping is beneficial in evolutionary computation by focusing on the classical Kitano mapping [12].

Two general methods of modifying fitness functions were described: the Complexity Translate modifier, and the Complexity Filter modifier. Their construction was motivated by previous results on phenotypic complexity and developmental mappings [15].

Experimental results indicate that the Complexity Filter modifier can make a problem harder for direct encoding, while preserving the difficulty of the problem for the Kitano mapping. Furthermore, the results indicate that Complexity Translate modifier can make a problem harder for the Kitano mapping, but unchanged for a direct encoding.

The approach was demonstrated on three different fitness functions: OneMax , DebDeceptive and RoyalRoad .

A density estimation plot of phenotypic complexity distribution aided in understanding complexity in direct encoding and Kitano mapping. Certain features in the results seem explainable simply by observing this density plot.

The authors believe the results are significant because they allow better appreciation and understanding of the differences between direct encodings and developmental mappings. Better understanding of the special characteristics of developmental mappings is needed if they are to be used in a practical setting.

A A short proof

Proposition 1. *Let x and y be two binary strings of equal length. Then*

$$\max(C(x), C(y)) - \min(C(x), C(y)) - c \leq C(x \oplus y) \tag{6}$$

$$C(x \oplus y) \leq C(x) + C(y) + c, \tag{7}$$

where c is a (small) constant.

Proof. Inequality (7) is trivial: If we have two programs which generate the binary strings x and y independently, then we can apply the XOR-operation (suppose the program for XOR takes c bits) to obtain $x \oplus y$. Now, to see inequality (6), suppose first that x is less complex than y. Assume p_1 is the shortest program which generates x, and p_2 is the shortest program which generates $x \oplus y$. We can make a program p_3 which combines p_1 and p_2 to compute $x \oplus (x \oplus y) = y$. The length of the program p_3 is $\ell(p_3) = \ell(p_1) + \ell(p_2) + c$, where c is the length of the program which computes the XOR-operation. The shortest program which computes y is therefor no longer than the length of program p_3, so we have $C(y) \leq \ell(p_1) + \ell(p_2) + c$, which implies $C(y) - C(x) - c \leq C(x \oplus y)$. An analog argument applies when x is more complex than y. □

B Definition of fitness functions

$$\textsc{OneMax}_n(x) = \sum_{i=1}^{n} x_i \quad \text{where } x \in \{0,1\}^n,$$

$$\textsc{RoyalRoad}_n(x) = \sum_{i=0}^{a} \prod_{j=ib+1}^{(i+1)b} x_j \quad \text{where } x \in \{0,1\}^n \text{ and } n = a \cdot b,$$

$$\textsc{DebDeceptive}_n(x) = \sum_{i=1}^{m} g(x[im...im+3]) \quad \text{where } x \in \{0,1\}^n, \text{ and } n = 4 \cdot m$$

$$g(x) = \begin{cases} 0.6 - 0.2h(x) & \text{if } h(x) < 4 \\ 1 & \text{if } h(x) = 4 \end{cases} \quad \text{and } h(x) = \sum_{i=1}^{4} x_i$$

References

1. Gruau, F.: Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm. PhD thesis, France (1994)
2. Belew, R.K., Kammeyer, T.E.: Evolving aesthetic sorting networks using developmental grammars. In: Proc. Fifth Int. Conf. on Genetic Algorithms, Morgan Kaufmann (1993) 629
3. Miller, J.F., Thomson, P.: A developmental method for growing graphs and circuits. In: Proc. 5th Intl. Conf. on Evolvable Systems, ICES'2003. Volume 2606 of LNCS., Springer-Verlag (2003) 93–104
4. Jakobi, N.: Harnessing morphogenesis. Technical report, School of Cognitive and Computing Sciences, University of Sussex (1995)
5. Eggenberger, P.: Evolving morphologies of simulated 3d organisms based on differential gene expression. In: Proc. 4th European Conf. on Artificial Life (ECAL97), Cambridge: MIT Press (1997)
6. Astor, J., Adami, C.: Development and evolution of neural networks in an artificial chemistry. In: Proc. Third German Workshop on Artificial Life, Verlag Harri Deutsch (1998) 15–30
7. Gordon, T., Bentley, P.: Towards development in evolvable hardware. In: The 2002 NASA/DoD Conf. on Evolvable Hardware, Jet Propulsion Laboratory, California Institute of Technology, IEEE Computer Society (2002)
8. Downing, K.: Developmental models for emergent computation. In: Proc. 5th Intl. Conf. on Evolvable Systems, ICES'2003. Volume 2606 of LNCS., Springer-Verlag (2003) 105–116
9. Tufte, G., Haddow, P.C.: Building knowledge into developmental rules for circuit design. In: Proc. 5th Intl. Conf. on Evolvable Systems, ICES'2003. Volume 2606 of LNCS., Springer-Verlag (2003) 69–80
10. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. Artificial Life **9** (2003) 93–130
11. Kumar, S., Bentley, P.: On Growth, Form and Computers. Academic Press (2003)
12. Kitano, H.: Designing neural networks using genetic algorithm with graph generation system. Complex Systems **4** (1990) 461–476

13. Gruau, F., Whitley, D., Pyeatt, L.: A comparison between cellular encoding and direct encoding for genetic neural networks. In: Genetic Programming 1996: Proc. First Annual Conf., Stanford University, CA, USA, MIT Press (1996) 81–89
14. Siddiqi, A., Lucas, S.: A comparison of matrix rewriting versus direct encoding for evolving neural networks. In: Proc. 1998 IEEE Intl. Conf. on Evolutionary Computation, IEEE Press (1998) 392–397
15. Lehre, P.K., Haddow, P.C.: Developmental mappings and phenotypic complexity. In: Proc. Congr. on Evolutionary Computation, Vol. 1, IEEE Press (2003) 62–68
16. Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and Its Applications. 2nd edn. Springer-Verlag, New York (1997)
17. Lewis, H.R., Papadimitriou, C.H.: Elements of the Theory of Computation. 2nd edn. Prentice Hall (1997)
18. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory **23** (1977) 337–342
19. Gailly, J.L., Adler, M.: zlib general purpose compression library. (2002)
20. Wegener, I.: Methods for the Analysis of Evolutionary Algorithms on Pseudo-Boolean Functions. In: Evolutionary Optimization. Kluwer Academic Publishers (2002)
21. Mühlenbein, H.: How genetic algorithms really work. i. mutation and hillclimbing. In: Proc. of PPSN II. (1992) 15–25
22. Mitchell, M., Forrest, S., Holland, J.H.: The royal road for genetic algorithms: Fitness landscapes and GA performance. In: Proc. of the First European Conf. on Artificial Life. (1992) 245–254
23. Wall, M.: GAlib: A C++ Library of Genetic Algorithm Components version 2.4 Documentation. MIT. Revision b edn. (1996)
24. Sendhoff, B., Kreutz, M., von Seelen, W.: A condition for the genotype-phenotype mapping: Causality. In: Proc. Seventh Intl. Conf. on Genetic Algorithms (ICGA97), San Francisco, CA, Morgan Kaufmann (1997)
25. Downing, K.: Development and the Baldwin Effect. Artificial Life **10** (2004) 39–63