ON IMPACT OF ENVIRONMENT ON THE COMPLEXITY GENERATED BY EVOLUTIONARY DEVELOPMENT

Michal Bidlo and Lukas Sekanina

Brno University of Technology
Faculty of Information Technology
Bozetechova 2, 61266 Brno
Czech Republic
bidlom@fit.vutbr.cz, sekanina@fit.vutbr.cz

Abstract: *A formal concept of a general developmental system is proposed in order to investigate the impact of an external information supplied to the developmental system from the environment on the complexity of developed objects. Genetic algorithm is utilized to design a developmental system (based on the formal model) that is able to generate as complex objects (strings) as possible. Dependence of the developed string complexity on the various properties of the developmental system and environment is investigated. The conducted experiments show that the complexity of target objects depends more on the information comming from environment than on the information included in the genotype.*

Keywords: *Evolutionary algorithm, development, environment, rewriting system, complexity.*

1 Introduction

In evolutionary algorithms (EA), the process of development (more precisely, computational development [14]) is usually considered as a non-trivial *genotype–phenotype mapping*. While genetic operators work with genotypes, the fitness calculation is applied on phenotypes created by means of a developmental system.

The utilization of the computational development is primarily motivated by the fact that natural development is one of the phenomena which is primarily responsible for the extraordinary diversity and sophistication of living creatures. It is assumed that the computational development (inspired by natural development) in connection with an evolutionary algorithm might be utilized to achieve the evolution of complex artificial objects and other objectives desired by evolutionary design systems, including adaptation, compacting genotypes, reduction of search space, regulation, regeneration, repetition, robustness, evolvability, parallel construction, emergent behavior and decentralized control (as discussed in [14]). Kumar and Bentley summarized an authoritative introduction to developmental biology and computational developemnt, together with various applications of the bet scientists in their fields representing the abilities of biologically inspired development, in [21].

In nature, the process of development is influenced by the genetic information of the organism and the environment in which the process is carried out. Cells use the mechanism of transcription and translation to read each gene and produce the string of amino acids that makes up a protein. Proteins activate or suppress synthesis of other genes, work as signals among cells, influence internal functions of the cells and perform many other important roles. Therefore, they control the growth, position and behavior of all cells [1].

Similarly, we can identify three important components in any artificial evolutionary system which utilizes the concept of development: genotype (a prescription supplied by EA for constructing a target object), phenotype (the target object which grows and is modified in the process of development) and environment (which influences the process of development). In the most general case, the process of computational development could be implemented by an algorithm which continually and iteratively samples the genotype, environment and phenotype and, as the result, updates the phenotype, environment and itself. Although a number of developmental models have been proposed in the recent years (see Section 2), none of them fully considers the proposed scenario. In most cases the following two scenarios have been adopted:

1. The phenotype is solely constructed on the basis of genetic information. A user defines a small instance of the target object (called the embryo) at the beginning of development. The genetic information is interpreted as a program (consisting of instructions, commands for a turtle etc.) which modifies the embryo in order to obtain a complex target solution. Examples include the evolutionary design of analog circuits [13], antennas [12], mechanical constructions [11] or arbitrary large sorting networks [22].

2. In addition to the genetic information, the program which constructs the target object utilizes a specific knowledge which is not included in the chromosome (genotype). This knowledge is usually interpreted as an environment surrounding the growing phenotype. The environment can be influenced and so modified

by various entities, including the growing phenotype and the program implementing the development. Typically, the interaction of genetic information and environment is modeled by constructions such as: *if (variable 2 of environment is greater than 5) then perform the instruction specified by gene 3 and diffuse X1 and X2 to environment.* This approach is called as an implicit embryogeny [15].

While the first scenario has been adopted to produce human-competitive results in real-world applications [13, 12, 11], the second scenario has been investigated mostly on artificial problems (3D pattern design [15], combinational circuit evolution [8]). It seems that it is difficult to evolve the required solutions using the second scenario although it offers us much more variability in the developmental mechanism.

In order to design a target entity by means of an evolutionary algorithm that utilizes a developmental phase, two sources of information can be utilized – genetic information and environment. In the first scenario, only the genetic information is utilized. In the second scenario, the genetic information as well as the information provided by environment is utilized. Our experience from nature and the experiments performed by Kumar and others suggest that the second approach must be able to generate much more complex entities than the approach ignoring the environment. Therefore, the hypothesis that will be defended in this paper is:

If an evolutionary design system (with a developmental phase) utilizes a piece of information from chromosome (i.e. genetic information) and another piece of information from environment (i.e. an external information), then more complex entities can be evolved in comparison with the situation in which the environment is not utilized and only the genetic information is considered for the development.

The strategy adopted to confirm this hypothesis is as follows: The process of development will be modeled by D system which is a generalized rewriting system possessing the Turing-machine computational power based on Lindenmayer system (L system) [18]. The first objective is to design rewriting rules of D system such that D system generates as complex strings as possible (the complexity will be measured by LZ-based compression algorithm [6]). Since the process of rewriting rules design is difficult, EA will be utilized to accomplish this task. The second objective is to design rewriting rules together with a pattern (i.e. environment) which influences the application of the rewriting rules. Again, the goal is to evolve as complex strings as possible. The comparison of amount of information utilized by these two scenarios and the complexity of obtained strings will show the differences between the two approaches.

The paper is structured as follows. Section 2 surveys the area of evolutionary algorithms that use a developmental phase. In order to analyze the proposed experiments rigorously, the developmental process as well as environment is defined formally in terms of theoretical computer science in Section 3. Sections 4 and 6 describe parameters of the proposed developmental model, genetic algorithm and environment used in experiments. Experimental results are summarized in Section 7 and discussed in Section 8. Concluding remarks are given in Section 9.

2 A Brief Survey of Related Works

Many approaches to the research and simulation of biologically-inspired development in the area of computer science have been published so far. They have been devised in order to demonstrate a possible solution to a particular problem or to investigate the properties of development in a particular application. In this section, several references to the research in this area will be given which relates in some way the proposed approach.

Lindenmayer introduced parallel rewriting systems (called L systems) for modeling the development of multicellular organisms in nature [18]. The basic ideas of L systems gave rise to an aboundance of language-theoretic problems, both mathematically challenging and interesting from the point of view of diverse applications (e.g. in computer graphics).

Wilson proposed a representation for biological development for simulating the evolution of simple multicellular systems using the genetic algorithm [26]. The representation consists of a set of production-like growth rules constituting the genotype, together with the method of executing the rules to produce the phenotype. In his system, cells contain an identical set of rules, which are the subject of artificial evolution. The development proceeds in discrete time steps, executing rules in the cells according to the conditions specified in the rules (e.g. presence of a given type of cell in the cell's neighborhood, receiving a signal emited by the surrounding cells etc.).

Haddow, Tufte and van Remortel have presented a case study where the mathematical formalism of L systems have been considered in the application of the development of digital circuits. The initial results have been obtained using extrinsic evolution. Moreover, an implementation platform has been presented for intrinsic evolution with development enabling on-chip evaluation of growing solutions [9].

Hornby and Pollack have described a system for creating generative specifications by combining L systems with evolutionary algorithms and applied it to the problem of generating table designs. Designs evolved by this system have reached an order of magnitude more parts than previous generative systems [5, 2]. By comparing

it against a non-generative encoding the authors have recognized that the generative system produces designs with higher fitness and is faster than the non-generative system [11].

Gordon has demonstrated that evolution can learn and encode useful circuit design abstractions in a developmental process. The author has shown that the computational power of the pattern formation model is able to learn patterns that can successfully be mapped to fully functional adder circuits, which is an important step towards tackling real-world problems with development [7].

Lehre and Haddow have investigated the effect of phenotypic complexity on distance correlation plots for two developmental mappings, a mapping based on L systems, and a 2D cellular automata mapping. Their treatment of complexity is based on the theory of Kolmogorov complexity [17]. A genotype sampling algorithm called Crossection Walk has been introduced [16].

Miller has introduced a method for evolving a developmental program inside a cell to create multicellular organisms of arbitrary size and characteristics. The cell genotype is evolved so that the organism will organize itself into well defined patterns of differentiated cell types (e.g. the French Flag). In addition, the cell genotypes are evolved to respond appropriately to environmental signals that cause metamorphosis of the whole organism. A number of experiments have been described that show that the organisms exhibit emergent properties of self-repair and adaptation [19].

Hartmann et al have investigated the genotypic complexity of evolved multiplier and adder circuits (evolved genotypes) using the LZ-compression. The results have shown that the sample mean complexity value for random bit strings was always greater than those of evolved genotypes implying more regularity in the evolved genotypes. Different encoding methods have been implemented for the genotypes and the results have shown that not only the complexity level for a given encoding varies, but that the range and distribution of the complexity values varies for different encoding schemes. The range provides an indication as to the minimum genotype length that may provide a big enough search space so as to find a genotype that can be developed into the phenotype sought. The complexity distribution also indicates how many multipliers or adders may be found in the given search space [10].

Stanley and Miikkulainen proposed a taxonomy providing a unified context for long-term research in the field of artificial embryogeny (AE) so that implementation decisions can be compared and contrasted along known dimensions in the design space of embryogenic systems[23]. The authors reviewed prior work in artificial embryogeny by examining two parallel lines of research in this field: (1) grammatical approach, originated by Lindenmayer [18] and (2) cell chemistry approach, inspired by the early work of Turing [25]. On the basis of knowledges in the area of development they identified five major dimensions which, by their own account, "... can help in selecting the right combination of developmental mechanisms to produce a computationally efficient AE methodology." These dimensions include cell fate, targeting, heterochrony, canalization and complexification. See [23] for details.

3 Introducing a Formal Concept of Developmental Systems

The models mentioned in Section 2 mostly represent application-specific developmental methods without a substantial interaction of genotype or phenotype with an environment surrounding the developmental system as usual in nature. Moreover, no general concept of development in the area of computer science has been proposed yet. In this section, a formal model representing the concept of a general developmental system is introduced which is intended to serve as a basis for the description of developmental models and proving their properties.

3.1 Basic Terminology

Before introducing the definition of the developmental system, some basic terms and symbols from the area of theoretical computer science have to be defined [20].

Alphabet, cardinality of a set. An *alphabet* is a finite set of symbols. For example, $\Sigma = \{a, b\}$ is an alphabet, a and b are the *symbols* contained in Σ, i.e. $a \in \Sigma, b \in \Sigma$. We denote $|\Sigma|$ the *cardinality* (i.e. the number of elements – symbols) of the set Σ, in this case $|\Sigma| = 2$.

Mapping. Let A, B be sets. A *mapping* (m) of the set A into the set B assigns to every element $a \in A$ an element $b \in B$ and is denoted as $m : A \to B$. For example, consider a set $A = \{a, b, c\}$, a set $B = \{0, 1, 2, 3\}$ and the mapping m defined as $m(a) = 1, m(b) = 1, m(c) = 3$, i.e. m assigns to the elements a, b and c from the set A the elements 1, 1 and 3 from the set B, respectively. Alternatively, we can write the mapping m as the set of pairs (x, y), where $y = m(x), x \in A, y \in B$, i.e. $m = \{(a, 1), (b, 1), (c, 3)\}$.

String, length of string, empty string. A *string* is a finite sequence of symbols from an alphabet. For example, $u = \varepsilon, v = a, w = abbab$ are strings over the alphabet $\Sigma = \{a, b\}$, where ε denotes the sequence of zero symbols and is refered to as the *empty string*. The *length* $|s|$ of a string s is the number of symbols of s. Considering the strings from the previous example, $|\varepsilon| = 0, |v| = 1, |w| = 5$.

Power of a set, iteration of a set. Σ^i is defined as the set of all strings of length i composed of the symbols from Σ and is refered to as i-th *power* of the set Σ. For example, if $i = 2$ and $\Sigma = \{a, b\}$, then $\Sigma^2 = \{aa, ab, ba, bb\}$. The *iteration* of Σ, Σ^*, is defined as the union of all powers of Σ, i.e. $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$. Note that $\Sigma^0 = \{\varepsilon\}$ and $\Sigma^1 = \Sigma$.

Language, finite language. An arbitrary subset of an alphabet iteration is refered to as a *language* (i.e a set of strings over the alphabet), e.g. if $L \subseteq \Sigma^*$, then L is a language over the alphabet Σ. If the number of strings in L is finite, then L is said to be a *finite language*. If $L = \emptyset$ (i.e. $|L| = 0$), then L is said to be the *empty language*, otherwise L is a *non-empty language*.

Set of languages, power set. Let $\mathcal{L}$ be the set of all possible languages over the alphabet Σ and $\mathcal{F} \subset \mathcal{L}$ the set of all finite languages (i.e. the languages containing a finite number of strings). We denote $\mathcal{F}^i$ the set of all finite languages with the maximal length of a string i. $\mathcal{F}^i$ is actually the set of all subsets (so-called *power set*) of $\Sigma^0 \cup \Sigma^1 \cup \ldots \cup \Sigma^i$ denoted as $\mathcal{F}^i = 2^{\Sigma^0 \cup \Sigma^1 \cup \ldots \cup \Sigma^i}$.

For example, consider the alphabet $\Sigma = \{a\}$ and the maximal length of a string $i = 2$. Let us denote $S = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 = \{\varepsilon, a, aa\}$ the set of all strings of the maximal length 2 over Σ. We will express the power set $\mathcal{F}^2 = 2^S$ as the union of auxiliary sets F_0, F_1, F_2 and F_3, where $F_0 = \{\emptyset\}$ contains the empty set (the only zero-element subset of S), F_1 contains the set of all single-element subsets of S, i.e. $F_1 = \{\{\varepsilon\}, \{a\}, \{aa\}\}$, F_2 contains the set of all two-element subsets of S, i.e. $F_2 = \{\{\varepsilon, a\}, \{\varepsilon, aa\}, \{a, aa\}\}$ and F_3 contains the set S itself, i.e. the only three-element subset of S, $F_3 = \{\{\varepsilon, a, aa\}\}$. Finally, $\mathcal{F}^2 = F_0 \cup F_1 \cup F_2 \cup F_3 = \{\emptyset, \{\varepsilon\}, \{a\}, \{aa\}, \{\varepsilon, a\}, \{\varepsilon, aa\}, \{a, aa\}, \{\varepsilon, a, aa\}\}$ represents the set of all finite languages with the maximal length of a string equal 2 over the alphabet $\Sigma = \{a\}$.

3.2 Introducing a Formal Model of Development

By means of the terms defined in Section 3.1, a concept of general developmental system will be defined for the purposes of computer science. The definition constitutes a basis of a particular system performing a kind of development. First, D system (a general rewriting system) will be defined which represents a "growth-simulating" unit. Its definition is based on the definition of L system introduced in [18]. For the purposes of development, the growth-simulating unit is enhanced to be able to process more than one symbol by applying a single rewriting rule. Basically, the rewriting system is parallel and non-deterministic. We have devised the D system because L systems possess some limitations related to the simulation of multicellular development in biology for which L systems were originally developed (e.g. only one symbol is allowed to be rewrited by one rewriting rule etc. [20, 18]). Moreover, an environment surrounding the developmental system and an algorithm determining rigorously the developmental process will be introduced.

Definition 1 A ***D system*** is a triple $D = (\Sigma, \sigma, w_0)$, where Σ is a finite alphabet, $w_0 \in \Sigma^*$ is an initial string referred to as embryo and σ is a ***finite string substitution*** over the alphabet Σ defined as follows. Let $W \subseteq \Sigma \cup \Sigma^2 \cup \ldots \cup \Sigma^m$ be a set of strings. σ is a mapping of W into $\mathcal{F}^n$ such that for each $w \in W, \sigma(w) = F$, where $F \in \mathcal{F}^n$ is a finite non-empty language. If $|\sigma(w)| = 1$ for all $w \in W$, then D is referred to as ***deterministic D system***. Alternatively, $\sigma(w) = F$ can be expressed as ***rewriting rules*** of the form $w \rightarrow w'$ for all $w \in W$ and for all $w' \in \sigma(w)$, where $1 \leq |w| \leq m$ and $0 \leq |w'| \leq n$. The integers m, respective n actually set the maximal length of the string on the left-hand side, respective right-hand side of the rewriting rule. ***Derivation step*** is defined as a relation $x \Rightarrow y$. For $x, y \in \Sigma^*$, $x \Rightarrow y$ if and only if x contains at least one substring w such that $w \in W$ The string y is obtained by substituting the string $w' \in \sigma(w)$ for the substring w in x.

The D system was defined in order to provide a formal model of systems that are able to grow, shrink or alter their structure in some way during their life, i.e. to develop (similarly to living creatures, plants etc. in nature). The following example shows a particular instance of a simple D system, its components and embryo development through a series of derivation steps.

Example 1: A simple D system Consider the alphabet $\Sigma = \{a, b\}$, maximal length of the left-hand side of a rewriting rule is $m = 3$, maximal length of the right-hand side of a rewriting rule is $n = 2$, the set of strings of the left-hand sides of rewriting rules is $W = \{a, bb, abb\}$ and the embryo is $w_0 = abba$. Consider the finite

string substitution σ defined as follows:

$\sigma(a) = \{ba, b\}$, i.e. the rewriting rules (1) $a \to ba$, (2) $a \to b$,
$\sigma(bb) = \{ab, \varepsilon\}$, i.e. the rewriting rules (3) $bb \to ab$, (4) $bb \to \varepsilon$ and
$\sigma(abb) = \{bb\}$, i.e. the rewriting rule (5) $abb \to bb$.

σ defines five different rules (1), (2),..., (5). Some of them have their left-hand sides identical (e.g. the rules (1) and (2)), i.e. the D system is non-deterministic. This D system can be expressed formally accroding to the definition of D system as

$$D = (\{a, b\}, \{(a, \{ba, b\}), (bb, \{ab, \varepsilon\}), (abb, \{bb\})\}, abba) .$$

By using the rewriting rules, we can derive the following strings from the embryo w_0, for example:

By applying rule (1) on the underlined substrings: $\underline{a}bb\underline{a} \Rightarrow babbba$,
now, by applying rule (2): $babbb\underline{a} \Rightarrow babbbb$,
next, apply rule (5), for example: $b\underline{abb}bb \Rightarrow bbbbb$,
then, apply rule (4): $\underline{bb}bbb \Rightarrow bbb$
and finally, rule (4) again: $\underline{bb}b \Rightarrow b$.
Now, there is no rule to apply. We can write this sequence of derivation steps as follows: $abba \Rightarrow babbba \Rightarrow babbbb \Rightarrow bbbbb \Rightarrow bbb \Rightarrow b$.

The reader is sure to find another derivations in this example of D system. As obvious, there is no unambiguous prescript specifying which rule to apply on which position of the string; this is caused by the non-deterministic rewriting rules, i.e. more rules are applicable on the same substring or some substrings can overlap for the application of different rules.

On the basis of the D system, a formal definition of a developmental system will be proposed. A developmental algorithm will be introduced in order to prescribe the development of D system and enable an interaction with environment during the developmental process as usual in nature.

Definition 2 A ***developmental system*** is a foursome $\Delta = (D, E, e, M)$, where D is a D system, E is a finite environment alphabet, $e \in E^*$ is an environment and M is a developmental algorithm. If $E = \emptyset$, then $e = \varepsilon$ and the system does not interact with environment, otherwise the system is said to have been developing in the environment e.

Example 2: A simple developmental system Consider the D system from Example 1. Let $E = \{0, 1\}$ be the environment alphabet, where the symbols 0/1 forbid/permit the applications of rewriting rules on given positions in the string. Let $e = 01010101\ldots$ represent the environment "surrounding" the developmental system. We introduce a kind of developmental algorithm which will (1) control the rewriting process according to the environment e, (2) make the D system work deterministically and (3) perform parallel rewriting of non-overlapping substrings.

The developmental algorithm M:

Let $x_j \to y_j$ represent the rewriting rule (j).
Let a_i denote the i-th symbol of the string being developed.
Let e_i denote the i-th symbol of the environment.
Let r denote the number of rewriting rules.
(1) $i = 1$
(2) **while** $a_i \neq STOP$ **loop**
(3) **if** $e_i = 1$ **then**
(4) $j = 1$
(5) $match = false$
(6) **while** $match = false$ **and** $j \leq r$ **loop**
(7) **if** $x_j = a_i a_{i+1} \ldots a_{i+|x_j|-1}$ **then**
(8) mark $a_i a_{i+1} \ldots a_{i+|x_j|-1}$ for substitution by y_j
(9) $match = true$
(10) **end if**
(11) **end loop**
(12) **if** $match = true$ **then** $i = i + |x_j|$ **else** $i = i + 1$ **end if**
(13) **else**
(14) $i = i + 1$
(15) **end if**

(16) **end loop**
(17) substitute the right-hand sides of the appropriate rewriting rules for the marked substrings

Consider the embryo $w_0 = abba$ in the environment $e = 01010101\ldots$.

```
1  2  3  4  5  6  7  8...     (index i)
0  1  0  1  0  1  0  1...     (environment e)
a  b  b  a                    (embryo)
```

The system will develop according to the algorithm M as follows. The index i is initialized to 1. Since $e_i = e_1 = 0$, no rule can be applied at this position (even though the rules (1), (2) and (5) match with the appropriate substrings beginning at this position). Therefore, $i = i + 1 = 1 + 1 = 2$. Now $e_i = e_2 = 1$, the rewriting rules from (1) to (5) respectively are tested for match with the appropriate substring beginning with the symbol $a_i = a_2 = b$. The first rewriting rule matching at this position is the rule (3) that matches with the substring $a_2a_3 = bb = x_3$. The substring bb is marked to be substituted and $i = i + |x_3| = 2 + 2 = 4$. Next, because $e_i = e_4 = 1$, the rules from (1) to (5) respectively are tested again for match at the position $i = 4$. As rule (1) matches ($x_1 = a_4 = a$), the appropriate substring a is marked for substitution. Since the end of the string has been reached, the substitutions are performed. In this case, $y_3 = ab$ is substituted for the substring $a_2a_3 = bb$ and $y_1 = ba$ is substituted for the substring $a_4 = a$. Now, the developmental system looks like as follows.

```
1  2  3  4  5  6  7  8...     (index i)
0  1  0  1  0  1  0  1...     (environment e)
a  a  b  b  a                 (developed string)
```

In the next run of the developmental algorithm, only rule (1) is applicable at the position $i = 1$, its right-hand side $y_1 = ba$ is substituted for the substring $a_1 = a$ and the system gets the following form.

```
1  2  3  4  5  6  7  8...     (index i)
0  1  0  1  0  1  0  1...     (environment e)
a  b  a  b  b  a              (developed string)
```

The next execution of M to perform the third developmental step is left to the reader. This example of developmental system can be expressed formally according to the definition as

$$\Delta = (D, \{0,1\}, 010101\ldots, M)\,,$$

where D denotes the D system from Example 1 and M represents the developmental algorithm.

In connection with the developmental systems and the formal model proposed, two additional terms can be defined expressing formally the developmental process: *stage* of a developmental system and *developmental step*.

Definition 3 A ***stage*** of a developmental system is a string $s \in \Sigma^*$. w_0 is refered to as initial (or embryonal) stage of a developmental system. We denote $T \subseteq \Sigma^*$ the set of all possible stages of a developmental system.

Example 3: Stages of a developmental system The strings $abba, aabba, ababba \in \Sigma^*$ represent stages of the developmental system from Example 2.

Definition 4 A ***developmental step*** of a developmental system is represented by a binary relation $\vdash$ on the set of stages T of a developmental system. Let s and s' be the stages of a developmental system. If $M(\sigma, s, e)$ contains s' (i.e. $M(\sigma, s, e) = T', T' \in 2^T, s' \in T'$), then $s \vdash s'$ for all $s \in T$.

Example 4: A developmental step Consider the developmental system from Example 2, its embryonal stage $w_0 = abba$ and the environment $e = 01010101\ldots$. By executing the developmental algorithm $M(\sigma, w_0, e)$ we get a set of stages containing just one string (because of deterministic behavior of the developmental system) that has emerged by development of the embryo w_0 in the environment e according to the developmental algorithm M, i.e. $M(\sigma, abba, 010101\ldots) = \{aabba\}$. The substitutions performed by the algorithm M represents a developmental step $abba \vdash aabba$. Similarly, $aabba \vdash ababba$ denotes the second developmental step of the developmental system Δ.

3.3 Theoretical Aspects of the Formal Model

The formal concept of developmental system has been created in order to describe various developmental models in the area of computer engineering and science. There are two basic parts in the definition of the developmental system which are required to be general: (1) the D system and (2) the developmental algorithm. In fact, the D system poses a type-0 grammar which is able to represent an arbitrary algorithm-based model [20]. In order to perform interaction of the developing object (described by the D system) with the environment, developmental algorithm has been introduced. This algorithm can be, for example, represented by a Turing machine which is a general computational model, i.e. an arbitrary control of the developmental process and the interaction with the environment can be established [20].

It has been shown for many formal systems (e.g. regulated rewriting systems [4]) that introducing a kind of external control, whose rules can be, in fact, considered as an environment, involves the increase of complexity of generated objects (typically strings of symbols). Therefore, the hypothesis formulated in the introduction may seem as totally predictable. The systems for which this behavior was shown were devised by experts who analyzed them in detail, i.e. much of information were known about that systems. On the basis of that information mathematical propositions were applied in order to prove rigorously the desired features of a particular system. However, in our case, we will apply an evolutionary algorithm, which has no information about how to design the rules of the rewriting system and the environment in order to develop as complex strings as possible. The only information supplied to the system is the developmental algorithm and a fitness function calculating the complexity of the string being developed. Therefore, it is not possible to predict exactly the behavior of the developmental system to be evolved and the features stated in the hypothesis.

Considering the work of Stanley et al. [23], the developmental model is in its origin based on grammatical approach. Since it is intended to be general, full range of "parameters" (with respect to the algorithmic basis of the constituent elements of the formal model) related to the major dimensions of development introduced by Stanley et al. is possible.

4 The Developmental System Utilized in the Experiments

The definition of D system has introduced a rule-based rewriting system for simulating the growth, shrink and other structural changes of the developing object. We have chosen this approach because it provides many useful features for analyzing the properties of developmental models, for example:

- a straightforward way for parametrizing its attributes (e.g. the number of symbols and rewriting rules, length of the substring to be substituted etc.) and so the potential complexity of target object (string) to be developed,
- the rewriting rules can be controlled in a simple way independently on each other,
- the strings being developed can be interpreted in many ways (e.g. as programs, FPGA configuration strings, graphic objects etc.) and various their properties can be easily measured or
- the environment (also representable by a string) can be simply associated with the developing object (string) to allow both controlling the system development (e.g. to permit/forbid the application of rewriting rules) and possibly interacting with the developing object – the form of target system is affected not only by the developmental rules whose applicability depends on the environment but also by the environment itself (or the environment also can be affected by the developing object).

The developmental system used in the experiments that will be presented in next sections corresponds to the system defined in Examples 1–4. For convenience, we summarize the characteristics of the utilized model and describe its parameters relevant for the experiments. Table 1 shows a survey of all the parameters of the developmental system, their appellation, description and the range of values considered in the experiments.

4.1 The D System

The D system is interpreted as an abstract rewriting system over an alphabet Σ. We denote $c = |\Sigma|$ the number of symbols of the alphabet which the D system works over. The D system contains r rewriting rules of the form $x \to y$. The maximal length of the left-hand side ($|x|$), respective right-hand side ($|y|$) of the rewriting rule is restricted by the parameter m, respective n. The embryo $w_0 = a$ is considered in all experiments.

4.2 The Environment

For the experiments, we consider a simple instance of the environment with binary alphabet $E = \{0, 1\}$. The symbol 0, respective 1 is intended for forbidding, respective permitting the application of a rewriting rule at a given position in the string being developed. The environment e consists of symbols from the environment alphabet and is specified by the length p of a pattern $e_1e_2 \dots e_p$ which is repeated periodically along the string being developed, i.e. $e_1e_2 \dots e_pe_1e_2 \dots e_pe_1e_2 \dots e_p \dots$. Therefore, the system is said to have been developing in the environment.

Note that the environment can be viewed from three basic points: (1) the environment influences the developmental process performed by the D system, (2) the environment interacts directly with the string being developed and (3) combination of the approaches (1) and (2). In this paper, we will consider the first approach only.

4.3 The Developmental Algorithm

The developmental algorithm controls (1) the rewriting process performed by D system and (2) the interaction of the developmental process with the environment in order to affect the developing object (similarly to the development of living organisms in nature).

The algorithm goes through the string (in case of the first run through the embryo) from left to right. At each position in the string pointed by a position pointer, the left-hand sides of rewriting rules are tested successively for match the substring beginning with the symbol pointed by the position pointer. If a rule matches and the environment symbol under the position pointer equals 1, then the appropriate substring is marked for substitution and the position pointer is shifted right by the number of symbols of the left-hand side of the matching rule. If no rule matches or the environment symbol under the position pointer equals 0, then the symbol under the position pointer remains unchanged and the position pointer is shifted right by one symbol. If the end of the string is reached, then the right-hand sides of the matching rules are substituted for the appropriate substrings. The developmental process continues in the same way until no substring can be substituted or a given number of developmental steps (runs of the developmental algorithm) is performed. A semi-formal notation of the developmental algorithm is given in Example 2 in Section 3.2.

5 Comparison with the Wilson's Work

Stewart W. Wilson dealed, among others, with the evolution of development using the genetic algorithm. Moreover, he published some papers devoted to the classifier systems. Since these issues, together with the model proposed in this paper, have much in common, we will point out the significant features, similarities and differences in this section.

5.1 The Evolution of Development

Wilson conducted experiments with the growing multicellular structures using production-like growth rules [26]. His work was devoted to one-dimensional development, when a linear structure of cells (symbols) is developed. Therefore, it is desirable to point out the differences of the proposed model against the Wilson's one. The crucial dissimilarity represents the interpretation of the symbols and the understanding of environment in the developmental system.

In case of the Wilson's model, the "symbols" are understood rather as cells (larger building blocks constituting the phenotype) which embody the growth prescription — developmental rules — identical for each cell. The environment of a cell is considered as the features of cells in its surrounding and the developmental rules are sensitive to the context of cells within which the rules are executed. In fact, his model has stronger

Table 1: A survey of parameters considered in experiments

Parameter appellation	Description	Range
c	the cardinality of the string alphabet Σ	2, 3, 4, 5
r	the number of D system rewriting rules (RR)	2, 3, 4,..., 16
m	the max. length of the left-hand side of the RR	2, 3, 4, 5
n	the max. length of the right-hand side of the RR	2, 3, 4, 5
p	the length of the environment pattern	0, 1, 2,..., 64

biological basis. Since the developmental rules constitute the genotype, the environment is therefore the subject of evolution and there is no straightforward way how to separate it. (For the reason that the environment is sometimes understood as a medium which ordinarily influence the phenotype in some way, but is separated from it, e.g. atmosphere is necessary for the development of mammals, its quality affect the development of the phenotype, however, the atmosphere itself is not evolved in a sense of application of the genetic operators).

The system presented herein works with the rewriting rules containing the symbols (cells) for the construcion of the phenotype, i.e. no internal program is stored inside the cells. The system is thus viewed more "technically" – there is a program (the rewriting rules) over the cells which controls the development of the phenotype. The context of the development of the cells is determined by the context-sensitive rules which can be considered as an analogy to the conditional rules in the Winson's model. Moreover, there is an environment in our developmental system represented by a bit string which is considered as a separate part of the developmental system affecting the phenotype being developed (similarly to the atmosphere in case of the mammals development). So it is possible to "take" a particular environment (the bit string), "put" a phenotype into it ant let the phenotype develop within that environment. However, due to the potential complexity of the environment and the developmental system, it is very difficult to design a specific environment a priori for a particular system to perform a given kind of development. Therefore, the environment bit string represents a part of the genotype together with the developmental rules which is evolved by the genetic algorithm similarly to the Wilson's approach. Finally, there is a developmental algorithm (designed a priori), which represents the third part of the developmental system, determining rigorously the developmental process (i.e. how to apply the developmental rules, how the environment should interact with the phenotype in order to influence the development etc.).

As obvious, though the implementation of the two developmental systems is totally different, the difficult issues — especially the formation of the environment — are approached in a similar way. In addition, the goal of the systems is identical – to develop a linear cell structure.

Since the proposed formal model od development is intended to be general, it would be possible, naturally, to simulate the behavior of the Wilson's system by designing an appropriate developmental algorithm, rewriting rules and a proper interpretation of the symbols.

5.2 Learning Classifier Systems

In addition to the biological development, Wilson's work deals with the classifier systems. ZCS [27] and XCS [28] represent probaly the most well-known ones. A classifier system consists of the genetic algorithm for the evolutionary design of a set of condition-action rules or "classifiers", consisting of the symbols from the set $\{0, 1, \#\}$, which process the *messages* placed on the *message list* (an internal memory of the classifier system) and comming to the system from outside. The goal is to evolve classifiers to solve a given task.

In general, a classifier system performs development according to the rules designed by means of the genetic algorithm. There is an alphabet containing the elements of the rules, a king of environment and an algorithm determining the interaction of the elements (i.e. a developmental algorithm controling the operation of the classifier system). Therefore, it can be considered as a specific case of development which can be described using the formal system introduced herein.

6 Setup of the Evolutionary Developmental System

An evolutionary algorithm is utilized for designing the rewriting rules of the D system and a suitable environment for the development of the embryo in order to generate as complex strings as possible. In other words, we are going to evolve such rewriting rules of the D system and environment which will lead to the development of strings which are as little compressible as possible.

6.1 The Evolutionary Algorithm

A simple genetic algorithm was utilized for the design of rewriting rules and environment for the development. Since the chromosome contains two distinct entities (environment and rewriting rules), a co-evolutionary approach is concerned. Each rewriting rule is encoded as two strings which represent the left-hand side and right-hand side of the rule. A finite sequence of rewriting rules comprise the first part of the chromosome. The second part includes the environment pattern which is encoded also by the string. According to the parameters of the developmental system, the rewriting rules and environment are designed. Figure 1 shows a general structure of the chromosome. In the case of experiments when no environment is considered, the environment part $e_1 e_2 \dots e_p$ of the chromosome is empty and only the rewriting rules are evolved.

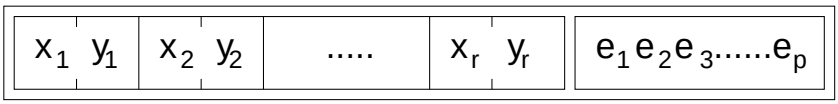

Figure 1: General structure of the GA chromosome. x_i, respective y_i denotes the left-hand side, respective right-hand side of the rewriting rule.

6.2 Genetic Operators

For the purposes of the developmental model presented in this paper, new genetic operators of crossover and mutation have been designed. These operators take into account the structure of the chromosome including the prescription in the form of rewriting rules and environment pattern and ensure correctness of the resulting chromosomes after execution of the operators. Consider the rewriting rule of the form $s_1 s_2 \dots s_m \rightarrow s'_1 s'_2 \dots s'_n$, where m, respective n denotes the maximal length (the number of symbols) of the left-hand side, respective right-hand side of the rewriting rule. The *mutation* operator has been designed in order to be able to randomly (1) alter a symbol in the left-hand side or right-hand side of the rule or (2) design the entire new left-hand side or right-hand side of the rule, and possibly change the number of symbols which the string is composed of or (3) change one symbol of the environment pattern. The *crossover* operator randomly selects a rewriting rule in both parent chromosomes and swaps these rewriting rules in the two offspring. The environment patterns are not crossed, i.e. only small chnages in the environment are allowed and mainly the rewriting rules are evolved. The principle of crossover operator is shown in Figure 2. *Selection* is performed using the tournament algorithm.

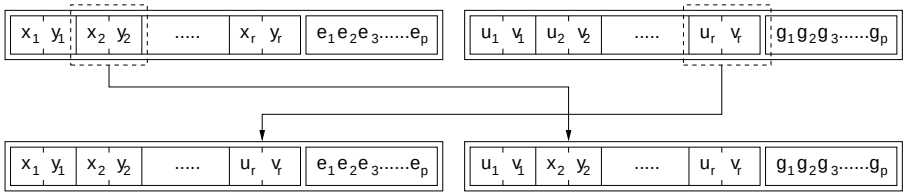

Figure 2: The principle of the crossover operator

6.3 Fitness Function

Each chromosome (the genotype) represents a prescription for the development of the embryo in the form of rewriting rules of D system and environment controlling the developmental process which is determined by the developmental algorithm. This algorithm is designed by the designer before starting the evolutionary process and is invariable during execution of the evolutionary process. Using the developmental algorithm, the genotype is mapped to a string (the phenotype) by performing 6 developmental steps starting in the embryonal stage, i.e. the developmental algorithm is run six times for each chromosome. The fitness value is computed as follows. The developed string is compressed using the LZ-based algorithm provided by the free *zlib* library [6]. The length (the number of bytes) of the compressed string equals the fitness value. Note that the compression algorithm appends some control-bytes into the compressed string. The fitness value (length) of the compressed string is considered without these control-bytes. Since complex strings exhibit low compressibility, it holds that the longer compressed string the higher fitness value. In theory, maximal fitness value of a string possessing the length k just equals k, i.e. the string is not compressible at all.

6.4 Experimental Setup

All the experiments were conducted using the following setup. The population contains 10 individuals (chromosomes). Each chromosome is mutated with the probability 0.95 by means of one randomly selected mutation method described in Section 6.2. Crossover is performed also with the probability 0.95. Individuals are selected using the tournament algorithm of base 4, i.e. four individuals are selected randomly from the population, the best two of them are crossed and mutated in order to pass into the next generation. Each chromosome performs 6 developmental steps and the developed string complexity is investigated using the LZ-based compression algorithm. In each experiment, 20000 generations are evolved after which the chromosome possessing the higher fitness value is recorded. For a particular settings of parameters of the developmental system, 100 independent runs of the evolutionary algorithm are executed.

7 The Impact of Environment on the Complexity of Generated Strings: Experimental results

An evolutionary algorithm has been used to find a suitable set of rewriting rules together with a proper environment pattern in which the system develops in order to generate as complex target strings as possible. The hypothesis is that the more complex environment is allowed the more complex object (string) emerges in case that a constant number of rewriting rules is utilized. On the contrary, if no environment is considered, then we expect that the more rewriting rules the more complex object emerge. Various configurations of parameters characterizing the developmental system have been considered in order to prove the hypohesis.

The complexity measurement has been considered from two basic points of view: (1) environment and (2) D system. While the former is intended for exploring the dependence of target object complexity on the allowable environment complexity, in the second case no environment is considered and the dependence of the complexity of target object on the number of rewriting rules of the D system is investigated. In general, the number of rewriting rules of the D system, respective the length of the environment pattern is interpreted as the amount of genetic information, respective the amount of information supplied to the system from "outside", i.e. from the environment.

The following parameters have been utilized in the experiments to set the properties of the developmental system. Complexity of the D system (i.e. maximal amount of genetic information) is determined by the values of parameters c, m and n denoting the number of symbols of the string alphabet Σ, maximal length of the left-hand side and maximal length of the right-hand side of rewriting rules respectively. For the purposes of this paper, we consider $c = m = n$. We are interested in values of those parameters in the range from 2 to 5. For each value, 100 independent experiments have been conducted in order to determine dependence of string complexity on (1) the maximal possible amount of information supplied by the environment (i.e. the length of environment pattern) and (2) amount of the information stored in the chromosome (i.e. the number of rewriting rules). The next sections describe both cases and the experimental results.

7.1 D system-based complexity measurement

Experiments have been conducted in which no environment has been considered ($e = \varepsilon$). Thus the developmental process depends on rewriting rules only whose application is handled by the developemtal algorithm. The dependence of the complexity of developed string on the number of rewriting rules of the D system has been investigated. Figure 3 shows the dependence of complexity of developed string on the number of rewriting rules of D system in the range from 2 to 16. Parameter settings of the D system are given in the caption of the figure.

7.2 Environment-based complexity measurement

This sort of experiments involves an external information that is supplied to the developmental system from the environment. Recall that the environment is composed of a binary pattern which is repeated periodically along the string being developed in order to forbid/permit application of rewriting rules at given positions of the string. Thus the string development depends not only on the rewriting rules but also on the environment surrounding the developmental system. We assume that the longer environment pattern the more complex rewriting regulation, i.e. more complex string could be developed. The length of environment pattern is considered in the range from 2 to 64. The simplest environment with pattern length 2 has the form $010101\ldots$ or $101010\ldots$. In general, pattern $00\ldots0$ is meaningless because it forbids application of any rule and thus prevents the string to develop. Similarly, the pattern $11\ldots1$ is meaningless as the behavior of the developmental system is identical with the case when no environment is considered ($e = \varepsilon$). This pattern actually permits the application of matching rules at any positions of the string being developed (there is no forbidding ability and hence no meaningful control). The number of rules of D system is set to 2 among all these experiments. Figure 4 shows the dependence of the developed string complexity on the number of symbols (length) of the environment pattern. Parameter settings of the D system are given in the caption of the figure.

7.3 Computational Effort

The experiments have been conducted using the standard PC (Pentium 1.6GHz, 512MB RAM). Two points of view can be considered for investigating the computational effort: (1) computational effort of the developmental process and (1) computational effort of the fitness calculation (evaluation of the chromosomes).

For the case (1), if the number of rewriting rules and the length of the left-hand side and right-hand side of the rewriting rules are high, then the developmental process is more time-consuming because many comparisons of symbols have to be performed in order to determine the matching rules. Moreover, if the number of symbols

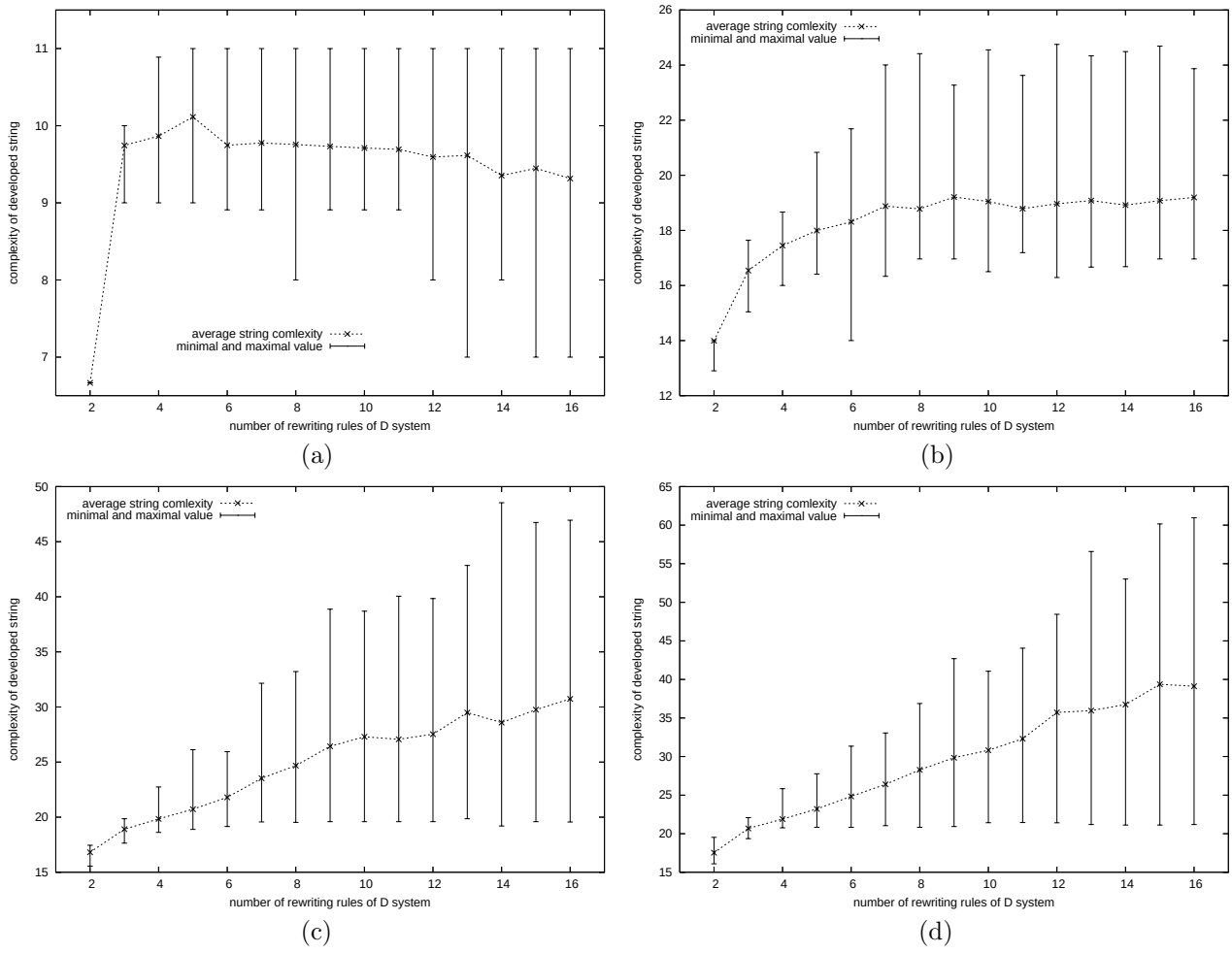

Figure 3: Dependence of the developed string complexity on the number of rewriting rules of D system for the configuration of developmental system determined by the number of symbols c, the maximal length m of the left-hand side of rewriting rules and the maximal length n of the right-hand side of rewriting rules: (a) $c = 2, m = 2, n = 2$, (b) $c = 3, m = 3, n = 3$, (c) $c = 4, m = 4, n = 4$, (d) $c = 5, m = 5, n = 5$; no environment is considered.

increases, the number of possible different left-hand sides increases and the developmental process becomes more time-consuming.

For the case (2), if the number of rewriting rules increases and the length of the right-hand side of the rewriting rules increases, longer strings can be developed through 6 developmental steps considered in the experiments and the longer string the more time-consuming evaluation process (compression of the string).

As obvious, the duration of an experiment depends on the configuration of the developmental system. The evolution of the simplest developmental system (the maximal length of the left-hand side $m = 2$, the maximal length of the right-hand side $n = 2$, the number of symbols $c = 2$, 2 rewriting rules) takes 12 seconds in average. On the contrary, the evolution of the more complex developmental system, for example, the maximal length of the left-hand side and right-hand side $m = n = 5$, the number of symbols $c = 5$, 16 rewriting rules, took approximately 70 seconds. Recall that 20000 generations of the evolutionary algorithm have been considered and 100 independent experiments have been conducted for each configuration of the developmental system.

For the D system-based complexity measurement, no environment has been considered, the number of symbols and the maximal length of the left-hand side and right-hand side of the rewriting rules have been considered in the range from 2 to 5 (recall that $c = m = n$ in the experiments). For each of those values, the number of rewriting rules has been considered in the range from 2 to 16 and for each of these configurations of the developmental system, 100 independent experiments have been conducted. In case of environment-based complexity measurement, the number of rewriting rules was set to $r = 2$ in all the experiments, c, m and n were considered in the range from 2 to 5, the length of environment pattern in the range from 2 to 64 was considered for each of the values of parameters c, m, n and for each of these configurations, 100 independent experiments were conducted. In total, more than 30000 independent experiments were performed.

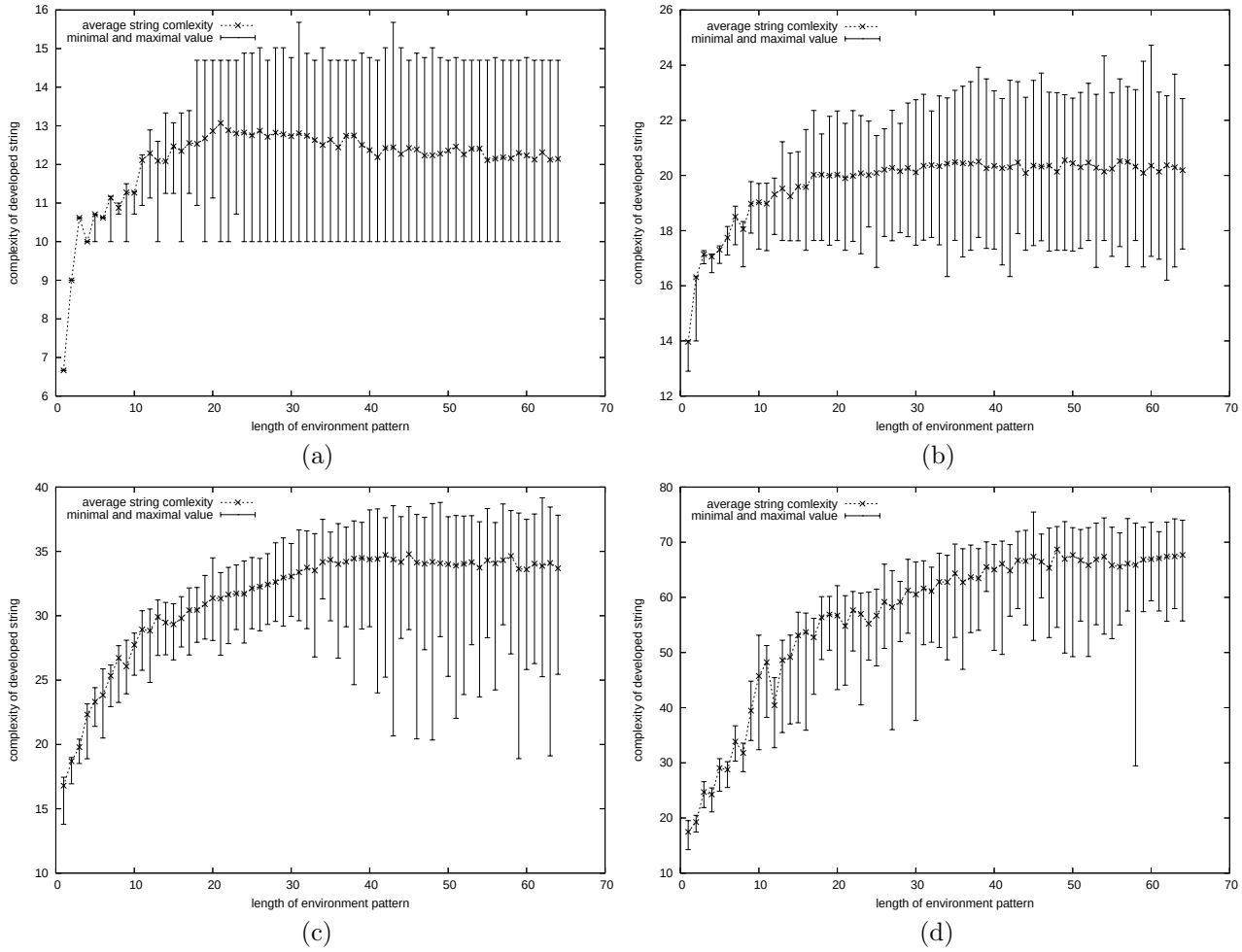

Figure 4: Dependence of the developed string complexity on the length of environment pattern for the configuration of developmental system determined by the number of symbols c, the maximal length m of the left-hand side of rewriting rules, the maximal length n of the right-hand side of rewriting rules and the number of rewriting rules $r = 2$: (a) $c = 2, m = 2, n = 2$, (b) $c = 3, m = 3, n = 3$, (c) $c = 4, m = 4, n = 4$, (d) $c = 5, m = 5, n = 5$

8 Discussion

The hypothesis that should be proven in this paper is that if an environment is considered during the developmental phase in an evolutionary design system, then the evolved genotype (a prescription for the construction of a target object) is able to generate (develop) more complex objects in comparison with the case when no environment is considered (i.e. the more external information is supplied to the developmental system the more complex objects emerge). Similarly, if no environment is considered, then the more genetic information is included in the chromosome the more complex objects emerge. The amount of external information has been measured as the length of a pattern that is repeated periodically along the developing object, constituting the environment. The amount of the genetic information included in the chromosome has been measured as the number of rewriting rules of the D system.

Figure 3, respective 4 shows the dependence of the developed string complexity on the number of rewriting rules, respective on the length of the pattern constituting the environment. The graphs depict the minimal, maximal and average fitness values (string complexities) of the best chromosomes developed in 100 independent runs of the EA for each value of the parameter on which the dependence of the developed string complexity has been measured. As obvious, the complexity of the developed strings increases with increasing the number of rewriting rules. Similar dependence of the string complexity on the length of the environment pattern has been observed. For the D system-based complexity measurement, a threshold value of the number of rewriting rules can be found out indicating that the string complexity has reached a maximal value and for the values of the number of rewriting rules larger than the threshold value the string complexity does not increase. Similarly, for the environment-based complexity measurement, a threshold value of the length of the environment pattern can be found out indicating that the string complexity has reached a maximal value and for the values of the

length of the environment pattern larger than the threshold value the string complexity does not increase. For example, in the case of D system-based complexity measurement depicted in Figure 3b the treshold value of the number of rewriting rules $r_T = 9$ or in the case of environment-based complexity measurement depicted in Figure 4c the threshold value of length of the environment pattern $p_T = 37$. By comparing the results of D system-based and environment-based complexity measurement, it is obvious that the environment causes much more complex strings that can be developed in comparison with the case when no environment is considered. Therefore, the hypothesis postulated in this paper has been proven.

The most simplest variant of the environment has been considered in the experiments. Only two symbols of the environment alphabet have been utilized in order to control the developmental process (in particular, to forbid/permit the application of rewriting rules during the development). In general, however, the environment can consist of theoretically arbitrary number of arbitrary symbols that even may but need not be included in the alphabet of D system. Moreover, the symbols can be interpreted in many ways and arbitrary forms of control of the developmental process can be considered (e.g. forbidding/permitting the application of rules during develpoment, parametrizing the properties of the developing object, switching the function of the developing object as demonstrated in polymorphic circuits [24, 3] etc.). Similarly, the developing object (string) can represent arbitrary entity of the real world (e.g. computer programs, digital ciruits, image information etc.).

9 Conclusions

In the paper, a formal concept of a general developmental system has been introduced. A hypothesis has been formulated about the impact of an external information supplied to the developmental system from the environment on the complexity of the developed objects. Genetic algorithm has been utilized to find suitable rewriting rules and environment pattern for generating as complex strings as possible. Dependence of the developed string complexity on the number of rewriting rules and on the length of the environment pattern has been investigated. The experiments have shown that there is a threshold value related to the number of rewriting rules and the length of the environment pattern for which the complexity of strings generated by the evolved developmental system reaches a maximal value and for greater values of the number of rewriting rules or length of the environment pattern than the threshold value the string complexity does not increase. The experiments have confirmed the validity of the proposed hypothesis.

The developmental system has been designed from an abstract point of view, i.e. the developmental process only performed growth of a linear structure (string) with the objective of as high complexity as possible. The strings also have been considered only as sequences of symbols without a particular interpretation. Therefore, the next research will be devoted to the potential application of the proposed model. In our previous research, some approaches have been presented demonstrating successful utilization of application-specific instruction-based developmental systems in the evolutionary design of digital circuits [22, 3]. However, no external mechanism (i.e. a kind of environment) has bees utilized to control the developmental process or influence the developing object. These issues form the basic hypotheses for our future research.

Acknowledgement: This work was partially supported by the Grant Agency of the Czech Republic under contract No. GP103/10/1517 *Natural Computing on Unconventional Platforms*, the Grant Fund (GRAFO) of Brno University of Technology (BUT), the internal BUT research project No. FIT-S-10-1 and the Research Plan No. MSM 0021630528 *Security-Oriented Research in Information Technology.*

References

[1] B. Alberts et al. *Essential Cell Biology, 2nd edition.* Garland Science/Taylor & Francis Group, 2003.

[2] P. J. Bentley and S. Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Proc. of the Genetic and Evolutionary Computation Conference*, volume 1, pages 35–43, San Francisco CA, USA, 1999. Morgan Kaufmann.

[3] M. Bidlo and L. Sekanina. Providing information from the environment for growing electronic circuits through polymorphic gates. In *Proc. of Genetic and Evolutionary Computation Conference – Workshops 2005*, pages 242–248. Association for Computing Machinery, 2005.

[4] J. Dassow and G. Paun. *Regulated Rewriting in Formal Language Theory.* Springer–Verlag, New York, US, 1990.

[5] H. de Garis. Artificial embryology: The genetic programming of an artificial embryo. In *Dynamic, Genetic and Chaotic Programming.* Wiley, 1992.

[6] J. Gilly and M. Adler. Zlib – a massively spiffy yet delicately unobtrusive compression library. http://www.zlib.net, Apr 2010.

[7] T. G. W. Gordon. Exploring models of development for evolutionary circuit design. In *Proc. of the 2003 Congress on Evolutionary Computation, CEC 2003*, pages 2050–2057. IEEE Press, 2003.

[8] T. G. W. Gordon. Exploiting development to enhance the scalability of hardware evolution, PhD thesis. Department of Computer Science, University College London, 2005.

[9] P. C. Haddow, G. Tufte, and P. van Remortel. Shrinking the genotype: L-systems for ehw? In *Proc. of the 4th International Conference on Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, vol. 2210*, pages 128–139. Springer-Verlag, 2001.

[10] M. Hartmann, P. K. Lehre, and P. C. Haddow. Evolved digital circuits and genome complexity. In *Proc. of the 2005 NASA/DoD Conference on Evolvable Hardware*, pages 79–86. IEEE Press, 2005.

[11] G. S. Hornby and J. B. Pollack. The advantages of generative grammatical encodings for physical design. In *Proc. of the 2001 Congress on Evolutionary Computation*, pages 600–607. IEEE Press, 2001.

[12] J. D. Lohn et al. An evolved antenna for deployment on nasa's space technology 5 mission. In *Proc. of Genetic Programming Theory Practice 2004 Workshop, GPTP 2004*, 2004.

[13] J. R. Koza and F.H. Bennett and D. Andre and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, 1999.

[14] S. Kumar. Investigating computational models of development for the construction of shape and form, PhD thesis. Department of Computer Science, University College London, 2004.

[15] S. Kumar and P. J. Bentley. Implicit evolvability: An investigation into the evolvability of an embryogeny. In *Proc. of the Genetic and Evolutionary Computation Conference – Late-Breaking Papers*, San Francisco CA, USA, 2000. Morgan Kaufmann.

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

[17] M. Li and P. M. B. Vitanyi. An introduction to kolmogorov complexity and its applications. New York US, 1997. Springer-Verlag.

[18] A. Lindenmayer. Mathematical models for cellular interaction in development, parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[19] J. F. Miller. Evolving developmental programs for adaptation, morphogenesis and self-repair. In *Advances in Artificial Life. 7th European Conference on Artificial Life, Lecture Notes in Artificial Intelligence, volume 2801*, pages 256–265, Dortmund DE, 2003. Springer.

[20] G. Rozenberg and A. Salomaa (eds.). *Handbook of Formal Languages, volume 1*. Springer-Verlag, 1997.

[21] S. Kumar and P. J. Bentley (eds.). *On Growth, Form and Computers*. Elsevier Academic Press, 2003.

[22] L. Sekanina and M. Bidlo. Evolutionary design of arbitrarily large sorting networks using development. *Genetic Programming and Evolvable Machines*, 6(3):319–347, 2005.

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

[24] A. Stoica, R. S. Zebulum, and D. Keymeulen. Polymorphic electronics. In *Proc. of International Conference on Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, volume 2210*, pages 291–302. Springer-Verlag, 2001.

[25] A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 237(641):37–72, 1952.

[26] S. W. Wilson. The genetic algorithm and biological development. In *Proc. of the Second International Conference on Genetic Algorithms*, pages 247–251, Hillsdale, New Jersey, 1987. Lawrence Erlbaum Associates.

[27] S. W. Wilson. A zeroth-level classifier system. *Evolutionary Computation*, 2:1–18, 1994.

[28] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3:149–175, 1995.