Evolutionary Design Space Exploration for Median Circuits

Lukáš Sekanina

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
e–mail: sekanina@fit.vutbr.cz

Abstract. This paper shows that it is possible to (1) discover novel implementations of median circuits using evolutionary techniques and (2) find out suitable median circuits in case that only limited resources are available for their implementation. These problems are approached using Cartesian genetic programming and an ordinary compare–swap encoding. Combining the proposed approaches a method is demonstrated for effective exploration of the design space of median circuits under various constraints.

1 Introduction

Starting in 1992 when Higuchi et al evolved multiplexers in a programmable logic array [5], the evolutionary circuit design has become an important part of applications of evolutionary computing. In contrast to evolvable hardware, evolutionary circuit design deals in principle with a static fitness function and its objective is to discover novel solutions automatically, possibly without an assistance of human designers. A number of innovative and fault-tolerant digital as well as analog circuits have been evolved and demonstrated up to now (see examples in [4, 14, 18]).

Sorting networks have recently been recognized as potentially suitable objects for the evolutionary design and optimization [6, 8]. They are also interesting from a hardware viewpoint because of their regular and combinational nature suitable for pipeline processing. For instance, Koza et al have used genetic programming to evolve a 7-sorting network directly in a field programmable gate array [11].

This paper deals with median circuits whose effective hardware implementations are crucial for high-performance signal processing. As far as we know, no research results are available dealing with their evolutionary design. Because the median circuits can easily be derived from sorting networks, it seems that their evolutionary design is useless. However, this paper shows that interesting median circuits can be evolved from scratch. The objective of this research is twofold: (1) to explore whether novel implementations of median circuits can be discovered using evolutionary techniques and (2) to find out suitable median circuits in case that only limited resources are available for their implementation. These problems will be approached using Cartesian genetic programming (which is applied in this context first time) and an ordinary compare–swap encoding (known from evolving sorting networks).

This paper is organized as follows. Sorting networks and median circuits are introduced in Section 2. Cartesian genetic programming is utilized for designing median circuits in Section 3. Section 4 presents the results obtained using the compare–swap encoding. Section 5 deals with designing median circuits under hardware constraints. Finally conclusions are given in Section 6.

2 From Sorting Networks to Median Circuits

A *compare–swap* of two elements (a, b) compares and exchanges a and b so that we obtain $a \leq b$ after the operation. A sorting network is defined as a sequence of compare–swap operations that depends only on the number of elements to be sorted, not on the values of the elements [9].

Although a standard sorting algorithm such as quicksort usually requires a lower number of compare operations than a sorting network, the advantage of the sorting network is that the sequence of comparisons is fixed. Thus it is suitable for parallel processing and hardware implementation, especially if the number of sorted elements is small. Figure 1 shows an example of a sorting network.

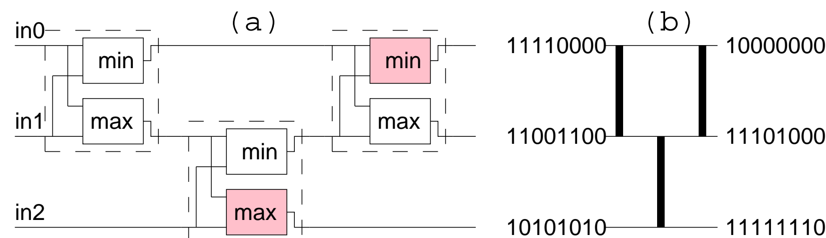

Fig. 1. (a) A 3-sorting network consists of 3 components, i.e. of 6 subcomponents (elements of maximum or minimum). A 3-median network consists of 4 subcomponents. (b) Alternative symbol. This sorting network can be tested in a single run if 2^3bits can be stored in a single data unit.

Having a sorting network for N inputs, the *median* is simply the output value at the middle position (we are interested in odd N only in this paper). For example, efficient calculation of the median value is important in image processing where median filters are widely used with N = 3x3 or 5x5 [14].

The number of compare–swap components and the delay are two crucial parameters of any sorting network. Since we will only be interested in the number of compare–swap components in this paper (we will deal with delay in future research), the following Table 1 shows the number of components of some of the best currently known sorting networks, i.e. those which require the least number of components for sorting N elements. Some of these networks (N = 13–16) were discovered using evolutionary techniques [1, 6, 8, 11]. The evolutionary approach

was based on the encoding that will be described in Section 4. Evolutionary techniques were also utilized to discover fault-tolerant sorting networks [15].

Note that the compare–swap consists of two subcomponents: maximum and minimum. Because we need the middle output value only in the case of the median implementation, we can omit some subcomponents (dead code at the output marked in gray in Fig. 1) and so to reduce the implementation cost in hardware. Hence in the case of K components, we can obtain $2K - N + 1$ subcomponents (Table 1, line 3 with median*).

However, in addition to deriving median networks from sorting networks, specialized networks have been proposed to implement cheaper median networks. Table 1 (line 2) also presents the best-known numbers of subcomponents for optimal median networks. These values are derived from the table on page 226 of Knuth's book [9] and from papers [2, 10, 19]. Note that popular implementation of the 9-median circuit in an FPGA proposed by Smith is also area-optimal (in terms of the number of components) [16].

Table 1. Best known minimum-comparison sorting networks and median networks for some N. $c(N)$ denotes the number of compare–swap operations, $s(N)$ is the number of subcomponents. The last line holds for median networks derived from sorting networks using dead code elimination.

N	3	4	5	6	7	8	9	10	11	12	13	14	15	16	25
sortnet, $c(N)$	3	5	9	12	16	19	25	29	35	39	45	51	56	60	144
median, $s(N)$	4	-	10	-	20	-	30	-	42	-	> 52	-	> 66	-	174
median*, $s(N)$	4	-	14	-	26	-	42	-	60	-	78	-	98	-	264

The *zero–one* principle helps with evaluating sorting networks (and median circuits as well). It states that if a sorting network with N inputs sorts all 2^N input sequences of 0's and 1's into nondecreasing order, it will sort any arbitrary sequence of N numbers into nondecreasing order [9]. Furthermore, if we use a proper encoding, on say 32 bits, and binary operators AND instead of minimum and OR instead of maximum, we can evaluate 32 test vectors in parallel and thus reduce the testing process 32 times. Figure 1 illustrates this idea for 3-median. Note that it is usually impossible to obtain the general solution if only a subset of input vectors is utilized during the evolutionary design [7].

3 CGP for Designing Median Circuits

As far as we know, Cartesian Genetic Programming (CGP) has not been utilized to evolve median circuits or sorting networks yet. Our initial hypothesis is that novel and more efficient median circuits can be evolved using smaller building blocks (such as subcomponents minimum and maximum) instead of using compare–swap components.

3.1 Cartesian Genetic Programming

Miller and Thomson have introduced CGP that has recently been applied by several researchers especially for the evolutionary design of combinational circuits [12]. In CGP, the reconfigurable circuit is modeled as an array of u (columns) $\times$ v (rows) of programmable elements (gates). The number of circuit inputs and outputs is fixed. Feedback is not allowed. Each gate input can be connected to the output of some gate placed in the previous columns or to some of circuit inputs. L-back parameter defines the level of connectivity and thus reduces/extends the search space. For example, if L=1 only neighboring columns may be connected; if L=u, the full connectivity is enabled. The designer has to define for a given application: the number of inputs and outputs, L, u, v and the set of functions performed by programmable elements. Figure 2 shows an example and a corresponding chromosome. Miller and Thomson originally used a very simple variant of evolutionary algorithm to produce configurations for the programmable circuit [12]. Our algorithm is based on this evolutionary technique.

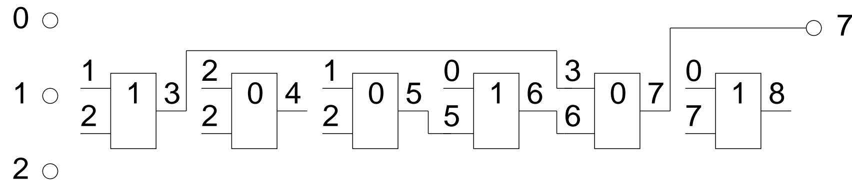

Fig. 2. An example of evolved 3-median circuit. CGP parameters are as follows: $L = 6$, $u = 6$, $v = 1$, functions Minimum (0) and Maximum (1). Gates 4 and 8 are not utilized. Chromosome: 1,2,1, 2,2,0, 1,2,0, 0,5,1 3,6,0, 0,7,1, 7. The last integer indicates the output of the circuit.

3.2 Experimental Setup

In order to evolve N-median circuits we utilized CGP with the following setting: $u = p$, $v = 1$, $L = p$, N inputs and a single output. p is the number of programmable elements depending on the problem size (for example, $p = 23$ for $N = 7$). Each of elements can be configured to operate as logical AND (minimum) or logical OR (maximum). Thus median circuits are rather constructed from subcomponents than from compare–swap components.

The evolutionary algorithm operates with population of 128 individuals. According to CGP, every individual consists of $u \times v \times 3 + 1$ integers. Four best individuals are considered as parents and every new population is generated as their clones. Elitism is supported. The initial population is generated randomly; however, the circuits with the maximum number of utilized elements are preferred. The evolution was typically stopped (1) when no improvement of the best fitness value occurs in the last 50k generations, or (2) after 600k generations.

Mutation works as follows: either the circuit output is mutated with the probability 25% or one of the used elements is mutated – gate inputs are changed with the probability 85%, function is changed with the probability 15%. Only one mutation is performed per circuit; however, when the best fitness value stagnates for 10k generations, two mutations are employed. The proposed parameters were chosen as suitable after the experimental testing.

During fitness calculation all possible input combinations are supplied at the circuit inputs (i.e. 2^N vectors). The fitness value of every candidate circuit is incremented by one if the circuit returns the correct median value for a given input vector.

3.3 Results

Table 2 shows that it is very difficult to evolve a perfect median circuit (with the fitness value 2^N) for more than 5 inputs. Furthermore, we were not able to reach optimal $s(N)$ for $N = 9$ and 11. The algorithm usually traps in local optima, which is very close to the perfect fitness value 2^N. It seems that the circuit evolution landscapes exhibit vast neutrality – similarly to Yu's and Miller's observations for even-parity problem [17].

Table 2. Summary of experiments performed to evolve small median circuits using CGP. N – # of inputs; p – # of columns in CGP; runs – # of runs performed; perfect – # of runs leading to the perfect fitness; perfect+opt $s(N)$ – # of runs with the perfect fitness and the optimal $s(N)$; best $s(N)$ – best obtained $s(N)$; best known $s(N)$

N	p	runs	perfect	perfect+opt $s(N)$	best $s(N)$	best known $s(N)$
3	8	50	50	15	4	4
5	16	100	11	1	10	10
7	23	2000	22	1	20	20
9	50	2000	35	0	36	30
11	90	200	2	0	71	42
13	120	200	none	-	-	-

Table 2 indicates that it is necessary to utilize much more programmable elements (the column denoted as p) than the resulting circuit requires in order to evolve a circuit with the perfect fitness. This requirement on redundancy was initially discussed by Miller et al [13]. For instance, 50 elements were used to evolve the 36-element median circuit for $N = 9$. However, we performed 200 runs with $p = 43$ for $N = 9$, but no circuit has appeared with the perfect fitness 512. The best circuits evolved using CGP are depicted in Fig. 3.

CGP has not allowed us to discover median circuits with lower $s(N)$ than the best known solutions exhibit. Furthermore, the approach produced median circuits only for small N. On the other hand, the obtained circuits are (1) totally

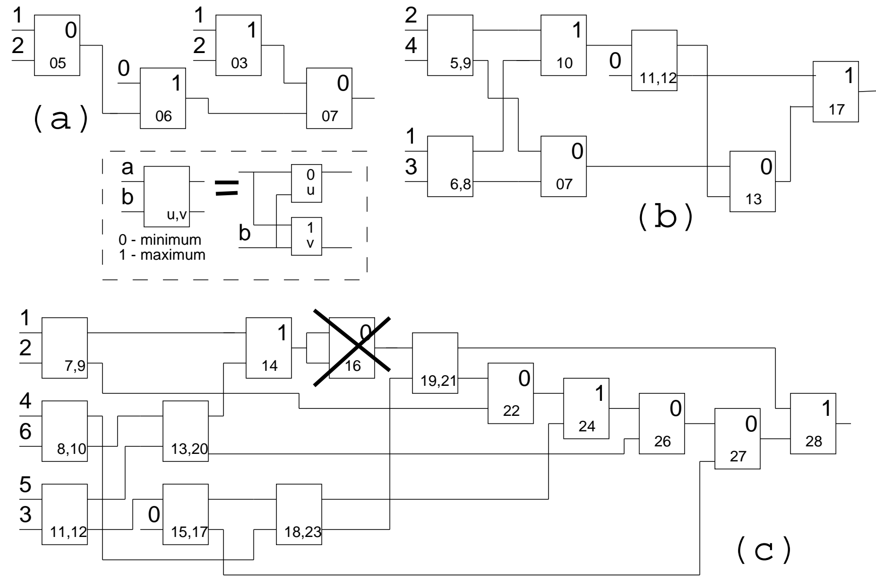

Fig. 3. Median circuits evolved using CGP: (a) 3-median, (b) 5-median and (c) 7-median circuits consist of the same number of subcomponents as the best–known conventional circuits. The evolved 7-median circuit contains one redundant subcomponent.

different from the known median circuits that are based on the compare–swap approach and (2) much cheaper than those median circuits derived from classical sorting networks (see Table 1, line 3 with median*).

4 A Compare-Swap Approach

The compare–swap approach is usually applied in order to evolve sorting networks. This approach works at the level of compare–swap components. Candidate solutions are represented as sequences of pairs (a, b) indicating that a is compared/swapped with b. More complicated representations have been developed in order to minimize the delay of the network (for instance, see [1]). We applied the compare–swap approach to evolve area-efficient median circuits.

4.1 Experimental Setup

Each chromosome consists of a sequence of integers that represents a median circuit. We used variable-length chromosomes of maximum length ml; a sentinel indicates the end of valid sequence.

A typical setting of the evolutionary algorithm is as follows. Initial population of 200 individuals is seeded randomly using alleles $0-(N-1)$. New individuals are generated using mutation (1 integer per chromosome). Four best individuals

are considered as parents and every newly formed population consists of their clones. The evolutionary algorithm is left running until a fully correct individual is found or 3000 generations are exhausted. We also increase mutation rate if no improvement is observable during the last 30 generations. Fitness calculation is performed in the same way as for CGP.

Because we would like to reduce the number of compare–swap components and because the fitness function does not consider the number of compare–swap operations, we utilized the following strategy. First, we defined a sufficient value ml for a given N (according to Table 1) and executed the evolutionary design. If a resulting circuit exhibits the perfect fitness, ml is decremented by 2, otherwise ml remains unchanged and the evolution is executed again. This is repeated until a predefined number of runs are exhausted. Because this approach works at the level of compare–swap components (i.e. $c(n)$), it was necessary to eliminate dead code to obtain $s(N)$.

4.2 Results

In contrary to CGP, perfectly operating median circuits were evolved up to $N = 25$ (see examples in Fig. 4). Table 3 shows that area-optimal circuits are up to $N = 11$. In spite of the best efforts of the author of this paper none specific values of $s(N)$ nor $c(N)$ were found in literature for median circuits with $N = 13 - 23$.

Table 3. The best median circuits evolved using the compare–swap approach

N	3	5	7	9	11	13	15	17	19	21	23	25
$c(N)$	3	7	13	19	26	34	44	55	65	80	94	109
$s(N)$	4	10	20	30	42	56	74	94	112	140	166	194
best known $s(N)$	4	10	20	30	42	?	?	?	?	?	?	174

While evolved median circuits are area-optimal for $N = 3 - 11$ and perhaps close to optimal for $N = 13 - 19$, the evolved median circuits for $N = 21 - 25$ seem to be area wasting. The reason is that the best known value $s(25)$ is 174 [2]; however, we reached $s(25) = 194$.

The evolution takes several days on a common PC for $N \geq 23$. Hence we tried to reduce the training set in our experiments; however, no solution has appeared producing correct outputs for all possible input combinations.

It was much easier to evolve perfect median circuits using the compare–swap encoding than using CGP. Similarly to the results obtained from CGP, the median circuits are not optimized for delay.

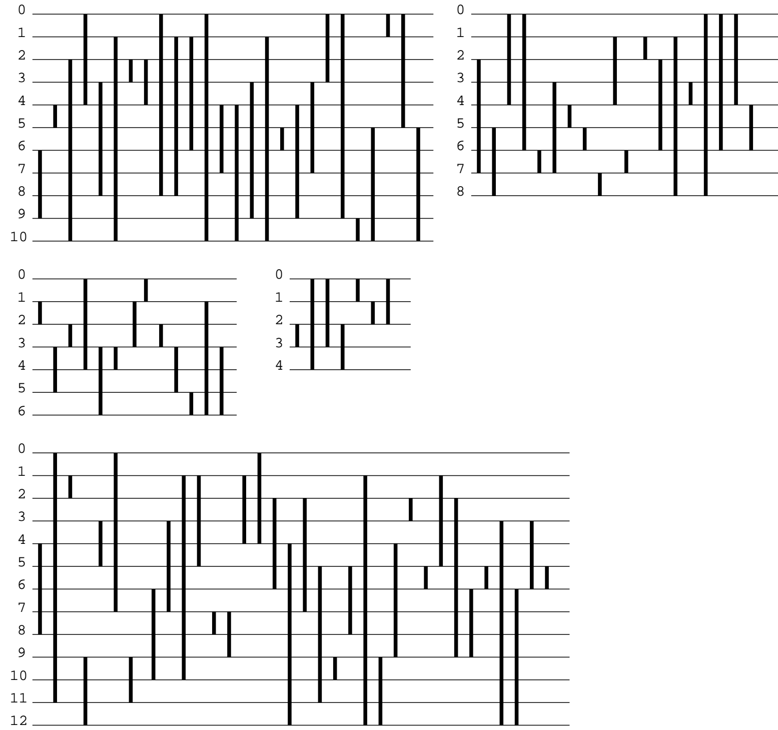

Fig. 4. Some of median circuits evolved using the compare–swap approach

5 Reducing Resources

The proposed evolutionary approach allows designers to explore much larger portion of the design space than conventional methods. This is demonstrated on the following experiment. The objective is to find out how many input combinations of all possible inputs lead to wrong output values if the number of circuit elements (compare–swap operations) is continually reduced. It is useful to know this characteristic from the fault tolerance point of view or when only limited resources are available for the implementation. Drechsler and Günther performed similar research for multiplexer circuits [3].

In order to evolve median circuits under hardware constraints, we utilized the experimental background from the previous section. Table 4 shows how reduction of the resources influences the number of outputs calculated correctly for median circuits of 5, 7 and 9 inputs. For instance, we need 13 compare–swap components to obtain the perfect fitness (128) for the 7-median circuit. If we reduce the number of components to 10, we obtain correct outputs for 122 input combinations. Using 5 components only, 102 out of 128 outputs are calculated correctly. It will probably be very difficult to design these “under-dimensioned” circuits by means of a conventional approach. Our method requires a few minutes

to find a solution using a common personal computer.

The method could be useful for designing area/energy-consumption efficient median filters for image preprocessing. In this application, most input combinations should be processed correctly. Rare mistakes are not critical because our eye is not able to see them.

Table 4. The number of correct calculations of output values for median circuits when reducing the number of compare–swap components $c(N)$ from 21 till 2

$c(N)$	2	3	4	5	6	7	8	9	10	11
5-median	24	26	27	28	30	32	32	32	32	32
7-median	90	96	98	102	104	108	112	116	122	126
9-median	346	366	372	384	390	402	410	420	430	442
$c(N)$	12	13	14	15	16	17	18	19	20	21
5-median	32	32	32	32	32	32	32	32	32	32
7-median	127	128	128	128	128	128	128	128	128	128
9-median	458	470	484	496	500	506	510	512	512	512

6 Conclusions

Two approaches for designing median circuits were proposed, implemented and analyzed in this paper. We were able to evolve better median circuits than those median circuits that can be created by means of dead code elimination from the best-known sorting networks. In some cases (up to N=11), the evolved median circuits are area-optimal. CGP has been applied in order to obtain median circuits directly without the need to eliminate same gates. However, the approach is suitable only for smaller N. Finally, we explored the design space of "reduced" median circuits and showed that these circuits operate correctly for most input vectors.

Combining the proposed approaches we provide a method for effective exploration of the design space of median circuits under various constraints.

Apart from hardware implementations of evolved circuits, our future research will be devoted to (1) designing larger median circuits by means of some developmental strategies, (2) investigating fault tolerance of evolved median circuits and (3) reducing latency.

Acknowledgment

The research was performed with the Grant Agency of the Czech Republic under No. 102/03/P004 *Evolvable hardware based application design methods* and No. 102/04/0737 *Modern Methods of Digital System Synthesis* and the Research intention No. MSM 262200012 – *Research in information and control systems.*

References

1. Choi, S. S., Moon, B. R.: More Effective Genetic Search for the Sorting Network Problem. In: Proc. of the Genetic and Evolutionary Computation Conference GECCO'02, Morgan Kaufmann, 2002, p. 335–342
2. Devillard, N.: Fast Median Search: An ANSI C Implementation. 1998 http://ndevilla.free.fr/median/median/index.html
3. Drechsler, R., Günther, W.: Evolutionary Synthesis of Multiplexer Circuits under Hardware Constraints. In: Proc. of the GECCO Workshop on Genetic Programming and Evolvable Hardware, Morgan Kaufmann Publishers, San Francisco 2000, p. 513–518
4. Gordon, T., Bentley, P.: On Evolvable Hardware. Soft Computing in Industrial Electronics, Physica-Verlag, Heidelberg 2001, p. 279–323
5. Higuchi, T. et al.: Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In: Proc. of the 2nd International Conference on Simulated Adaptive Behaviour, MIT Press, Cambridge MA 1993, p. 417–424
6. Hillis, W. D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D 42 (1990) 228–234
7. Imamura, K., Foster, J. A., Krings, A. W.: The Test Vector Problem and Limitations to Evolving Digital Circuits. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, IEEE CS Press, 2000, p. 75–79
8. Juillé, H.: Evolution of Non-Deterministic Incremental Algorithms as a New Approach for Search in State Spaces. In Proc. of 6th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1995, p. 351–358
9. Knuth, D. E.: The Art of Computer Programming: Sorting and Searching (2nd ed.), Addison Wesley, 1998
10. Kolte, P., Smith, R., Su, W.: A Fast Median Filter Using AltiVec. In Proc. of the IEEE Conf. on Computer Design, Austin, Texas, IEEE CS Press, 1999, p. 384–391
11. Koza, J. R., Bennett III., F. H., Andre, D., Keane, M. A.: Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann, 1999
12. Miller, J., Thomson, P.: Cartesian Genetic Programming. In: Proc. of the 3rd European Conference on Genetic Programming, LNCS 1802, Springer Verlag, Berlin 2000, p. 121–132
13. Miller, J., Job, D., Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits – Part I. In: Genetic Programming and Evolvable Machines, Vol. 1(1), Kluwer Academic Publisher (2000) 8–35
14. Sekanina, L.: Evolvable Components: From Theory to Hardware Implementations. Natural Computing Series, Springer Verlag, Berlin 2003
15. Shepherd, R., Foster, J.: Inherent Fault Tolerance in Evolved Sorting Networks. In Proc. of GECCO 2003, LNCS 2723, Springer Verlag, 2003, p. 456–457
16. Smith, J. I.: Implementing Median Filters in XC4000E FPGAs. Xcell 23, Xilinx, 1996 http://www.xilinx.com/xcell/xl23/xl23_16.pdf
17. Yu, T., Miller, J.: Finding Needles in Haystacks Is Not Hard with Neutrality. In: Proc. of the 5th European Conference on Genetic Programming, Kinsale, Ireland, LNCS 2278, Springer-Verlag, 2002, p. 13–25
18. Zebulum, R., Pacheco, M., Vellasco, M.: Evolutionary Electronics – Automatic Design of Electronic Circuits and Systems by Genetic Algorithms. CRC Press, Boca Raton 2002
19. Zeno, R.: A reference of the best-known sorting networks for up to 16 inputs. 2003, http://www.angelfire.com/blog/ronz/