World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:3, 2008

Designing a Novel General Sorting Network Constructor Using Artificial Evolution

Michal Bidlo, Radek Bidlo, and Lukáš Sekanina

***Abstract*—** A method is presented for the construction of arbitrary even-input sorting networks exhibiting better properties than the networks created using a conventional technique of the same type. The method was discovered by means of a genetic algorithm combined with an application-specific development. Similarly to human inventions in the area of theoretical computer science, the evolved invention was analyzed: its generality was proven and area and time complexities were determined.

***Keywords*—** Development, genetic algorithm, program, sorting network.

I. Introduction

THE concept of sorting networks was introduced in 1954; Knuth traced the history of this problem in his book [1]. Sorting networks are combinational circuits of N inputs and N outputs. A compare–swap operation (a comparator) is usually considered as a basic building block of a sorting network. The main goal of the design of a sorting network is (1) to optimize the area of the circuit (i.e. to reduce the number of compare–swap operations) and (2) to optimize the delay of the sorting network (the shorter delay, the faster sorting).

Since the direct design of efficient large sorting networks ($N > 8$) is very difficult, various non-traditional methods have been utilized. For instance, Choi and Moon have devised a fast approximate measure for evaluating the potential quality of a given sorting network and by combining with an effective local search heuristic based on this measure, they have rediscovered the best-known 16-input sorting network [3], [5]. Hillis has used simulated evolution in combination with a species of co-evolving parasites implemented on a parallel computer for generating efficient 16-input sorting networks [4]. However, the first 32 comparators were fixed at the beginning of the design process in case of those methods. On the contrary, Juillé has used a population-based model for searching in the state space and incremental construction of the solutions by means of which he has discovered a new area-efficient 13-input sorting network and rediscovered the best-known 16-input sorting network from scratch, i.e. without fixing any comparators at the beginning of the design process [2]. Koza has used the genetic programming approach to solve various problems, including 7-input sorting networks [6].

Michal Bidlo is with Brno University of Technology, Faculty of Information Technology, Božetěchova 2, 612 66 Brno, Czech Republic (e-mail: bidlom@fit.vutbr.cz).

Radek Bidlo is with Brno University of Technology, Faculty of Information Technology, Božetěchova 2, 612 66 Brno, Czech Republic (e-mail: bidlor@fit.vutbr.cz).

Lukáš Sekanina is with Brno University of Technology, Faculty of Information Technology, Božetěchova 2, 612 66 Brno, Czech Republic (e-mail: sekanina@fit.vutbr.cz).

In contrast to the mentioned approaches, Sekanina and M. Bidlo have devised a developmental method in combination with a genetic algorithm for the design of arbitrarily large sorting networks [7]. By using this approach, the conventional insertion/selection principle for the design of sorting networks has been rediscovered and some novel design methods have been evolved.

The objective of this paper is (1) to describe the genetic algorithm combined with the developmental system used for the design of arbitrary even-input sorting networks, (2) to present the best evolved algorithm which can be considered as a new general sorting network design method together with the resulting sorting networks, (3) to analyze the properties of these sorting networks and compare them with the properties of sorting networks created by means of the conventional insertion/selection principle and (4) to introduce formal description of sorting networks and prove the ability of the evolved algorithm to design arbitrary large sorting networks. Therefore, the evolved algorithm will be analyzed using the methods of theoretical computer science similarly to the analysis of human inventions in the area of theoretical computer science.

The paper is organized as follows. Section II briefly describes the concept of sorting networks. Section III presents the evolutionary algorithm combined with the developmental method, which was used for the design of arbitrarily large sorting networks. In Section IV, formal definitions of the basic terms related to the sorting networks and the proof of generality of the evolved design method are proposed. Section V discusses the properties of the evolved method and the resulting sorting networks. Concluding remarks and an inspiration for the future research are given in Sect. VI.

II. Sorting Networks: An Informal Overview

A sorting network (SN) is defined as a sequence of compare–swap operations (let's call them in short comparators) that depends only on the number of elements to be sorted, not on the values of the elements. A compare–swap of two elements $(a; b)$ compares and exchanges a and b so that we obtain $a \leq b$ after the operation.

The main advantage of the sorting network is that the sequence of compare–swap operations is fixed. Thus it is suitable for parallel processing and hardware implementation. Figure 1 shows the structure of a comparator in an example of a 3-input sorting network.

The number of compare–swap components and delay are two crucial parameters of sorting networks. A continuous sequence of comparators whose inputs and outputs are independent of each other can be grouped together to form so-called parallel layer. Comparators inside a parallel layer can

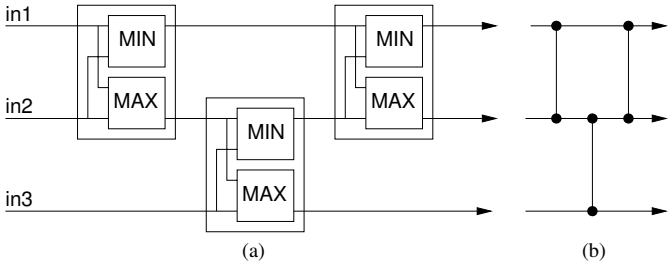

Fig. 1. (a) A three-input sorting network consists of three comparators. (b) Alternative symbol. This network can be described using the string [1;2][2;3][1;2].

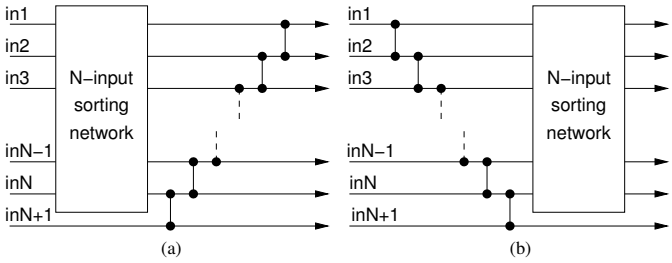

Fig. 2. Creating $(N+1)$-input sorter from an N-input sorter: (a) insertion principle, (b) selection principle

be executed in parallel. Delay of a sorting network is defined as the minimal number of its parallel layers, i.e. groups of comparators that are executed sequentially. Designers try to minimize the number of comparators, delay or both parameters.

In order to determine whether an N-input sorting network operates correctly, we should test $N!$ input combinations. However, by applying the zero–one principle, the number of test vectors can be reduced to 2^N. Zero-one principle says that if a sorting network sorts all the possible combinations of 0's and 1's correctly, then it sorts correctly every arbitrary sequence of values [1]. A sorting network is said to be valid if it sorts all the possible 2^N binary combinations correctly.

Sorting networks are usually designed for a fixed number of inputs. However, some conventional approaches exist to the design of arbitrarily large sorting networks. Figure 2 shows two basic principles for constructing $(N+1)$-input sorting network when an N-input sorting network is given [1].

Although the sorting networks created by means of the insertion or selection principle are much larger than the networks designed directly for a particular N, these methods can be treated as general design principles for building arbitrarily large sorting networks. In the next sections, we will show that it is possible to improve these approaches to obtain substantially better arbitrarily large sorting networks.

III. An Evolutionary Design Method Utilizing Development

Development is a biological process in which adult, multicellular organism is formed from a zygote. In the field of the evolutionary design, usually a simplified model of the development is applied to allow the target system to "grow" and increase its complexity. When a sort of development is included into an evolutionary algorithm, the chromosome has to contain a prescription for constructing a target object rather than a description of a target object itself.

A genetic algorithm has been used to design a *constructor* (a program) that is able to create a larger sorting network from a smaller sorting network (the smallest one is called the *embryo*). The constructor — directly represented by the chromosome — is a finite sequence of instructions, each of which is encoded as three integers: operation code, arg. 1 and arg. 2. Considering a comparator to be a basic building block of a sorting network, the following types of instructions have been chosen in order to manipulate the building blocks and enable the sorting network to increase its number of inputs, i.e. to "grow".

1) Modify-instructions (ModS, ModM). these instructions make a copy of a given comparator, take its indices of inputs, add up the instruction arguments respectively to the first and second index and perform the modulo-division of the modified indices by the number of inputs of the sorting network being constructed in order to ensure that the modified comparator belongs properly to the sorting network. Thereby a new comparator is created. Two variants of the Modify-instructions exist which differ in updating the pointer pointing to the comparator being processed (so-called the embryo-pointer – see the next two paragraphs): ModS does not update the pointer, i.e. the next instruction of the constructor processes the same comparator as the previous instruction did and the ModM instruction shifts the pointer to the following comparator of the sorting network.
2) Copy-instructions (CpS, CpM). Let w denote the number of inputs of the sorting network being created and k denote the first argument of the instruction. A Copy-instruction copies a sequence of $w-k$ comparators beginning with the comparator pointed by the embryo pointer (see the next two paragraphs). The second argument of the Copy-instructions is meaningless. While CpS instruction does not update the embryo pointer, the CpM instruction moves the pointer by $w-k$ comparators ahead.

Table I summarizes the instruction set utilized in this paper.

First, the constructor is applied to the embryo in order to build a larger sorting network. Then the same constructor is applied to this sorting network and larger circuit emerges again. This approach can be repeated so that the sorting network "grows" continually and infinitely (see Fig. 3). The goal is to obtain a valid sorting network after each constructor application. The constructor possessing this ability is said to be a *general constructor*. A single application of the constructor represents a *developmental step*. The difference between the number of inputs of two neighbouring sorting networks during the development denotes the *size of the developmental step*. Note that the sorting network resulted from an application of the constructor contains all the comparators of its predecessor. Fitness value is computed as the sum of the number of test vectors sorted correctly using the resulting SN after each developmental step. Since it is not possible to verify

TABLE I

INSTRUCTION SET UTILIZED IN THE DEVELOPMENT. LET $[c; d]$ BE A COMPARATOR. BY APPLYING AN INSTRUCTION TO IT, A NEW COMPARATOR $[c'; d']$ IS CREATED. EXPERIMENTS HAVE SHOWN TO BE USEFUL TO PUT $[c'; d']$ INTO THE NEWLY CREATED SORTING NETWORK ONLY IF $c' < d' \wedge c' < N \wedge d' \leq N$. N DENOTES THE NUMBER OF INPUTS OF THE EMERGING SORTING NETWORK, ip, ep AND np REPRESENT INSTRUCTION POINTER, EMBRYO POINTER AND NEXT-POSITION POINTER RESPECTIVELY.

Instruction	Arg1	Arg2	Description
0: ModS	a	b	$c' = c + a, d' = d + b, ip = ip + 1, np = np + 1$
1: ModM	a	b	$c' = c + a, d' = d + b, ip = ip + 1, np = np + 1, ep = ep + 1$
2: CpS	k	–	copy $N - k$ comparators, $ip = ip + 1, np = np + N - k$
3: CpM	k	–	copy $N - k$ comparators, $ip = ip + 1, np = np + N - k, ep = ep + N - k$
4: Nop	–	–	no operation (an empty instruction)

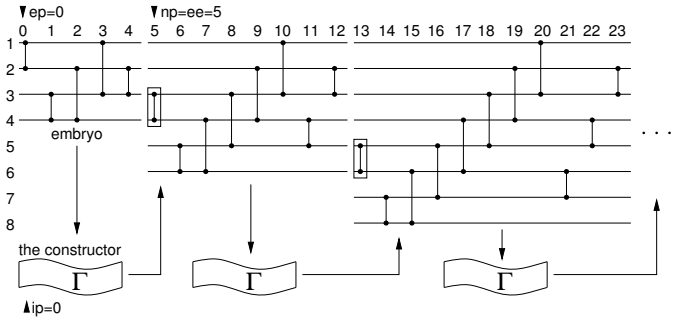

Fig. 3. The principle of designing larger sorting networks from smaller sorting networks by means of a constructor. The growth of the SN is illustrated using the best evolved constructor Γ=(ModS 2 2)(ModS 4 4)(ModS 3 4)(ModM 2 3)(ModM 2 0)(CpM 4 2)(ModS 2 2)(CpM 4 4). The initial configuration of pointers is shown.

all the sorting networks, only three developmental steps are considered for the fitness calculation. For instance, using a 4-input embryo and the developmental step of size 2, $Fitness = F(6)+F(8)+F(10)$, where $F(i)$ is the number of test vectors sorted correctly by the developed i-input sorting network. The generality of the evolved constructors is verified for the construction of up to 28-input sorting networks.

Figure 3 shows the principle of the developmental method and the growth of the sorting network using the best evolved constructor. A 4-input embryo was utilized. The instructions are executed sequentially; the instruction pointed by instruction pointer (ip) processes the comparator pointed by embryo pointer (ep) and the resulting comparator is placed to the first free position pointed by next-position pointer (np). Before an application of the constructor, the end of embryo poses the first free position in the sorting network being created. Let us denote it symbolically ee. After executing an instruction, the pointers are updated according to Tab. I. A developmental step is finished if the last instruction of the constructor is executed or if the end of embryo is reached ($ep = ee$), i.e. the constructor cannot process the comparators produced by itself in the actual developmental step. At the beginning of the developmental process, ep and ip are initialized to 0, ee and np are set to the first free position. After each developmental step, ip is initialized to 0, ep and np keep their values resulted from the last developental step and ee is set to the first free position.

The following settings of the evolutionary system was chosen with the parameters determined experimentally. The constructors were evolved as variable-length chromosomes in constant-length arrays whose maximal length was limited by 10 instructions. The instructions are encoded in the chromosome as a sequence of triples *operation code, argument 1, argument 2*. A special *Nop* instruction is utilized in order to allow the chromosome to vary its effective length (see Tab. I). A simple genetic algorithm was utilized with population size 60, tournament selection operator and one-point crossover operator with the crossover rate 0.7. Mutation operator was applied to each chromosome with the mutation probability 0.023 by random selection of an instruction in which either the operation code or one of its arguments is randomly mutated. The evolutionary algorithm is stopped if a constructor is found which creates fully functional sorting networks in three developmental steps or until 100000 generations passes. For the experiment presented in this paper 100 independent runs of the evolutionary developmental system were performed of which 40 general constructors were evolved.

The sorting networks produced by the best evolved constructor contain redundant comparators (in Fig. 3, the marked comparators at positions 5 and 13). However, these comparators can be removed from the SN without any loss of its functionality. After their removal, the delay of sorting networks is reduced substantially and, moreover, they require less comparators in comparison with conventional SNs. Table II and Fig. 4 survey the properties of the evolved sorting networks in comparison with the conventional insertion/selection principle.

In fact, the presented system was able to design a wide variety of structures of growing sorting networks [7]. Moreover, polymorphic[1] sorting networks have been discovered which sorts the input sequence into the non-decreasing order in one mode and into the non-increasing order in another mode [11]. In the next section, we focus on analyzing the best evolved constructor presented herein.

IV. IS THE EVOLVED CONSTRUCTOR GENERAL?

The previous section has demonstrated a rare case in which EA discovered a new design principle (rather than a single solution!). It remains to prove that the evolved constructor is

[1] A polymorphic circuit is a special kind of electronic circuit whose behavior depends on some external conditions, e.g. temperature, light, Vdd, external voltage etc., while the circuit structure remains unchanged, i.e. the circuit is inherently multifunctional [9], [10].

TABLE II
THE NUMBER OF COMPARATORS AND DELAY OF THE SORTING NETWORKS CREATED BY MEANS OF THE EVOLVED CONSTRUCTOR IN COMPARISON WITH THE CONVENTIONAL INSERTION OR SELECTION ALGORITHM

#inputs	6	8	10	12	14	16	18	20	22	24	26	28
Evolved SN: the number of comparators	13	24	38	55	75	98	124	153	185	220	258	299
Evolved SN: the number of redundant comparators	1	2	3	4	5	6	7	8	9	10	11	12
Evolved SN: the num. of comps. after removing the redundant ones	12	22	35	51	70	92	117	145	176	210	247	287
Conventional SN: the number of comparators	15	28	45	66	91	120	153	190	231	276	325	378
Evolved SN: delay	7	12	17	22	27	32	37	42	47	52	57	62
Evolved SN: delay after removing the redundant comparators	6	9	12	15	18	21	24	27	30	33	36	39
Conventional SN: delay	9	13	17	21	25	29	33	37	41	45	49	53

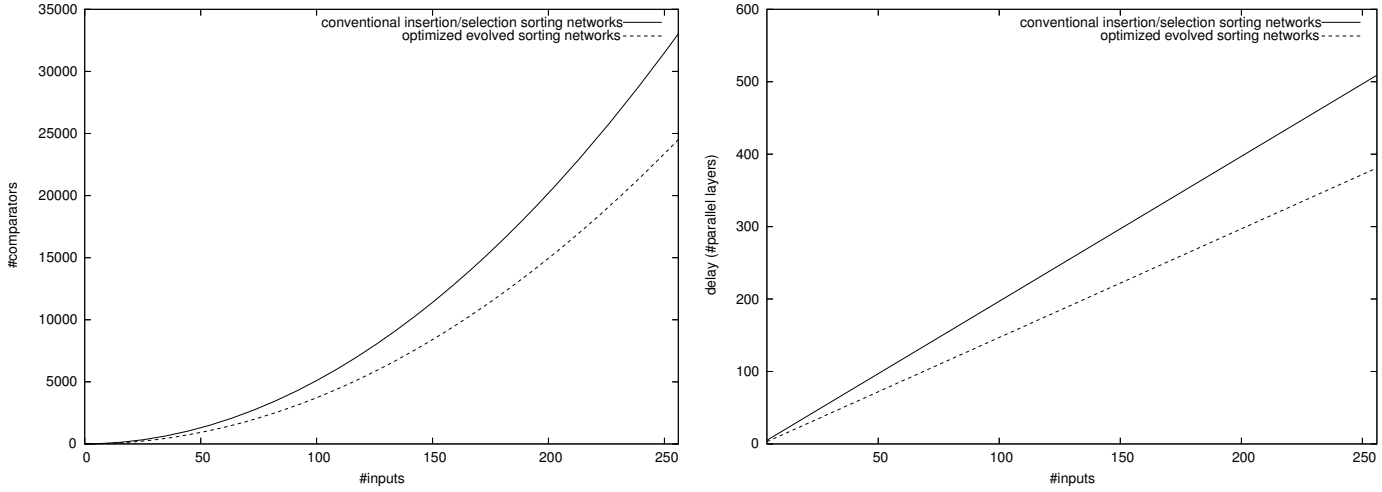

Fig. 4. Area- and delay-complexity of the optimized evolved sorting networks in comparison with the conventional insertion/selection principle

really a general constructor. In order to do that the following definitions have been formulated [8].

Definition 1: Let $I = \{1, 2, \ldots, N\}$ be an index set and let A be a set with an order relation $\leq$. A data sequence is a mapping $a : I \rightarrow A$. The set of all data sequences of length N over A is denoted by A^N.

Definition 2: The sorting problem consists of reordering an arbitrary data sequence $a_1, a_2, \ldots, a_N$, $a_i \in A$ for $i = 1, 2, \ldots, N$, to a data sequence $a_{\Phi(1)}, a_{\Phi(2)}, \ldots\ a_{\Phi(N)}$ such that $a_{\Phi(i)} \leq a_{\Phi(j)}$ for $i < j$, where Φ is a permutation of the index set $I = \{1, 2, \ldots, N\}$.

The definition of a sorting network is based on comparator networks introduced in [1]. Herein we consider a comparator $[i; j]$ as a circuit element that sorts the i-th and the j-th element of a data sequence into the nondecreasing order.

Definition 3: A comparator is a mapping $[i; j] : A^N \rightarrow A^N$, $i, j \in \{1, 2, \ldots, N\}$, where $[i; j](a)_i = min(a_i, a_j)$, $[i; j](a)_j = max(a_i, a_j)$, $[i; j](a)_k = a_k$ for all $k \neq i$, $k \neq j$, $i < j$ and for all $a \in A^N$.

We can simplify the formal notation of the comparator as follows. Let $[i; j]$ be a comparator applied to a data sequence $a \in A^N$. If $[i; j](a)_i = a_j$ and $[i; j](a)_j = a_i$, then we say that the comparator $[i; j]$ swaps a_i with a_j in a.

Definition 4: A parallel layer, S, is a composition of comparators $S = [i_1; j_1] \cdot [i_2; j_2] \cdot \ldots \cdot [i_k; j_k]$, $k \geq 0$ such that i_r and j_s are distinct for all $i_r = 1, \ldots, N-1, j_s = 2, \ldots, N, i_r \neq j_s$, for all $r = 1, \ldots, k, s = 1, \ldots, k, r \neq s$. Comparators within a parallel layer are executed in parallel.

Definition 5: A comparator network is a composition of parallel layers.

Note that the orientation of a comparator in a comparator network is important. We assume that every $[i; j]$ satisfies $i < j$. That being supposed, if $a_i > a_j$, then $[i; j]$ swaps a_i with a_j in $a \in A^N$. Moreover, the order of parallel layers in a comparator network is important since it defines the reordering algorithm. However, the order of the comparators within a parallel layer is not important because they are independent of each other.

Definition 6: A sorting network is a comparator network that sorts all the data sequences correctly.

Because of uniformity in terminology with respect to the previous sections, we will use the term sorting network in the rest of the paper even before proving the correctness of the appropriate comparator network.

Definition 7: Let S be a sorting network with even number of $N = 2k$ inputs, $k \geq 0$. We define k to be the degree of S.

Since we are applying the zero–one principle for testing the correctness of the sorting networks, the elements of a data sequence contain only binary values, i.e. $a_i \in \{0, 1\}$ for all $i = 1, \ldots, N$.

By means of the evolved constructor from Fig. 3, we are able to create 6-input sorting network from a 4-input sorting network, 8-input sorting network from a 6-input sorting network and so on by appending comparators into the preceeding solution. In general, we extend N-input sorting network to $(N + 2)$-input sorting network in each developmental step.

Since we have required the sorting networks to be correct after each developmental step (i.e. to be able to sort all possible data sequences), we can generalize this approach for arbitrary even-input sorting network and formulate the following theorem:

Theorem 1: Every arbitrary k-degree sorting network can be used as a base for constructing $(k+1)$-degree sorting network by appending $3k+1$ comparators, specifically $[2k+1; 2k+2]$, $[2k; 2k+2]$, $[2k-1; 2k+1]$, ..., $[1; 3]$ and $[2; 3]$, $[4; 5]$, ..., $[2k; 2k+1]$.

Proof: We make the proof by induction on the degree of sorting network, i. Recall that zero–one principle is applied in this proof.

Basis

Let the degree $i = 0$. Increasing i by one, we get a 2-input sorting network containing just one comparator, specifically $[1; 2]$. Proof of its correctness follows directly from Def. 3.

Induction Hypothesis

Assume that Theorem 1 holds for all $i \leq k$, where k is a positive integer.

Induction Step

Consider an arbitrary k-degree sorting network and use Theorem 1 to create $(k+1)$-degree sorting network. According to the induction hypothesis, the obtained sorting network is correct. Let z denote the number of 0's contained in the data sequence $a_1 a_2 \ldots a_{2k}$. Since the k-degree sorting network is correct, it produces non-decreasing data sequence of z 0's followed by $2k-z$ 1's. We have to prove that the comparators $[2k+1; 2k+2]$, $[2k; 2k+2]$, $[2k-1; 2k+1]$, ..., $[1; 3]$ and $[2; 3]$, $[4; 5]$, ..., $[2k; 2k+1]$ appended by the constructor are able to put all the possible binary combinations of elements a_{2k+1} and a_{2k+2} of the data sequence into their proper positions. The situation is illustrated in Fig. 5.

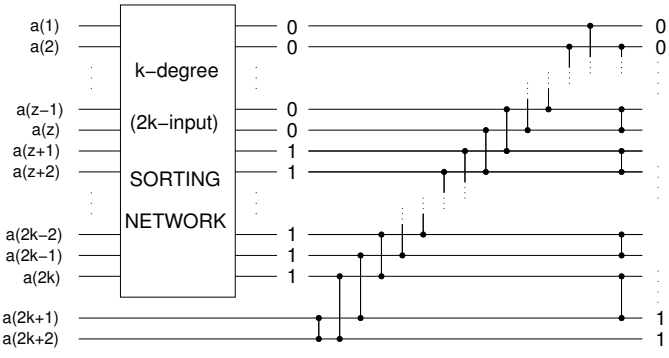

Fig. 5. The principle of creating $(k+1)$-degree sorting network from a k-degree sorting network

a) $a_{2k+1} = 0$ and $a_{2k+2} = 0$
The situation is illustrated in Fig. 6a. Consider $0 \leq z < 2k$. Observe that comparators $[2k; 2k+2]$, $[2k-1; 2k+1]$, ..., $[z+1; z+3]$ successively swap the input values processing zero-elements a_{2k+1} and a_{2k+2} of the data sequence. There are only zero-inputs in comparators $[z; z+2]$, $[z-1; z+1]$, ..., $[1; 3]$ so their execution has no effect. Similarly, none of comparators $[2; 3]$, $[4, 5]$, ..., $[2k; 2k+1]$ needs to swap its inputs. If $z = 2k$, then the data sequence has been already sorted.

b) $a_{2k+1} = 0$ and $a_{2k+2} = 1$
The situation is illustrated in Fig. 6b. Let $0 \leq z < 2k$. Comparators $[2k-1; 2k+1]$, $[2k-3; 2k-1]$, ..., $[z+(z \ mod \ 2)+1; z+(z \ mod \ 2)+3]$ successively swap the input values processing zero-element a_{2k+1} of the data sequence. If z is odd and $0 < z < 2k$, then the comparator $[z+1; z+2]$ swap its inputs to finish the resulting order of the data sequence. If $z = 2k$, then the data sequence has been already sorted.

c) $a_{2k+1} = 1$ and $a_{2k+2} = 0$
Consider $0 \leq z < 2k$. Observe that after swapping a_{2k+1} with a_{2k+2} by the comparator $[2k+1; 2k+2]$, the sorting proceeds in the same way as in the case (b); consider Fig. 6b with $a_{2k+1} = 1$ and $a_{2k+2} = 0$.

d) $a_{2k+1} = 1$ and $a_{2k+2} = 1$
In this case, the data sequence has been already sorted.

Observe that for every combination of values a_{2k+1} and a_{2k+2}, the $(k+1)$-degree sorting network has processed all the data sequences correctly, i.e. the evolved approach is general. ■

V. Discussion

The proposed method is based on the same idea as the conventional insertion or selection principle – creating a larger sorting network from a smaller one by appending some comparators. Unlike the insertion or selection algorithm, the evolved program constructs $(n+2)$-input sorting network from an even-n-input sorting network. Thus there are some restrictions in the design process in comparison with the conventional approaches. However, the sorting networks obtained by means of the evolved approach are substantially better in terms of both the number of comparators and delay than the conventional SNs (see Tab. II). For example, we can construct 18-input sorting network from the best known 16-input sorting network and the resulting network exhibits better properties than the network created by means of insertion or selection principle. Equations (1), respective (2) express the area complexity of the evolved, respective conventional sorting networks and equations (3), respective (4) express the delay complexity of the evolved, respective conventional sorting networks depending on the number of inputs N, $N \leq 4$. Although the asymptotic complexities are identical, the evolved algorithm constructs sorting networks with better properties (the number of comparators and delay) than the conventional insertion or selection principle. As we have proved, the evolved approach forms an improved general method which is similar to the conventional insertion or selection principle. Although it is focused primarily on the construction of even-input sorting networks, the resulting circuits can be reduced to odd-input sorting networks, which retain better properties in comparison with conventional SNs as demonstrated in [7].

$$C(N)_{\text{evol}} = \frac{3}{8}N^2 - \frac{1}{4}N = \mathcal{O}(N^2) \quad (1)$$

$$C(N)_{\text{conv}} = \frac{1}{2}N^2 + N = \mathcal{O}(N^2) \quad (2)$$

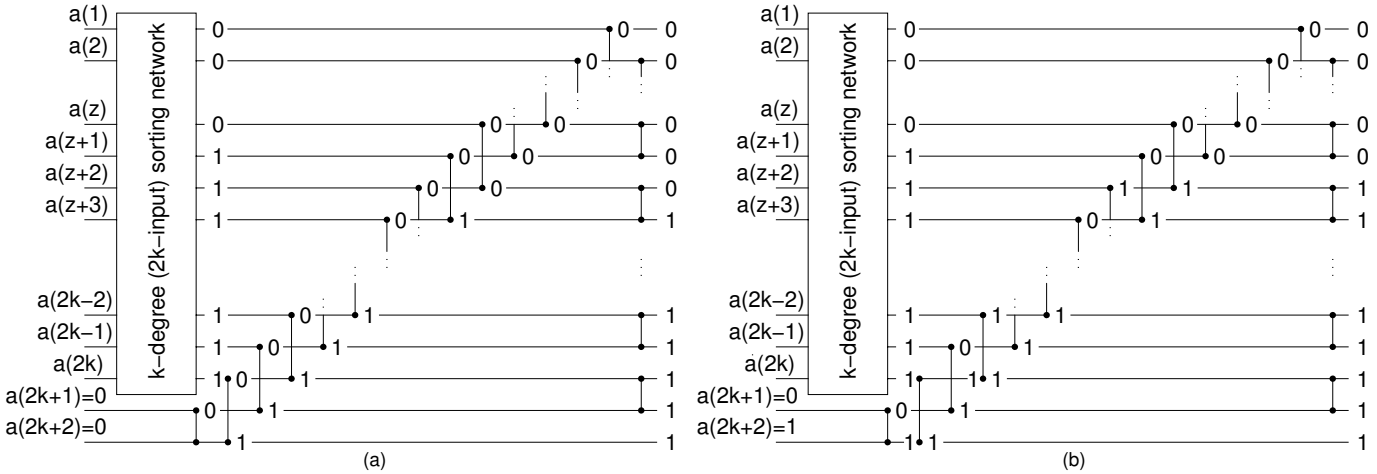

Fig. 6. Processing data sequences: (a) $a_{2k+1} = 0, a_{2k+2} = 0$, (b) $a_{2k+1} = 0, a_{2k+2} = 1$

$$D(N)_{\text{evol}} = \frac{3}{2}N - 3 = \mathcal{O}(N) \tag{3}$$

$$D(N)_{\text{conv}} = 2N - 3 = \mathcal{O}(N) \tag{4}$$

VI. Conclusions

In the paper, an improved general method has been presented for the construction of arbitrary even-input sorting networks, which was discovered by means of a genetic algorithm combined with an application-specific development. In contrast to the conventional method, the evolved algorithm utilizes a greater developmental step and more complex arrangement of the comparators. Similarly to human inventions in the area of theoretical computer science, the evolved invention was analyzed: its generality was proven and area and time complexities were determined.

We suppose that similar construction methods exist at least for the size of the developmental step equal to a power of two. According to the results presented herein, it could be expected that the greater developmental step the better properties of resulting sorting networks. These issues form the basic hypotheses for our future research.

Acknowledgment

The research was performed with the support of the Grant Agency of the Czech Republic under No. 102/06/0599 *Methods of Polymorphic Digital Circuit Design* and No. 102/05/H050 *Integrated Approach to Education of PhD Students in the Area of Parallel and Distributed Systems*.

References

[1] Knuth, D. E.: The Art of Computer Programming: Sorting and Searching (2nd ed.). Addison Wesley, 1998
[2] Juillé, H.: Evolution of Non-Deterministic Incremental Algorithms as a New Approach for Search in State Spaces.In: Proc. of 6th Int. Conference on Genetic Algorithms, Morgan Kaufmann, 1995, p. 351–358
[3] Choi, S., Moon, B.: A Hybrid Genetic Search for the Sorting Network Problem with Evolving Parallel Layers. In: Genetic and Evolutionary Computation Conference, San Francisco, 2001, p. 258–265
[4] Hillis, W. D.: Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure. Physica D 42, 1990, p. 228–234
[5] Choi, S., Moon, B.: More Effective Genetic Search for the Sorting Network Problem. In: Genetic and Evolutionary Computation Conference, New York, 2002, p. 335–342
[6] Koza, J. R. et al.: Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers, San Francisco, CA, 1999
[7] Sekanina, L., Bidlo, M.: Evolutionary Design of Arbitrarily Large Sorting Networks Using Development. Genetic Programming and Evolvable Machines. Vol. 6, Num. 3, 2005, p. 319–347
[8] Lang, H. W.: Algorithmen. Institut fűr medieninformatik und technische informatik. http://www.iti.fh-flensburg.de/lang/algorithmen/algo.htm (March 2006)
[9] Stoica, A. et al.: Polymorphic Electronics. Proc. of International Conferencee on Evolvable Systems: From Biology to Hardware, LNCS 2210, Springer, 2001, p. 291–302
[10] Stoica, A. et al.: On Polymorphic Circuits and Their Design Using Evolutionary Algorithms. Proc. of IASTED International Confernece on Applied Informatics, AI 2002, Innsbruck, AU, 2002
[11] Bidlo, M., Sekanina, L.: Providing Information from the Environment for Growing Electronic Circuits Through Polymorphic Gates. Proc. of Genetic and Evolutionary Computation Conference – Workshops 2005, ACM, New York, US, 2005, p. 242–248