Evolutionary Development of Growing Generic Sorting Networks by Means of Rewriting Systems

Michal Bidlo and Michal Dobeš

***Abstract*—This paper presents an evolutionary developmental method for the design of arbitrarily growing sorting networks. The developmental model is based on a parallel rewriting system (a grammar) that is specified by an alphabet, an initial string (an axiom), and a set of rewriting rules. The rewriting process iteratively expands the axiom in order to develop more complex strings during a series of development steps (i.e., derivations in the grammar). A mapping function is introduced that allows for converting the strings onto comparator structures—building blocks of sorting networks. The construction of the networks is performed in such a way that a given (initial) sorting network grows progressively by adding further building blocks within each development step. For a given (fixed) alphabet, the axiom together with the rewriting rules themselves are the subjects of the evolutionary search. It will be shown that suitable grammars can be evolved for the construction of arbitrarily large sorting networks that grow with various given sizes of development steps. Moreover, the resulting networks exhibit significantly better properties (the number of comparators and delay) in comparison with those obtained by means of similar existing methods.**

***Index Terms*—Development, genetic algorithm (GA), rewriting system, scalability, sorting network.**

I. Introduction

Evolutionary design has become a widely used and successful concept for solving various tasks. This concept uses evolutionary algorithms (EAs) in combination with a suitable representation and domain-specific knowledge of the problem to be solved (supplied by the designer) in order to automatically generate candidate solutions, given some criteria to be preferred and optimized during many iterations (generations) of the EA. One of the most remarkable advantages of EA-based design approaches is that innovative solutions can be produced that are beyond the scope of conventional engineering methods. Bentley surveyed various methods and applications of this field in [1] and [2]. The reasons for its usefulness can be summarized as follows [1].

1) Evolution is a good, general-purpose problem solver.
2) Evolution and the human design process share many similar characteristics.
3) The most successful and remarkable designs known to mankind were created by natural evolution, the inspiration for EAs.

On the basis of these statements, the following general goal can be formulated when applying the evolutionary design in the engineering domain: to produce original, innovative, efficient, and useful solutions to complex problems that can be created with minimal effort and domain knowledge of a designer. Therefore, the main challenge for the designer is to choose a suitable EA (together with the representation, fitness function, etc.) in order to *automatically* search for new solutions of a given problem rather than to perform the creative design manually.

For example, evolutionary design has succeeded (and in some cases outperformed human designers) in solving hard problems for example, designing analogue circuits (e.g., [3]–[6]), digital circuits (e.g., [7] and [8]), antennas (e.g., [9]–[11]), image filters (e.g., [12]), classifier and control systems (e.g., [13]), performing on-chip evolutionary design using field-programmable gate array (FPGA) (e.g., [14]–[17]), or evolution-in-materio using liquid crystals or carbon nanotubes (e.g., [18] and [19]).

Although the functioning of systems produced by evolutionary design can often be viewed as sophisticated and difficult to understand, the innovative and most successful designs have mostly been obtained for relatively "simple" systems—meaning simple in size (i.e., composed of less components rather than very extensive systems). This issue corresponds to the fact that large designs require long chromosomes, i.e., the search space becomes bigger and more difficult to be effectively explored by EA, which is referred to as the *problem of scale of the representation* (as pointed out by Torresen in [20]). Moreover, in the case of designing digital circuits, evaluation time grows exponentially with an increasing number of inputs, which is known as the *problem of scale of the fitness calculation*. In order to overcome the scaling problems in the evolutionary circuit design, researchers have introduced various concepts, the most important of which are the following: evolution at the functional level (introduced by Murakawa *et al.* [21]),

Manuscript received July 24, 2018; revised November 12, 2018, January 30, 2019, and May 10, 2019; accepted May 15, 2019. Date of publication May 22, 2019; date of current version March 31, 2020. This work was supported in part by the Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) through Project "IT4Innovations Excellence in Science" under Grant LQ1602, and in part by the IT4Innovations Infrastructure from the Large Infrastructures for Research, Experimental Development and Innovations through Project "IT4Innovations National Supercomputing Center" under Grant LM2015070. *(Corresponding author: Michal Bidlo.)*

The authors are with the Faculty of Information Technology, Centre of Excellence IT4Innovations, Brno University of Technology, 61266 Brno, Czechia (e-mail: bidlom@fit.vutbr.cz; michal.dobes.jr@gmail.com).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2019.2918212

1089-778X © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

incremental evolution (proposed by Toressen [20]) and development (for the circuit evolution introduced by Haddow and Tufte [22] and Gordon and Bentley [23]). Recently, Vašíček and Sekanina [24] introduced a formal verification-based approach to eliminate the problem of scale of the fitness evaluation of large digital circuits. In this paper, we will deal with development applied to the problem of evolutionary design of growing sorting networks.

A. Development for Evolutionary Design

In nature, development embodies the processes that occur during the whole life of an organism. Development is essentially the emergence of organized structures from an initially very simple group of cells [25]. In EAs, the concept of *computational development* [26] allows for the evolution of nontrivial indirect representations (prescriptions represented by genotypes) according to which target objects (solutions to a problem—phenotypes) can be constructed. The importance of development in the field of evolutionary design is that some features can reasonably be achieved only by employing development. For example, shrinking genotypes and reducing the search space, adaptation, self-organization, emergent behavior, growth, scalability, or generic design usually cannot be achieved using direct representations. Kumar surveyed some basic methods for computational development in [26, Ch. 2].

One of the techniques suitable for performing the computational development is a grammar-based approach in which two basic scenarios have been applied.

1) The development is controlled by a grammar with *given rewriting rules* and the target solutions develop as (or according to) evolving "programs" which are derived using the grammar [27]. This concept, currently known as *grammatical evolution*, has gained popularity in recent years and nowadays provides many variants for solving problems in different areas (e.g., see a survey in [28]). Specifically, for example, grammatical evolution has been applied to predict functions [29], solve architectural and engineering design problems [30], predict the stock price development [31], or optimize heterogeneous cellular networks [32].
2) In a more general way, the form of the rewriting rules may be the *subject of evolution*, the development is performed as a sequence of derivations by means of such (evolving) rules in order to construct a target object.

In this paper, the latter approach will be applied in order to design growing (scalable) sorting networks using a specific type of grammar called *Lindenmayer systems* (currently known as *L-systems*). Lindenmayer introduced the concept of *L*-systems in [33]. *L*-systems represent a class of formal grammars that allow several parts of a string to be rewritten *in parallel* during a single rewriting step (in general, using different rewriting rules). This feature makes *L*-systems particularly useful for modeling and simulation of natural organisms, the development of which is inherently parallel. Hence, *L*-systems have mostly been used for studying various aspects of biological development and for modeling complex systems in general (e.g., graphical modeling of plants [34] or evolution of virtual creatures [35]). However, successful applications of *L*-systems also exist in other areas. For example, a rewriting developmental (neuro) system was investigated by Kitano [36]. Later, Boers and Kuiper [37] utilized *L*-systems to create the architecture of feed-forward artificial neural networks. 3-D mechanical objects have been designed by Hornby and Pollack [38] by evolving a variant of *L*-system for their genotype–phenotype mapping. Haddow *et al.* [39] applied *L*-systems in the problem of development of digital circuits using the concept of extrinsic evolution. Escuela *et al.* [40] evolved *L*-systems as an inference procedure for folded structures on simple lattice models. Ashlock *et al.* [41], [42] studied simultaneous evolution of *L*-system rules and their interpretation in order to solve various rendering problems. Beaumont and Stepney [43] applied grammatical evolution of *L*-systems in order to investigate the effect of elitism, and the form of the underlying grammar. Campos *et al.* [44] used *L*-systems for evolutionary generation of neural network architectures which are capable of performing various tasks.

B. Motivation and Goals

Although the development is traditionally viewed as a technique that should allow us to achieve a certain degree of complexity of evolved phenotypes (compared to the "classical" evolutionary design), some studies have shown that the obtained solutions exhibit complexity similar to those generated without development (e.g., [22], [23], and [45]). Nevertheless, the complexity may lie in various aspects of the solutions, depending on application—for example, the size of a circuit regarding its number of inputs/outputs, the number of low-level elements (e.g., transistors or logic gates), the ability of adaptation, self repair, etc. However, apart from several studies, where the scalability problem has been primarily taken into account (e.g., [38] and [46]–[50]), a truly *generic* evolutionary design in which the target solution can *grow* and is actually able to solve a general instance of a problem represents a rather rare case. However, we believe that this concept is worth further investigation because several studies have shown the potential of evolutionary design in order to discover innovative solutions in various areas (e.g., instruction-based development of generic sorting networks [46], [47] or evolution of transition functions for generic square calculations in cellular automata [48], [51]). It is important to note that in order to achieve such results, a suitable developmental representation needs to be proposed in combination with a proper objective function and EA, which represents a nontrivial task.

The goal of this paper is to propose a developmental scheme based on a parallel rewriting system (a grammar) that is specified by an alphabet, an initial string (an axiom) and a set of rewriting rules. The rewriting process iteratively expands the axiom in order to develop more complex strings during a series of development steps (i.e., derivations in the grammar). A mapping function is introduced that allows converting the strings onto comparator structures—building blocks of sorting networks. The construction of the networks is performed in such a way that a given (initial) sorting network grows progressively by adding further building blocks within each

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

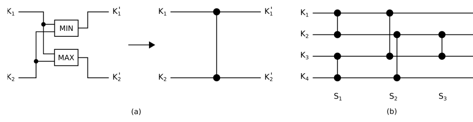

Fig. 1. (a) Compare-swap component and its schematic symbol and (b) example of a 4-input sorting network.

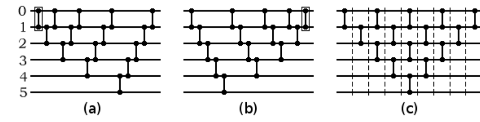

Fig. 2. Sorting network created by (a) insertion principle, (b) selection principle (in both cases from a 2-input network marked by a rectangle). Both networks are equivalent and exhibit a structure shown in (c) with the parallel layers separated by vertical dashed lines.

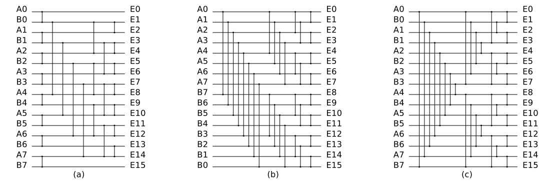

Fig. 3. Examples of sorting networks created by Batcher's algorithms for merging two 8-element sorted sequences into a sorted 16-element output sequence [56]. (a) Odd-even merge sort, (b) original bitonic merge sort, and (c) modified bitonic merge sort.

development step. For a given (fixed) alphabet, the rewriting rules themselves together with the axiom are the subjects of the evolutionary search. It will be shown that suitable grammars can be evolved for the construction of arbitrarily large sorting networks that grow with various given sizes of development steps. Moreover, the resulting networks exhibit significantly better properties (the number of comparators and delay) in comparison with those obtained by means of similar existing methods.

II. Sorting Networks and Their Design

The concept of sorting networks was introduced in 1954; Knuth summarized its history and some basic principles in [52]. A sorting network is defined as a sequence of compare-swap components (called comparators) the arrangement of which in the sorting network depends only on the number of elements to be sorted (i.e., inputs of the network), not on the values of the elements. Therefore, sorting networks are said to be *data-independent*. This means that the structure of a sorting network of n inputs is fixed for the given n. Such a feature is especially suitable for parallel processing and hardware implementation. A comparator unit contains two inputs (K_1, K_2) and two outputs (K'_1, K'_2). The comparator compares and possibly exchanges the two inputs, so that the outputs satisfy $K'_1 \leq K'_2$ for arbitrary values of K_1, K_2. The structure of a comparator, its symbolic representation and an example of a 4-input sorting network are shown in Fig. 1(a)–(c), respectively.

The delay and the number of comparators represent two crucial parameters of any sorting network. The delay of a sorting network is the minimum number of groups of comparators where all comparators inside a group process independent inputs. Therefore, the delay at the hardware level determines the longest combinational path of the network because the comparators inside each group can be evaluated in parallel while the evaluation of the groups is sequential. Note that the pipelined implementation is not considered for calculating the delay. Both the number of comparators and the delay increase with the size of the sorting network.

For example, the sorting network in Fig. 1(b) has delay 3—see the groups of comparators marked by rounded rectangles and denoted as S_1, S_2, and S_3. Note that the groups of independent comparators are called *parallel layers*. The aim of designing sorting networks is to minimize the number of comparators, delay or both parameters.

In order to determine whether an n-input sorting network is said to be *valid* (i.e., is able to sort any input sequence), the so-called *zero–one* principle can be applied. The zero–one principle states that if a sorting network correctly sorts all 2^n binary input vectors, then it sorts correctly any sequence of arbitrary values [52]. This allows us to reduce the number of test vectors from $n!$ to 2^n. Even so, the complete test of sorting networks in this way is not feasible for a large n because the number of test vectors grows exponentially with n. Furthermore, it is generally impossible to obtain a correct solution if only a subset of test vectors is utilized for an evaluation of the sorting network [53]. However, there are other techniques which allow us to further shorten the evaluation time substantially. For example, Bundala and Závodný [54] applied an approach based on formal verification using an SAT solver.

A. Conventional Sorting Network Design

Although the design of a sorting network is usually performed for a fixed number of inputs (with the aim of optimizing the parameters of the specific network), several generic algorithms are known from the literature which can be used for the construction of arbitrarily large sorting networks.

One class of such algorithms uses a method in which a larger sorting network *grows* from a valid smaller network by adding a suitable arrangement of comparators. Consider a valid sorting network with n inputs. A widely known insertion principle (called *straight insertion sort*) or selection principle (denominated as *bubble sort*) allows us to create a valid $n+1$-input network from the smaller one by adding n comparators (see Fig. 2). Although such networks are, in fact, the least efficient compared to those created for a fixed n, these approaches probably represent the simplest general design principles of sorting networks.

Another class of algorithms performs *merging* of two already sorted sequences into a single sorted sequence (instead of growing a target sorting network). There are two well known approaches for constructing sorting networks this way: 1) the odd–even merge sort and 2) bitonic merge sort (proposed by Batcher [55] with a further analysis provided in [56]). Unlike the growing sorting networks, the Batcher's algorithms take two already sorted sequences of the same length, A_i and B_i (for $i = 1, 2, \ldots 2^{k-1}$), and merge them into a single sorted sequence E_j (for $j = 1, 2, \ldots 2^k$). Fig. 3 illustrates this principle which belongs to the most efficient techniques for creating variable size sorting networks.

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

Various other approaches have been recently proposed to synthesize or optimize sorting networks. For example, Schiller proposed an agglomeration law for sorting networks, the purpose of which was to better fit the given algorithm and the platform which it runs on. In the case of sorting networks, the agglomeration groups, the input data and the original (sorting) algorithm is generalized to work on the agglomerated input while keeping its original structure. This results in a new access opportunity of sorting networks well-suited for efficient parallelization on modern multicore computers, computer networks or GPGPU [57]. Zuluaga *et al.* [58] presented a domain-specific language and compiler to automatically generate hardware implementations of sorting networks which have reduced the area and are optimized for latency or throughput. For this purpose, a special hardware structure called streaming sorting networks was proposed which allowed us to achieve improved cost-performance tradeoffs of resulting solutions [58], [59]. Sklyarov *et al.* designed an FPGA-based accelerator for parallel data sorting. The architecture is based on three steps: 1) the initial data set is decomposed into subsets that are individually sorted; 2) several subsets are joined to form bigger sorted subsets; and 3) finally, these subsets are merged in order to produce the final sorted set. The proposed circuits permit the number of clock cycles per data item to be significantly reduced, thus optimizing the entire sorting process [60]. Bundala *et al.* analyzed optimality of sorting networks and proved depth limits for instances with 9–16 inputs (an open problem since the publication of the first edition of Knuth's work [52] in 1968). Their approach combines symmetry breaking and Boolean satisfiability using an SAT solver [54], [61], [62]. Codish *et al.* [63] studied some properties of the front and back parts of sorting networks, and illustrated their utility in the search for new bounds of optimal sorting networks. In particular, new solutions were presented for 17–20 inputs in [64]. Note that in addition to an extensive theoretical analysis of appropriate parts of sorting networks the authors also applied a simple EA to help optimize prefixes of the resulting solutions.

B. Evolutionary Sorting Network Design

Designing efficient sorting networks represents a difficult combinatorial optimization problem, especially for higher a number of inputs. Although some generic design techniques exist (see Section II-A), sorting networks designed by means of them are generally not optimal. This means that solving the problem of an optimal sorting network design requires finding the appropriate arrangements of comparators for every given number of inputs. Several works have been published that attempt to solve this task by means of various unconventional techniques, especially using EAs. In this section, some of the relevant works regarding the evolutionary design of both generic sorting networks and those designed for a fixed number of inputs are mentioned.

Although the insertion and selection principles, mentioned in Section II-A, enable a simple design of sorting networks for an arbitrary number of inputs (even by extending—growing—the existing optimal networks), the resulting solutions are not cost- or delay-efficient. This approach appends a certain structure of comparators to an existing valid n-input network in order to create a valid $n + 1$-input network, which may be considered a single step of growth (or development) of the sorting network. In [46] and [47], an EA was applied to grow $n+2$-input networks from n-input networks, i.e., a *single* development step was found to be sufficient in order to extend (grow) the network by two inputs. It was shown that if a more complex comparator arrangement (discovered by evolution) is appended to the growing network during this single step, then both the total number of comparators and the delay of the resulting networks can be reduced substantially in comparison with the insertion or selection principle. Note that this observation represents the main idea of the hypothesis formulated in Section I-B.

Some advances can be observed regarding the evolution of fixed-size sorting networks. Some of the structures (re)discovered in the past by means of evolution still represent state-of-the-art solutions of sorting networks for a given number of inputs. After only mentioning some of (now classical) works in which the artificial evolution was involved for the first time (e.g., [16] and [65]–[69]), the following summary focuses on relevant publications from recent years. In 2005, Choi and Moon [70] published a graph-theoretical approach that allows for the introduction of a repair heuristic into genetic algorithm (GA) for strong local optimization of sorting network structures. Their approach enabled the rediscovery of some of the best 16-input networks known at that time using a common (single-processor) PC, which was a result comparable to those obtained using supercomputers. Graham *et al.* [71] proposed a statistical analysis regarding parameter tuning of a GA in combination with specific heuristic functions in the problem of evolving sorting networks. Kubalík published an iterative algorithm (called POEMS) that seeks the best variation of a sorting network (for up to 16 inputs) in each iteration using so-called hypermutations. A hypermutation is a variation in the network structure, the discovery of which is performed using a GA. The POEMS approach showed an ability to outperform traditional mutation-based algorithms based on evolution strategies [72]. Coles used a concept of filters—a fixed sequences of comparators at the beginning of the sorting network—and its extension from smaller to larger networks using a stochastic process [73]. The filter should sort as many input sequences as possible and further improve the optimization of the entire sorting network. For example, a filter of a 9-input network was extended to 25 inputs, which allowed him to attain the best known bound (the number of comparators) for the network of this size. Valsalam and Miikkulainen [74] proposed an symmetry and evolution-based network sort optimization approach (SENSO) which utilizes the symmetry of the problem to decompose the minimization goal into subgoals that are easier to solve. Their method allowed the authors to improve various sorting network structures with up to 23 inputs. López-Ramírez and Cruz-Cortés [75] used a method based on an artificial immune system combined with a local search strategy to produce new optimal sorting networks for 9–15 inputs.

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

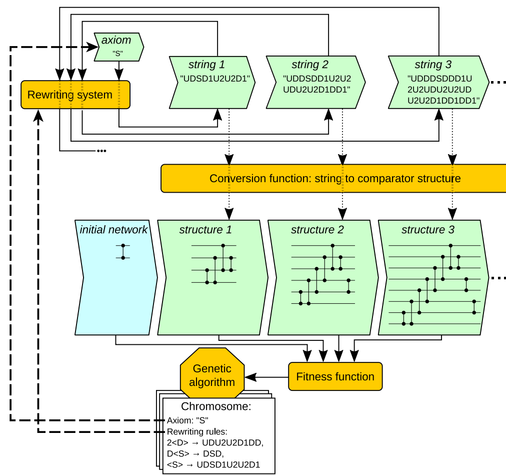

Fig. 4. Concept of the method proposed for the evolutionary development of generic sorting networks by means of rewriting systems. Note that the chromosome at the bottom of the figure shows an *example* of the axiom (S) and rewriting rules, both are the subject of an evolutionary search.

III. Evolutionary Development of Sorting Networks

The objective of this paper is to propose an application-specific developmental scheme for a GA, which allows us to design arbitrarily growing sorting networks. It can be assumed that a suitable representation of sorting networks for the GA in combination with the proposed developmental scheme may provide some innovative results in comparison with relevant methods published in the literature. The developmental scheme is based on a rewriting system, i.e., a grammar in which an iterative application of suitable rewriting rules on a string from the previous iteration (or on an initial string in case of the first iteration) produces more complex strings that encode the growing sorting networks. Remembering that the conventional evolutionary design of sorting networks typically treats fixed-size networks, the application of the evolutionary development for the design of generic (growing) sorting networks represents a rare case.

A. Evolutionary Developmental System

The scheme proposed for the evolutionary development of sorting networks is illustrated in Fig. 4. A sorting network is represented by a text string consisting of specific symbols whose interpretation will be described in Section III-B. A rewriting system (which will be described in Section III-C) is used to iteratively transform the strings in order to develop larger sorting networks—see the *Rewriting system* block in Fig. 4. An initial string (or *axiom*) and a set of rewriting rules represent the main ingredients of the rewriting system and both are generated by the GA—see the *Genetic algorithm* block in Fig. 4 that will be described in Section III-D. In order to perform the development of sorting networks, a simple valid sorting network is specified that will grow during the development (see the *initial network* block in Fig. 4). Specifically, a 2-input single-comparator initial network will be used in this paper. The development works as follows. The application of rewriting rules on the axiom generates a more complex string that encodes a comparator structure used for extending the initial network (see the *string 1* and *structure 1* blocks in Fig. 4). If more developmental steps are applied iteratively on the current string, more complex strings emerge that encode larger comparator structures (see the further *string* and *structure* blocks in Fig. 4). A single application of rewriting rules that generates the next string is interpreted as a *developmental step*. In our approach, both the axiom and the rewriting rules are the subject of evolution. This means that the axiom and rules (encoded in a single GA chromosome) represent a candidate recipe according to which the sorting networks can be developed.

B. Mapping of Strings to Sorting Networks

The developmental process utilizes a conversion function that maps the strings to sorting networks (see the *conversion function* block in Fig. 4). The following scheme is used to convert the strings from the rewriting system to the structures of comparators.

At the beginning of the developmental process, a wire pointer i will be introduced with an initial value of 0 (i.e., pointing to the top wire of the network to be created), the number of inputs of the initial network w_0 will be specified, a constant Δ will be chosen denoting the size of the developmental step and a step counter t will be initialized to 1. The rewriting system works with the alphabet $\Sigma = \{U, D, 1, 2, 3, 4, S, a, b, c, d\}$ the symbols of which have the following interpretation.

The symbol U represents the command *move the pointer up one wire if possible* (i.e., if $i > 0$ then $i = i - 1$). The symbol D represents the command *move the pointer down one wire if possible* (i.e., if $i < w - 1$ then $i = i + 1$). A digit $d \in \{1, 2, 3, 4\} \subset \Sigma$ represents a command *make a comparator* $(i, i + d)$. This implies that a comparator may compare values across at most 4 "spaces" between wires of the network. We chose to limit the size of comparators this way because larger values induced extensive design space in which the evolution failed to discover working solutions. On the basis of our initial experiments, we also limited the maximal size of the developmental step considered in our experiments to 4. The remaining alphabet symbols $\{S, a, b, c, d\} \subset \Sigma$ are interpreted as no-operation commands. An arbitrary nonempty string $V_x \in \Sigma^*$ is processed left to right in order to be transformed to a structure of comparators according to the aforementioned interpretation. By performing the tth developmental step, the number of inputs of the sorting network created in this step will be determined as $w_t = w_0 + \Delta \cdot t$. After each developmental step, the time will be increased ($t = t + 1$) and the wire pointer i will be reset to 0.

For example, let us consider the number of inputs of the initial sorting network to be $w_0 = 2$ and the size of the developmental step $\Delta = 2$. By performing the first developmental step, the axiom is rewritten to UDSD1U2U2D1, which is interpreted as an arrangement of comparators (see the *string 1* and *structure 1* blocks in Fig. 4). This structure extends the initial sorting network into a larger instance with $w_1 = w_0 + \Delta = 2 + 2 = 4$ inputs. Fig. 5 (to be read line

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

Symbol	i	Action	Comparator structure
U	0	Not applicable	
D	0	$i = i + 1 = 1$	
S	1	No operation	
D	1	$i = i + 1 = 2$	
1	2	$(2, 3)$	$(2, 3)$
U	2	$i = i - 1 = 1$	$(2, 3)$
2	1	$(1, 3)$	$(2, 3)(1, 3)$
U	1	$i = i - 1 = 0$	$(2, 3)(1, 3)$
2	0	$(0, 2)$	$(2, 3)(1, 3)(0, 2)$
D	0	$i = i + 1 = 1$	$(2, 3)(1, 3)(0, 2)$
1	1	$(1, 2)$	$(2, 3)(1, 3)(0, 2)(1, 2)$

Fig. 5. Example of a process transforming the string UDSD1U2U2D1 to a sequence of comparators of a 4-input comparator structure. Note that the action for the first symbol (U) is not applicable because $i = 0$.

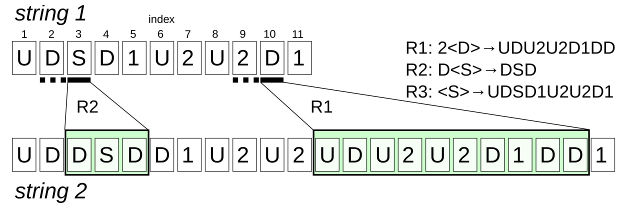

Fig. 6. Example of rewriting the *string 1* to the *string 2* using three MDIL-system-based rewriting rules (all shown in Fig. 4).

by line from top to bottom) shows the interpretation of the string UDSD1U2U2D1, the actions to be performed and the structure of comparators to be generated. After finishing this step, $t = t + 1 = 1$ and the wire pointer $i = 0$.

For the second developmental step (represented by the *string 2* and *structure 2* blocks in Fig. 4), the width is determined as $w_2 = w_0 + \Delta \cdot t = 2 + 2 * 2 = 6$, the index i is reset to 0 and the structure of comparators is generated according to the *string 2*. Since the sorting network is intended to grow during each developmental step, it is important to note that the sorting network created within the given developmental steps includes all the comparators generated during previous steps. This means that the 6-input network from the second step is made by joining *structure 1* and *structure 2* from Fig. 4, the 8-input network from the third developmental step arises by joining the *structure 3* to the previously created 6-input network, etc.

C. Rewriting System

The rewriting system was designed on the basis of so-called IL-systems which were studied in detail in [76]. An IL-system is a parallel context-sensitive rewriting system with rewriting rules of the form $V_L\langle V\rangle V_R \rightarrow V_T$, where $V_L, V_R, V_T \in \Sigma^*$, $V \in \Sigma$. Let us denominate the part on the left of the arrow the *left-hand side* of the rule and the part on the right of the arrow the *right-hand side* of the rule. The symbols $\langle, \rangle$ are used in the notation of the rules in order to distinguish the symbol to be rewritten from its contexts. A rule is said to match if V has been found in the string and the substring immediately on the left of V corresponds to V_L and the substring immediately on the right of V corresponds to V_R. Note that V_L and V_R represents the so-called *left context* and *right context* of V, respectively. The context may even be an empty string. A matching rule is said to be *applied* if the symbol V in the string to be rewritten is substituted by the right side of the rule V_T.

For the purposes of sorting network development, an extended concept of IL-systems will be proposed herein and denominated as multisymbol deterministic IL-system (or MDIL-system). In this extension, the rewriting rules have the form $V_L\langle V_S\rangle V_R \rightarrow V_T$, where $V_S \in \Sigma^+$, which means that a string V_S rather than a single symbol V can be rewritten. This allows the string to be rewritten to shrink in length if needed (or even V_S to be replaced by an empty string). A given finite number of candidate rewriting rules is arranged sequentially in each chromosome of the EA. Each rule is assigned a unique priority that is determined by its position in the chromosome. Specifically, the first rule takes the highest priority, while the last rule in the chromosome has the lowest priority. The processing of the rules according to their priorities ensures that the rewriting is deterministic in the case of more than one match at the same or conflicting position (e.g., if overlapping substrings are to be rewritten). The rewriting process in MDIL-system works as follows. The rules are investigated (according to the priorities from the highest to the lowest) for a match with respect to the string to be rewritten. The inspection of the match in the string is performed sequentially left to right. All nonoverlapping positions that match a rule are marked to be rewritten. After determining all matches, the matching rules are applied in parallel on the appropriate positions in the string.

For example, let us take into account the *string 1* and *Evolved rewriting rules* block in Fig. 4 and let us demonstrate a developmental step in order to generate the *string 2*. Fig. 6 shows a detailed scheme of this step. The *string 1* is inspected from left to right for the match of the rewriting rules. If more than one matching rule is found for a particular position, then the rule with the higher priority is chosen to be applied. As the first match the string "S" of the rule R2 is found at index 3, hence the string S is marked to be rewritten. Note that the rule R3 also matches at index 3, however, it has a lower priority and hence cannot be applied at the same position. The next match is the rule R1 (string "D" at index 10), i.e., the string D will also be marked for rewriting. Since no more matches can be found in *string 1*, the matching rules are applied at the marked strings and *string 2* is derived.

D. Evolution of Rewriting Rules

In order to design rewriting rules for the MDIL-system using an EA, a chromosome structure based on a 1-D array of symbols was chosen which encodes the axiom that is also a subject of evolution, and a sequence of r rewriting rules in the format described in Section III-C (the number of rewriting rules r is specified as a parameter). Each chromosome has the structure $[axiom \mid V_{L1}\langle V_{S1}\rangle V_{R1} \rightarrow V_{T1} \mid \ldots \mid V_{Lr}\langle V_{Sr}\rangle V_{Rr} \rightarrow V_{Tr}]$.

A simple GA combined with the MDIL-system-based developmental model was applied for the evolutionary development of growing sorting networks. The *Genetic algorithm* block in Fig. 4 implements an evolutionary scheme whose pseudocode is shown in the listing of Algorithm 1. The following GA settings were experimentally determined and used to obtain the results presented in Section IV.

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: GA Used

```
Initialize the time, t = 0.
Randomly generate the initial population, P(0).
while the stop condition is false do
    Develop SNs using each chromosome in P(t).
    Evaluate fitness of the SNs, assign it to chromosomes.
    The best chromosome and its mutant go into P(t + 1).
    while P(t + 1) is not full do
        A parent is the winner of tournament from P(t).
        Mutate the parent and put it into P(t + 1).
    end
    t = t + 1
end
```

The evolutionary process (i.e., each independent run of the GA) has two goals. The first goal, called the *design goal*, is to find such an axiom and rewriting rules that are able to generate valid sorting networks for the given number of developmental steps. We call such sets of rewriting rules *fully operational*. The second goal, called the *optimization goal*, is to optimize the parameters of the generated sorting networks. The fitness function (which will be described later in this section) ensures that after the design goal has been fulfilled, the remaining computational time of the run is utilized solely for the optimization goal. This approach, performing both the evolutionary design and optimization in a single run, was applied on the basis of our initial experiments with the design goal only, which produced results of poor quality (the sorting networks consisted of a high number of comparators and exhibited large delays).

The fitness evaluation of each chromosome is performed as follows. Each chromosome is used to develop s sorting networks, using the approach described in Section III-B, where s denotes a parameter specifying the number of developmental steps. After each developmental step $t = 1, \dots s$ the fitness f_t of the w_t-input sorting network is calculated as the number of correctly sorted output bits for each of the 2^{w_t} possible input binary test vectors. The fitness of a chromosome for the design objective is given by $F_{\text{design}} = \sum_{t=1}^{s} f_t$ and the maximal design fitness of a chromosome (that is able to develop fully working networks within the given number of steps) is $F_{\max} = \sum_{t=1}^{s} w_t 2^{w_t}$. The number of erroneous output bits can be expressed as $E = F_{\max} - F_{\text{design}}$, the value of which equals 0 for fully operational solutions. The metric E is used for calculating the fitness of candidate solutions during the design goal and optimization goal. Let C_s denote the total number of comparators and D_s the delay of a sorting network developed after the sth step. If $E > 0$ for a candidate solution, then its fitness is given by the first row of (1). The evolutionary run is considered successful if a fully operational set of rules has been obtained.

The aim of the optimization goal is to reduce the number of comparators and delay of resulting sorting networks. Therefore, after detecting at least one fully operational solution (with $E = 0$), such solutions need to be *strongly* preferred to other candidate solutions. In order to do that, a suitable constant $b \gg F_{max}$ is introduced (specifically, $b = 10^9$ in this paper) and the fitness of fully operational solutions will be determined according to the second row of

$$F = \begin{cases} F_{\max} - E - C_s - D_s & \text{if } E > 0 \\ F_{\max} + b - C_s - D_s & \text{if } E = 0. \end{cases} \quad (1)$$

The following genetic operators are used during the evolution. The tournament selection operator with base 4 is applied for selecting a parent chromosome in each generation. The parent undergoes mutation which is performed by randomly selecting one of the following ways: 1) each symbol in the chromosome is replaced by a new random symbol with the probability 0.2; 2) a randomly selected symbol is *removed* from the chromosome (this allows for shortening the strings the rewriting rules are composed of); and 3) a new random symbol is *inserted* at a randomly selected position in the chromosome (this allows extending the strings; the insertion is not performed if the maximal length of the given part of the rule would be exceeded). Moreover, a special swap operator is applied with the probability 1/(RULE SWAP), the parameter RULE SWAP = 6 in this paper. In particular, the swap is performed by randomly selecting two rules and swapping their positions in the chromosome. This operator has been introduced herein in order to allow us to change the priority of the rules without necessarily changing their form.

IV. Experimental Results

In order to test the hypothesis posed in Section I-B, several sets of experiments have been conducted using the proposed evolutionary developmental system in various configurations. For each configuration, 120 independent evolutionary runs have been performed. The time of each run was 12 h using multicore Intel Xeon E5-2695 processors on the Salomon cluster[1] that is a part of Czech IT4Innovations National Supercomputing Center.[2] The cluster provides more than a thousand of equivalent 24-core nodes which ensures that each run has the same amount of available computing resources.

The experiments have been evaluated statistically with respect to the success rate and computational effort expressed as the number of generations performed within the given time limit. As the specific combination of the given configuration parameters influence the cardinality of the search space and the time needed to evaluate a candidate solution, the correlation between the computational effort and success rate for various configurations may show us how efficient the proposed system is. This approach was chosen since it is difficult to find the most suitable criterion for terminating the evolution.

In order to evaluate the results, fully operational solutions from each set of experiments have been tested by further development of larger sorting networks. More specifically, we evaluated the correctness of resulting networks by performing four further developmental steps in addition to the three steps which were performed during the evolution. For the purposes of this paper, the solutions that pass the test are considered general (see [77] for details of how to prove this formally).

It is worth noting that most results obtained for the growing networks produce solutions containing redundant comparators.

[1] https://docs.it4i.cz/salomon/hardware-overview/

[2] https://www.it4i.cz/

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

TABLE I
OVERVIEW OF PARAMETERS AND THEIR SETTINGS USED IN THE PROPOSED EVOLUTIONARY DEVELOPMENTAL SYSTEM

Rewriting system		Genetic algorithm	
Parameter name	**Value(s)**	**Parameter name**	**Value(s)**
ALPHABET	{S,U,D,1,2,3,4,a,b,c,d}	POPULATION SIZE	8
MAXIMAL AXIOM LENGTH	26	TOURNAMENT BASE	4
MAX. LEFT CONTEXT LENGTH	1, 2	CROSSOVER	NOT USED
NUM. OF SYMBOLS TO REWRITE	1, 2	MUTATION PROBABILITY	0.2
MAX. RIGHT CONTEXT LENGTH	0, 1, 2	RULE SWAP	6
MAX. RIGHT SIDE LENGTH	12, 18, 24	DEVELOPMENTAL STEP SIZE	2, 3, 4
NUM. OF REWRITING RULES	1, 3, 6, 8	NUM. OF DEVEL. STEPS	3

In order to present the effective cost of the solutions, the data in the following sections regarding the number of comparators and delay will be presented after removing redundant components. Although redundancy is undesirable and conventional design methods can avoid it naturally, this represents an issue in the case of evolutionary design. We identify redundant comparators by observing their activity (i.e., swapping the input values) during the evaluation of the network. Those comparators that have not swapped their inputs at all can be removed without any significant computational overhead. This way the evaluation time of candidate solutions may be reduced.

A. Evolutionary System Setup

In the area of evolutionary computation, it is usual to tune the parameters of the experimental system in order to achieve a reasonable performance, evolvability, and quality of obtained solutions. However, the proposed system requires a higher number of parameters to be set and the tuning of them all does not represent a reasonable approach. Therefore, for the purpose of this paper, some of the parameters have been set experimentally, on the basis of our previous experience or according to recommendations from the literature. Some of them are the subject of various experimental setups for conducting experiments whose results will be presented in this section. More specifically, we have identified that a higher selection pressure during the evolution is needed in order to obtain good solutions. In this paper, this is accomplished by elitism, the appropriate tournament selection base and a strong preference of fully operational solutions in the fitness function (as described in Section III-D). A complete list of the parameters involved by the proposed system and their values chosen is shown in Table I. The parameters with multiple values specified are those for the experimental evaluation.

In order to evaluate abilities of the proposed system with respect to various aspects of the rewriting-based developmental process, several forms of rewriting rules have been used that differ in the maximal allowed length of the contexts and the strings to be rewritten, as well as the maximal length of the right side of the rules. For example, a rule of the form $1\langle 1\rangle 0 \rightarrow 12$ means that it may have the left context of at most one symbol, does not use the right context and replaces just a 1-symbol-substring by a string of at most 12 symbols. Other forms of rules, considered in our experiments, are $1\langle 1\rangle 1 \rightarrow 18$ and $2\langle 2\rangle 2 \rightarrow 24$. For each form of the rewriting rules, several selected combinations of values of the number of rewriting rules and size of the developmental step were evaluated.

The statistics of all the sets of experiments are listed in Table II. The configuration of each experiment is specified by the size of the developmental step, the form and the number of the rewriting rules. The resulting data from each set includes the number of fully operational solutions and the number of general solutions out of the 120 independent runs. Moreover, the average number of generations processed within the 12-h limit is presented together with its standard deviation. The number of generations is inversely proportional to the computational cost of the experiment (i.e., the lower value per 12 h, the higher the computational cost).

We analyzed the statistical results and found an interesting correlation between the search space size (which depends on the type and the number of the rewriting rules), the number of fully operational solutions (including the number of general solutions), and the computational effort. For example, the cardinality of the search space for the simplest case (i.e., a single rewriting rule containing up to two symbols on the left and up to 12 symbols on the right), the search space constitutes approximately $11^{2+12} = 3.8 \times 10^{14}$ candidate solutions (note that 11 is the number of symbols in the alphabet). For the rule type $2\langle 2\rangle 2 \rightarrow 24$ it is $11^{6+24} = 2.82 \times 10^{187}$ candidate solutions. The analysis showed that the number of fully operational solutions, computational effort as well as the number of general solutions, exhibit better values with the increasing size of the search space. This indicates that a larger search space may contain a non-negligible amount of potentially good solutions and the proposed EA can effectively explore such a search space. It is, however, intractable to examine such search spaces completely.

B. Results for the Developmental Step of Size 2

The experiments performed using the developmental step of size 2 were motivated with the aim of rediscovering or possibly improving (by means of the rewriting system) the best solution published in [46]. As shown in Table II, the vast majority of experiments provided more than 100 fully operational results out of 120 runs. Although the number of general solutions is very low for some configurations, it increases significantly if more resources are provided in the evolution (see Table II).

Although no improvement in the resulting sorting networks obtained in this set of experiments has been observed with respect to [46], an interesting general context-free system has been obtained that generates sorting networks of the same structure as in [46]. While in [46] the resulting networks contained some redundant comparators and had to be manually optimized, the solution obtained by means of the rewriting system generates sorting networks *without any redundant comparators*. In fact, the known way of growing even-input sorting networks that was published for the first time in [46] has been *rediscovered* in this paper. The result obtained by the evolved rewriting system is illustrated in Fig. 7. The solution works with three rewriting rules, the highest-priority one rewrites the symbol 1 to ϵ (the empty string). This causes removing all 1 s from the current string during each derivation, subsequently more 1 s immediately emerge again at other positions in the string by applying the remaining

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

TABLE II
RESULTS OF EVOLUTIONARY EXPERIMENTS OUT OF 120 RUNS FOR EACH CONFIGURATION. THE COLUMNS REPRESENT: THE AVERAGE NUMBER OF GENERATIONS PERFORMED WITHIN A 12-h LIMIT (AVG. #GENS), THE NUMBER OF FULLY OPERATIONAL RESULTS (#FULLY OPER. RESULTS) AND THE NUMBER OF RESULTS WHICH ARE CONSIDERED TO BE GENERAL (#GEN. RESULTS).

Rules in chromosome		**1 rule**				**3 rules**				**6 rules**				**8 rules**			
Step size	**Rule type**	**#gen. results**	**#fully oper. results**	**Avg. #gens $\times 10^6$**	**Std. dev. [gens]**	**#gen. results**	**#fully oper. results**	**Avg. #gens $\times 10^6$**	**Std. dev. [gens]**	**#gen. results**	**#fully oper. results**	**Avg. #gens $\times 10^6$**	**Std. dev. [gens]**	**#gen. results**	**#fully oper. results**	**Avg. #gens $\times 10^6$**	**Std. dev. [gens]**
2	$1\langle 1\rangle 0 \rightarrow 12$	0	85	1794	260	13	117	1112	773	65	119	802	182	90	119	668	237
	$1\langle 1\rangle 1 \rightarrow 18$	1	82	1682	356	11	117	860	174	44	120	678	174	65	120	556	176
	$2\langle 2\rangle 2 \rightarrow 24$	52	65	1980	224	81	117	1131	818	92	120	697	110	91	119	587	104
3	$1\langle 1\rangle 0 \rightarrow 12$	39	108	1354	325	54	119	700	131	61	117	473	155	65	118	449	741
	$1\langle 1\rangle 1 \rightarrow 18$	60	109	1263	301	66	118	686	142	60	119	527	805	57	120	328	107
	$2\langle 2\rangle 2 \rightarrow 24$	58	99	1304	374	65	118	705	481	66	119	466	434	63	115	430	483
4	$1\langle 1\rangle 0 \rightarrow 12$	3	90	1017	386	52	112	574	237	56	117	477	879	42	115	348	589
	$1\langle 1\rangle 1 \rightarrow 18$	43	105	920	365	57	115	574	580	49	117	345	251	38	120	233	68
	$2\langle 2\rangle 2 \rightarrow 24$	52	103	910	373	49	117	531	561	60	118	330	98	54	118	285	519

TABLE III
COMPARISON OF THE NUMBER OF COMPARATORS OF THE BEST FOUND SOLUTION TO OTHER METHODS. REDUNDANT COMPARATORS WERE REMOVED

Width	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
Step 2	22		35		51		70		92		117		145		176		210		247		287		330		376
Step 3	23			43			69			101			139			183			233			289			351
Step 4A			34				65				106				157				218				289		
Step 4B			31				58				93				136				187				246		
From 4B	19				41				71				109				155				209				271
Existing methods																									
Bubble-Sort	28	36	45	55	66	78	91	105	120	136	153	171	190	210	231	253	276	300	325	351	378	406	435	465	496
Bitonic sort	24								80																240
Odd-Even merge.	19								63																191
Best known[3]	19	25	29	35	39	45	51	56	60	71	78	85	91	100	107	115	120	132	139	150	155	165	172	180	185

TABLE IV
COMPARISON OF THE DELAY OF THE BEST FOUND SOLUTION TO OTHER METHODS. REDUNDANT COMPARATORS WERE REMOVED

Width	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
Step 2	9		12		15		18		21		24		27		30		33		36		39		42		45
Step 3	10			15			20			25			30			35			40			45			50
Step 4A			11				16				21				26				31				36		
Step 4B			9				14				18				23				27				32		
From 4B	6				10				15				19				24				28				33
Existing methods																									
Bubble-Sort	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51	53	55	57	59	61
Bitonic sort	10								15																21
Odd-Even merge.	10								15																21
Best known[4]	6	7	7	8	8	9	9	9	9	10	11	11	11	12	12	12	12	14	14	14	14	14	14	14	14

rules. Note that 1 encodes a comparator across neighboring wires, so it represents a very important symbol for creating working sorting networks. Another general solution for the step size 2 was discovered using a single rule in the chromosome (specifically, the axiom $1da2SD1$ and the rule $\langle 1d\rangle \rightarrow DbUaDSD1d2aU2dDbSDcb1UUU$) that produces equivalent sorting networks as those in Fig. 7. The number of comparators and delay of the corresponding sorting networks are shown in Tables III and IV, respectively, (in the rows denominated "step 2").

C. Results for the Developmental Step of Size 3

The increase of the step size provides more possibilities of the comparator arrangements during the growth of the sorting networks. From a single comparator (i.e., a 2-input initial network), larger instances with 5, 8, 11, etc. inputs can be developed. Fig. 8 shows one of the most interesting solutions obtained in this set of experiments using 3 rewriting rules in the chromosome. In addition, the evolved rewriting system generates sorting networks without any redundant comparators. The number of comparators and delay of the corresponding sorting networks are shown in Tables III and IV, respectively, with the appropriate data denominated "step 3." It can be seen that although the delay of these networks is worse against the solutions for step 2 (see Table IV), the number of comparators in this case exhibits lower values for the number of inputs (width) 14 or higher—see Table III. As evident for the widths of the networks appropriate for both the steps 2 and 3 solutions, the step 3 networks contain less comparators against the step 2 solution and this difference increases for larger networks (e.g., the step 3 solution is

[3]See the Appendix in the supplementary material for details.
[4]See the Appendix in the supplementary material for details.

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

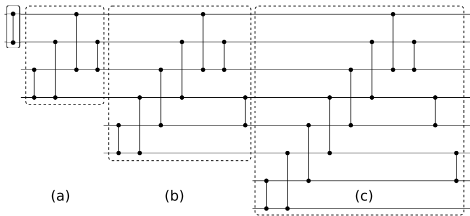

Fig. 7. Even-input sorting networks generated from the evolved axiom $bb1U42c1Da$ and the set of rewriting rules $\{\langle 1\rangle \rightarrow \epsilon, \langle 4\rangle \rightarrow DD4d12U2U, \langle a\rangle \rightarrow 1aDa\}$. The structures used to grow the networks are coded by means of the derived strings (a) $bbUDD4d12U2U2cD1aDa$, (b) $bbUDDDD4d12U2Ud2U2U2cD1aDaD1aDa$, and (c) $bbUDDDDDD4d12U2Ud2U2Ud2U2U2cD1aDaD1aDaD1aDaD1 - aDa$. The lengths of the strings coding the networks shown is 19, 31, and 48 symbols, respectively. This result was obtained using step size 2 and 3 rules of the type $1\langle 1\rangle 0 \rightarrow 12$.

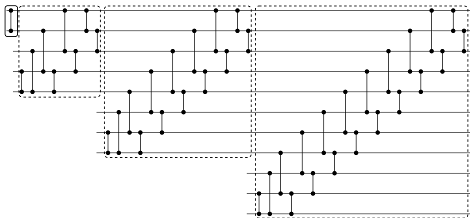

Fig. 8. Sorting networks generated for the step size 3 using the evolved axiom $DccSScU133cb4b44UDb1D1$ and rewriting rules $\langle S\rangle \rightarrow SDDS$, $\langle 4\rangle \rightarrow \epsilon$ and $\langle b\rangle \rightarrow bcU2DDd1UU$. The derived strings are not shown because of larger lengths (52, 91, and 142 symbols after the first, second, and third derivation, respectively). This result was obtained using three rules of the type $1\langle 1\rangle 0 \rightarrow 12$.

better by 25 comparators for 32 inputs compared to the step 2 network). Therefore, this result partially supports the validity of the hypothesis posed in Section I-B—for the number of comparators, in particular.

D. Results for the Developmental Step of Size 4

The most advanced experiments conducted in this paper utilized the developmental step of size 4, the aim of which was to achieve further improvement both in the number of comparators and delay of the networks. Two general solutions were obtained in which the evolution optimized both these parameters which eventually fully confirmed the hypothesis stated in Section I-B. Both were obtained using the rewriting rules of the form $2\langle 2\rangle 2 \rightarrow 24$. Since none of these results were able to generate sorting networks without redundant comparators, a final analysis was performed after their removal.

The first result was obtained with eight rewriting rules in the chromosome and the corresponding solution is illustrated in Fig. 9. The comparator arrangements appended to the growing network after each of the three developmental steps shown in this figure consists of 14 comparators (3 are redundant, i.e., 21%), 32 comparators (11 are redundant, i.e., 34%), and 50 comparators (19 are redundant, i.e., 38%), respectively. It is evident that the percentage of redundant comparators in each developmental step increases. However, a further analysis of these networks showed that after removing the redundant comparators the total number of comparators and delay of the networks optimized this way can be reduced

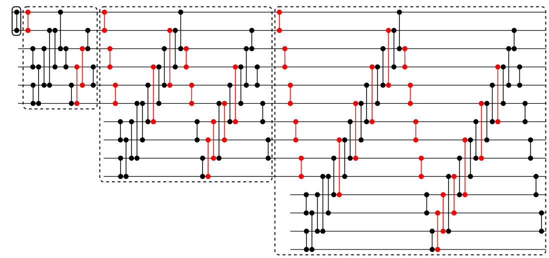

Fig. 9. Sorting networks generated for the step size 4 using the evolved axiom $c1d32dbbDD11b4U1D$ and rewriting rules $\langle 4\rangle \rightarrow aaUccDb$, $\langle 21\rangle \rightarrow \epsilon$, $\langle 4\rangle \rightarrow b2Dd1dddc$, $\langle 4\rangle \rightarrow SdScd2UcabU$, $\langle a3\rangle \rightarrow Sa1$, $\langle b\rangle \rightarrow SDUabcU32U2$, $\langle I\rangle \rightarrow S1DcD1DDa1$, $\langle I\rangle \rightarrow SSbc4UD3Uc34SD1S4bS$. The comparators shown in red are redundant. This result was obtained using eight rules of the type $2\langle 2\rangle 2 \rightarrow 24$.

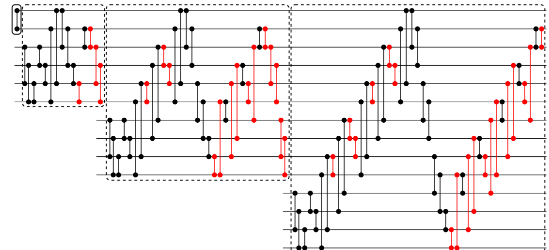

Fig. 10. Sorting networks generated for the step size 4 using the evolved axiom $S42UUUUD2DS11$ and a single rewriting rule $\langle S\rangle \rightarrow DD2D2DS41U4U41DUD1UUUD4U$. The comparators shown in red are redundant. This solution represents the best result of this paper and was obtained using a single rule of the type $2\langle 2\rangle 2 \rightarrow 24$.

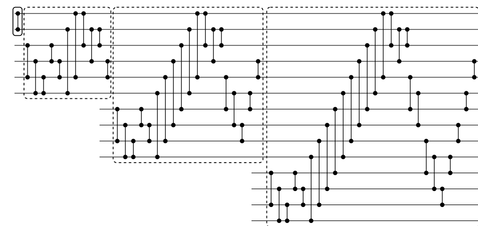

Fig. 11. Manually optimized sorting networks created by removing redundant comparators from the solution shown in Fig. 10 which represents the best result of this paper.

substantially. The resulting properties of the optimized sorting networks are summarized in Tables III and IV for the number of comparators and delay, respectively, the appropriate data is denominated "step 4A." As can be seen, there is a certain improvement against both steps 3 and 2 solutions. For example, the 14-input step 4A sorting network consists of 65 effective comparators (compared to 69 and 70 for steps 3 and 2 solutions, respectively—see Table III), the delay of the step 4A solution exhibits the value 16 (compared to 20 for steps 3 and 18 for the step 2 nets—see Table IV).

The second result, which represents the best solution of this paper, has been obtained using a single rewriting rule in the chromosome and the corresponding network structures are shown in Fig. 10. In addition, the sorting networks generated in this case contain some redundant comparators that had to be removed before performing the final analysis. Fig. 11 displays the result without the redundant comparators. More specifically, there are 15 comparators appended to the growing

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

network after the first developmental step (out of which 4 are redundant, i.e., 27%), 33 comparators appended after the second step (14 are redundant, i.e., 42%) and 45 comparators appended after the third step (18 are redundant, i.e., 40%) as shown in Fig. 10. The properties of the final optimized sorting networks are summarized in Table III (the number of comparators) and Table IV (delay) and the appropriate data is denominated step 4B. The comparison of both parameters against the previously mentioned solutions shows that the improvement in the step 4B result is remarkable. For example, the 26-input step 4B network consists of 187 comparators and exhibits delay 27 (the previous network needed 218 comparators for 26 inputs with delay 31, the step 3 solution of the same width is a 233-comparator network with the delay 40 and the solution using step 2 contains 247 comparators and delay 36 in the case of the 26-input network).

E. Comparison and Discussion

In order to summarize the evaluation of the results obtained, whose properties are shown in Tables III and IV, a comparison with some other selected generic sorting network construction methods has been performed. For this purpose, the existing Bubble-Sort algorithm, Bitonic sort and odd–even merge-sort have been chosen whose properties are summarized in these tables under the title "Existing methods." The Bubble-Sort is mentioned just for completeness because it represents the least efficient sorting algorithm both from the view of the number of comparators and delay. However, it uses the same principle of growing sorting networks as considered in this paper, hence the comparison with this method is adequate. As shown in Section IV-B, a more efficient generic method which overcomes the Bubble-Sort is the step 2 approach, originally published in [46], which has been rediscovered in this paper. However, the results obtained by utilizing the step size 3 and 4 represent the main contribution of this paper because they overcome both the Bubble-Sort and step 2 solution. In particular, the step 4 solutions represent the best results of this paper so that a more detailed comparison with other existing generic algorithms will be described.

As two further sorting network construction methods, the Bitonic sort and odd–even merge-sort have been chosen for comparison. These methods belong to the best known generic approaches, although their principles are different from those considered in this paper. The original idea of these methods are based on a direct construction (i.e., not growing) of sorting networks whose number of inputs equals a power of two. Therefore, only the numbers of inputs 8, 16, and 32 have data specified in Tables III and IV. Although none of the results proposed in this paper overcame the Bitonic or odd–even method, our best result (step 4B) definitely has some advantages.

1) The proposed step 4B solution creates sorting networks for more values of the number of inputs than the aforementioned existing methods. More specifically, any number of inputs given by $4i + 2$, where $i \geq 0$ is an integer parameter, is possible.
2) We determined that any $4i$-input network may be constructed from the $4i + 2$-input network, provided by the original step 4B solution, by removing the two top wires and the comparators connected to them. The properties of these $4i$-input networks are summarized in Tables III and IV in the rows denoted "From 4B." It is evident from the tables that these networks still exhibit better properties compared to all previous results (i.e., steps 2, 3, or 4A). This way, any even-input sorting network can be obtained using the step 4B solution.
3) Since the proposed method for growing sorting networks assumes that the smaller k-input network, from which the $k + \Delta$-input network ought to be grown (Δ is the step size), is valid, it is possible (by means of the solutions obtained) to create a $k+\Delta$-input network from any valid k-input network. For example, a 36-input network may be created from the 32-input odd–even merge-sort network by appending 4 inputs and the set of comparators from the result step 4B whereas the resulting network will exhibit substantially better properties than if constructed by means of another conventional method (e.g., the Bubble-Sort).

We have analyzed the obtained results and identified the following aspect regarding the rewriting rules of the resulting solutions. Surprisingly, context-sensitive rules occur very rarely. In fact, none of the best solutions in this paper contain such a rule. Furthermore, most of the solutions utilize only one or two rules which expand the string, all other rules remove symbols from the string.

V. Conclusion

In this paper, an EA has been applied in order to design rules of a rewriting system for the generic construction of sorting networks. The main idea is based on encoding various comparator structures of sorting networks by means of text strings over a suitable alphabet of the rewriting system. A sequence of derivations using a set of evolved rewriting rules from a suitable initial string (axiom) allows us to develop growing sorting networks. Several sets of experiments have been presented in which we have shown that various sorting networks can be developed from a simple initial instance. Various sizes of the developmental step, by which the number of inputs of the growing networks is increased, have been investigated with respect to the properties of the resulting sorting networks. It has been shown that for larger developmental steps, the solutions obtained allow us to develop more efficient sorting networks (considering both the number of comparators and delay) in comparison with similar approaches published in the past.

Although successful experiments have been presented in this paper, from which the best result exhibits remarkably better properties against some of the known methods, it is not possible to say whether the solutions obtained for the given settings are optimal. Since the proposed evolutionary system involves a higher number of control parameters, some of which have been the subject of experimental evaluation, their fine-tuning represents a challenging task. We believe, however, that more in-depth investigation of the evolutionary

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

system and developmental model together with a theoretical analysis of the resulting sorting networks may provide us with valuable information for further research.

REFERENCES

[1] P. J. Bentley, Ed., *Evolutionary Design by Computers*. San Francisco, CA, USA: Morgan Kaufmann, 1999.

[2] D. Corne and P. Bentley, *Creative Evolutionary Systems*. San Francisco, CA, USA: Morgan Kaufmann, 2001.

[3] M. J. Streeter, M. A. Keane, and J. R. Koza, "Routine duplication of post-2000 patented inventions by means of genetic programming," in *Proc. 5th Eur. Conf. Genet. Program.*, vol. 2278, 2002, pp. 26–36. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45984-7_3#citeas

[4] L. Sekanina and R. S. Zebulum, "Intrinsic evolution of controllable oscillators in FPTA-2," in *Evolvable Systems: From Biology to Hardware*, J. M. Moreno, J. Madrenas, and J. Cosp, Eds. Heidelberg, Germany: Springer, 2005, pp. 98–107.

[5] Q. Wu, Y. Shi, J. Zheng, R. Yao, and Y. Wang, "Analog circuit evolution based on FPTA-2," in *Evolvable Systems: From Biology to Hardware*, L. Kang, Y. Liu, and S. Zeng, Eds. Heidelberg, Germany: Springer, 2007, pp. 109–118.

[6] C. Mattiussi and D. Floreano, "Analog genetic encoding for the evolution of circuits and networks," *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 596–607, Oct. 2007.

[7] V. K. Vassilev, D. Job, and J. F. Miller, "Towards the automatic design of more efficient digital circuits," in *Proc. 2nd NASA/DoD Workshop Evol. Hardw.*, 2000, pp. 151–160.

[8] V. Mrázek and Z. Vašíček, "Evolutionary design of transistor level digital circuits using discrete simulation," in *Proc. Genet. Program. 18th Eur. Conf. (EuroGP)*, vol. 9025, 2015, pp. 66–77. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-16501-1_6#citeas

[9] J. D. Lohn, D. S. Linden, G. S. Hornby, and W. F. Kraus, "Evolutionary design of an X-band antenna for NASA's space technology 5 mission," in *Proc. NASA/DoD Conf. Evol. Hardw.*, 2003, pp. 155–163.

[10] G. Oliveri, P. Rocca, M. Salucci, and A. Massa, "Evolution of nature-inspired optimization for new generation antenna design," in *Proc. IEEE Symp. Comput. Intell. Commun. Syst. Netw. (CIComms)*, Orlando, FL, USA, 2014, pp. 1–6.

[11] V. A. Shakhnov, L. A. Zinchenko, V. Verstov, and B. Sorokin, "Evolutionary antenna design for intelligent systems networks," in *Proc. IEEE Int. Conf. Evol. Adapt. Intell. Syst. (EAIS)*, 2015, pp. 1–6.

[12] Z. Vašíček, M. Bidlo, and L. Sekanina, "Evolution of efficient real-time non-linear image filters for FPGAs," *Soft Comput.*, vol. 17, no. 11, pp. 2163–2180, 2013.

[13] P. Kaufmann, K. Glette, T. Gruber, M. Platzner, J. Torresen, and B. Sick, "Classification of electromyographic signals: Comparing evolvable hardware to conventional classifiers," *IEEE Trans. Evol. Comput.*, vol. 17, no. 1, pp. 46–63, Feb. 2013.

[14] A. Thompson, "An evolved circuit, intrinsic in silicon, entwined with physics," in *Evolvable Systems: From Biology to Hardware*, T. Higuchi, M. Iwata, and W. Liu, Eds. Heidelberg, Germany: Springer, 1997, pp. 390–405.

[15] A. Thompson, P. Layzell, and R. S. Zebulum, "Explorations in design space: Unconventional electronics design through artificial evolution," *IEEE Trans. Evol. Comput.*, vol. 3, no. 3, pp. 167–196, Sep. 1999.

[16] J. R. Koza, F. H. Bennett, J. L. Hutchings, S. L. Bade, M. A. Keane, and D. Andre, "Evolving sorting networks using genetic programming and the rapidly reconfigurable Xilinx 6216 field-programmable gate array," in *Proc. Conf. Rec. 31st Asilomar Conf. Signals Syst. Comput.*, vol. 1, 1997, pp. 404–410.

[17] A. Stoica, X. Guo, R. S. Zebulum, M. I. Ferguson, and D. Keymeulen, "Evolution-based automated reconfiguration of field programmable analog devices," in *Proc. IEEE Int. Conf. Field Program. Technol. (FPT)*, Hong Kong, 2002, pp. 403–406.

[18] J. F. Miller, S. L. Harding, and G. Tufte, "Evolution-in-materio: Evolving computation in materials," *Evol. Intell.*, vol. 7, no. 1, pp. 49–67, 2014.

[19] M. Mohid, J. F. Miller, S. L. Harding, G. Tufte, M. K. Massey, and M. C. Petty, "Evolution-in-materio: Solving computational problems using carbon nanotube–polymer composites," *Soft Comput.*, vol. 20, no. 8, pp. 3007–3022, 2016.

[20] J. Torresen, "A scalable approach to evolvable hardware," *Genet. Program. Evol. Mach.*, vol. 3, no. 3, pp. 259–282, Sep. 2002.

[21] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi, "Hardware evolution at function level," in *Proc. 4th Int. Conf. Parallel Problem Solving Nat. (PPSN)*, vol. 1141, 1996, pp. 206–217. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-61723-X_970#citeas

[22] P. C. Haddow and G. Tufte, "Bridging the genotype–phenotype mapping for digital FPGAs," in *Proc. 3rd NASA/DoD Workshop Evol. Hardw.*, 2001, pp. 109–115.

[23] T. G. W. Gordon and P. J. Bentley, "Towards development in evolvable hardware," in *Proc. NASA/DoD Conf. Evol. Hardw.*, Alexandria, VA, USA, 2002, pp. 241–250.

[24] Z. Vašíček and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genet. Program. Evol. Mach.*, vol. 12, no. 3, pp. 305–327, 2011.

[25] L. Wolpert, *Principles of Development*. Oxford, U.K.: Oxford Univ. Press, 2015.

[26] S. Kumar, "Investigating computational models of development for the construction of shape and form," Ph.D. dissertation, Dept. Comput. Sci., Univ. College London, London, U.K., 2004.

[27] C. Ryan, J. J. Collins, and M. O'Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Proc. 1st Eur. Workshop Genet. Program.*, vol. 1391, 1998, pp. 83–96. [Online]. Available: https://link.springer.com/chapter/10.1007/BFb0055930#citeas

[28] C. Ryan, M. O'Neill, and J. J. Collins, Eds., *Handbook of Grammatical Evolution*. Cham, Switzerland: Springer, 2018.

[29] T. Kuroda, H. Iwasawa, and E. Kita, "Application of advanced grammatical evolution to function prediction problem," *Adv. Eng. Softw.*, vol. 41, no. 12, pp. 1287–1294, 2010.

[30] M. Fenton, J. Byrne, and E. Hemberg, "Design, architecture, and engineering with grammatical evolution," in *Handbook of Grammatical Evolution*. Cham, Switzerland: Springer, 2018, pp. 317–339.

[31] E. Kita, H. Sugiura, Y. Zuo, and T. Mizuno, "Application of grammatical evolution to stock price prediction," *Comput. Assist. Methods Eng. Sci.*, vol. 24, no. 1, pp. 67–81, 2017.

[32] M. Fenton, D. Lynch, S. Kucera, H. Claussen, and M. O'Neill, "Multilayer optimization of heterogeneous networks using grammatical genetic programming," *IEEE Trans. Cybern.*, vol. 47, no. 9, pp. 2938–2950, Sep. 2017.

[33] A. Lindenmayer, "Mathematical models for cellular interaction in development: Parts I and II," *J. Theor. Biol.*, vol. 18, no. 3, pp. 280–315, Mar. 1968. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/5659071

[34] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. New York, NY, USA: Springer-Verlag, 1996.

[35] R. Dawkins, *The Blind Watchmaker*. New York, NY, USA: Norton, 2015.

[36] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Syst.*, vol. 4, no. 4, pp. 461–476, 1990.

[37] E. J. W. Boers and H. Kuiper, "Biological metaphors and the design of artificial neural networks," M.Sc. thesis, Dept. Comput. Sci. Exp. Theor. Psychol., Leiden University, Leiden, The Netherlands, 1992.

[38] G. S. Hornby and J. B. Pollack, "The advantages of generative grammatical encodings for physical design," in *Proc. Congr. Evol. Comput.*, 2001, pp. 600–607.

[39] P. C. Haddow, G. Tufte, and P. van Remortel, "Shrinking the genotype: L-systems for EHW?" in *Proc. 4th Int. Conf. Evol. Syst. Biol. Hardw.*, vol. 2210, 2001, pp. 128–139. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45443-8_11#citeas

[40] G. Escuela, G. Ochoa, and N. Krasnogor, "Evolving L-systems to capture protein structure native conformations," in *Genetic Programming*, M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, Eds. Heidelberg, Germany: Springer, 2005, pp. 74–84.

[41] D. Ashlock, K. M. Bryden, and S. P. Gent, "Simultaneous evolution of bracketed L-system rules and interpretation," in *Proc. IEEE Int. Conf. Evol. Comput.*, Vancouver, BC, Canada, 2006, pp. 2050–2057.

[42] D. A. Ashlock, S. P. Gent, and K. M. Bryden, "Evolution of L-systems for compact virtual landscape generation," in *Proc. IEEE Congr. Evol. Comput.*, vol. 3, 2005, pp. 2760–2767.

[43] D. Beaumont and S. Stepney, "Grammatical evolution of L-systems," in *Proc. IEEE Congr. Evol. Comput.*, 2009, pp. 2446–2453.

[44] L. M. L. D. Campos, M. Roisenberg, and R. C. L. D. Oliveira, "Automatic design of neural networks with L-systems and genetic algorithms—A biologically inspired methodology," in *Proc. Int. Joint Conf. Neural Netw.*, 2011, pp. 1199–1206.

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.

[45] J. F. Miller and P. Thomson, "A developmental method for growing graphs and circuits," in *Proc. 5th Conf. Evol. Syst. From Biol. Hardw. (ICES)*, vol. 2606, 2003, pp. 93–104. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-36553-2_9#citeas

[46] L. Sekanina and M. Bidlo, "Evolutionary design of arbitrarily large sorting networks using development," *Genet. Program. Evol. Mach.*, vol. 6, no. 3, pp. 319–347, 2005.

[47] M. Bidlo and J. Škarvada, "Instruction-based development: From evolution to generic structures of digital circuits," *Int. J. Knowl. Based Intell. Eng. Syst.*, vol. 12, no. 3, pp. 221–236, 2008.

[48] M. Bidlo, "On routine evolution of complex cellular automata," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 742–754, Oct. 2016.

[49] T. G. W. Gordon and P. J. Bentley, "Development brings scalability to hardware evolution," in *Proc. NASA/DoD Conf. Evol. Hardw.*, Washington, DC, USA, 2005, pp. 272–279.

[50] A. P. Shanthi *et al.*, "Development based evolution for scalable, fault tolerant digital circuits," in *Proc. Workshop Soft Comput. Int. Conf. High Perform. Comput.*, 2003, pp. 165–176.

[51] M. Bidlo, "Evolution of generic square calculations in cellular automata," in *Proc. 8th Int. Joint Conf. Comput. Intell.*, vol. 3, 2016, pp. 94–102.

[52] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, 2nd ed. Upper Saddle River, NJ, USA: Addison Wesley, 1998.

[53] K. Imamura, J. A. Foster, and A. W. Krings, "The test vector problem and limitations to evolving digital circuits," in *Proc. 2nd NASA/DoD Workshop Evol. Hardw.*, Palo Alto, CA, USA, 2000, pp. 81–87.

[54] D. Bundala and J. Závodný, "Optimal sorting networks," in *Language and Automata Theory and Applications*, A.-H. Dediu, C. Martín-Vide, J.-L. Sierra-Rodríguez, and B. Truthe, Eds. Cham, Switzerland: Springer Int., 2014, pp. 236–247.

[55] K. E. Batcher, "Sorting networks and their applications," in *Proc. Comput. Conf. (AFIPS)*, Apr./May 1968, pp. 307–314.

[56] C. Rüb, "On Batcher's merge sorts as parallel sorting algorithms," in *Proc. STACS*, vol. 1373. Paris, France, 1998, pp. 410–420. [Online]. Available: https://link.springer.com/chapter/10.1007%2FBFb0028577#citeas

[57] L. I. Schiller, "An agglomeration law for sorting networks and its application in functional programming," in *Proc. 29th 30th Workshops (Constraint) Logic Program. (WLP) 24th Int. Workshop Functional (Constraint) Logic Program. (WFLP)*, 2017, pp. 165–179.

[58] M. Zuluaga, P. Milder, and M. Püschel, "Computer generation of streaming sorting networks," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2012, pp. 1245–1253.

[59] M. Zuluaga, P. Milder, and M. Püschel, "Streaming sorting networks," *ACM Trans. Design Autom. Electron. Syst.*, vol. 21, no. 4, pp. 1–55, 2016.

[60] V. Sklyarov, I. Skliarova, and A. Sudnitson, "FPGA-based accelerators for parallel data sort," *Appl. Comput. Syst.*, vol. 16, no. 1, pp. 53–63, 2015.

[61] M. Codish, L. Cruz-Filipe, and P. Schneider-Kamp, "The quest for optimal sorting networks: Efficient generation of two-layer prefixes," in *Proc. 16th Int. Symp. Symbolic Numer. Algorithms Sci. Comput.*, 2014, pp. 359–366.

[62] D. Bundala, M. Codish, L. Cruz-Filipe, P. Schneider-Kamp, and J. Závodný, "Optimal-depth sorting networks," *J. Comput. Syst. Sci.*, vol. 84, pp. 185–204, Mar. 2017.

[63] M. Codish, L. Cruz-Filipe, and P. Schneider-Kamp, "Sorting networks: The end game," in *Language and Automata Theory and Applications*, A.-H. Dediu, E. Formenti, C. Martín-Vide, and B. Truthe, Eds. Cham, Switzerland: Springer Int., 2015, pp. 664–675.

[64] M. Codish, L. Cruz-Filipe, T. Ehlers, M. Müller, and P. Schneider-Kamp, "Sorting networks: To the end and back again," *J. Comput. Syst. Sci.*, Apr. 2016. [Online]. Available: https://doi.org/10.1016/j.jcss.2016.04.004

[65] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," *Physica D Nonlin. Phenomena*, vol. 42, nos. 1–3, pp. 228–234, 1990.

[66] H. Juillé, "Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces," in *Proc. 6th Int. Conf. Genet. Algorithms*, 1995, pp. 351–358.

[67] S. S. Choi and B. R. Moon, "A hybrid genetic search for the sorting network problem with evolving parallel layers," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2001, pp. 258–265.

[68] S. S. Choi and B. R. Moon, "More effective genetic search for the sorting network problem," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2002, pp. 335–342.

[69] S. S. Choi and B. R. Moon, "Isomorphism, normalization, and a genetic algorithm for sorting network optimization," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2002, pp. 327–334.

[70] S.-S. Choi and B.-R. Moon, "A graph-based lamarckian-baldwinian hybrid for the sorting network problem," *IEEE Trans. Evol. Comput.*, vol. 9, no. 1, pp. 105–114, Feb. 2005.

[71] L. Graham, H. Masum, and F. Oppacher, "Statistical analysis of heuristics for evolving sorting networks," in *Proc. 7th Annu. Conf. Genet. Evol. Comput. (GECCO)*, 2005, pp. 1265–1270.

[72] J. Kubalík, "Solving the sorting network problem using iterative optimization with evolved hypermutations," in *Proc. 11th Annu. Conf. Genet. Evol. Comput. (GECCO)*, 2009, pp. 301–308.

[73] D. Coles, "Efficient filters for the simulated evolution of small sorting networks," in *Proc. 14th Annu. Conf. Genet. Evol. Comput. (GECCO)*, 2012, pp. 593–600.

[74] V. K. Valsalam and R. Miikkulainen, "Using symmetry and evolutionary search to minimize sorting networks," *J. Mach. Learn. Res.*, vol. 14, pp. 303–331, Feb. 2013.

[75] B. C. López-Ramírez and N. Cruz-Cortés, "Designing minimal sorting networks using a bio-inspired technique," *Computación y Sistemas*, vol. 18, pp. 731–739, Dec. 2014.

[76] G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems* (Pure and Applied Mathematics). New York, NY, USA: Academic, 1980.

[77] M. Bidlo, R. Bidlo, and L. Sekanina, "Designing a novel general sorting network constructor using artificial evolution," *Trans. Eng. Comput. Technol.*, vol. 15, no. 10, pp. 85–90, 2006. [Online]. Available: https://waset.org/publications/6144

[78] B. Dobbelaere. (2018). *Smallest and Fastest Sorting Networks for a Given Number of Inputs*. [Online]. Available: http://users.telenet.be/bertdobbelaere/SorterHunter/sorting_networks.html

Michal Bidlo received the Ph.D. degree in information technology from the Faculty of Information Technology (FIT), Brno University of Technology (BUT), Brno, Czechia, in 2009.

He is an Assistant Professor with the Department of Computer Systems FIT BUT. He has authored or coauthored over 20 conference/journal papers focused on evolutionary design and evolvable hardware. His current research interests include cellular automata, evolutionary computation, evolvable hardware and bio-inspired systems.

Michal Dobeš received the bachelor's degree in information technology and the master's degree in intelligent systems from the Faculty of Information Technology (FIT), Brno University of Technology (BUT), Brno, Czechia, in 2015 and 2017, respectively.

His current research interests include prototyping of avionic systems and security and applications of evolutionary algorithms.

Authorized licensed use limited to: Brno University of Technology. Downloaded on April 03,2020 at 06:32:29 UTC from IEEE Xplore. Restrictions apply.