Evolutionary Approximation of Edge Detection Circuits

Petr Dvoracek and Lukas Sekanina

Brno University of Technology, Faculty of Information Technology,
IT4Innovations Centre of Excellence
Božetěchova 2, 612 66 Brno, Czech Republic
idvoracek@fit.vutbr.cz, sekanina@fit.vutbr.cz

Abstract. Approximate computing exploits the fact that many applications are inherently error resilient which means that some errors in their outputs can safely be exchanged for improving other parameters such as energy consumption or operation frequency. A new method based on evolutionary computing is proposed in this paper which enables to approximate edge detection circuits. Rather than evolving approximate edge detectors from scratch, key components of existing edge detector are replaced by their approximate versions obtained using Cartesian genetic programming (CGP). Various approximate edge detectors are then composed and their quality is evaluated using a database of images. The paper reports interesting edge detectors showing a good tradeoff between the quality of edge detection and implementation cost.

1 Introduction

Reduction of energy consumption is one of the key issues of current society. For example, widely popular battery-powered personal electronics requires energy-efficient computing to reduce the need for battery recharging and big data and supercomputing centers require energy-efficient computing to minimize their operation costs. In recent years, a new approach to reducing the energy consumption has been adopted—*approximate computing*. It exploits the fact that some applications are inherently*error resilient* which means that the errors in their outputs can safely be exchanged for energy consumption reduction. This is a typical feature of multimedia applications in which some errors are not recognizable because human perception capabilities are limited.

Edge detection is an important pre-processing step in advanced image processing applications such as feature detection and feature extraction. The goal of edge detection is to find sharp changes in image brightness. As edge detection is performed very often it makes sense to optimize its implementation. This paper deals with efficient circuit implementations of edge detection based on the Sobel edge detector.

Evolutionary computing has been employed to develop approximate implementations of existing circuits or to evolve approximate implementations from scratch. The objective of this paper is to propose and evaluate a method based

on evolutionary computing which will enable to approximate edge detection circuits. Rather than evolving approximate edge detectors from scratch, we propose to approximate key components of existing edge detectors. In particular, Cartesian genetic programming (CGP) is used to generate approximations of adders which are basic components of Sobel edge detectors. Various approximate edge detectors are then composed of the approximate adders and their quality is evaluated using a database of images. The implementation cost is measured as the number of used gates. It has been shown in the literature that this measure provides a good estimate of power consumption [1].

The rest of the paper is organized as follows. Section 2 surveys relevant work. The proposed method is presented in Section 3. Experimental results are reported in Section 4. Conclusions are given in Section 5.

2 Relevant work

2.1 Edge Detectors

The majority of edge detection methods is based on the computation of image gradients. These gradients are often estimated to reduce the computation requirements. The gradient magnitude is then compared with a predefined threshold and used as an indicator whether edges are present or not at an image point. Detailed description of various edge detection algorithms can be found in [2].

The Sobel operator is one of the most popular edge detectors. It uses two convolution kernels (each as a 3×3 pixel window) to estimate gradients in an image. Let A be the input image. The horizontal and vertical derivative approximations are computed as

$$X = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} * A, \qquad Y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix} * A,$$

where $*$ is the convolution operator. At each point of the image, the gradient magnitude is given by

$$G = \sqrt{X^2 + Y^2}. \tag{1}$$

In order to reduce the computational requirements, the gradient magnitude computed using the square root function is often replaced with a calculation of the absolute value, i.e.

$$G' = |X| + |Y|. \tag{2}$$

Edge detection algorithms have often been accelerated in hardware in order to meet real time constrains of a given application, see, for example, a fast stereo vision system in FPGA [3].

Figure 1 shows an example of a hardware implementation of Sobel edge detector which operates according to formula 2. In total, this edge detector consists of twenty NOT gates, twenty XOR gates, four 8-bit adders, four 9-bit

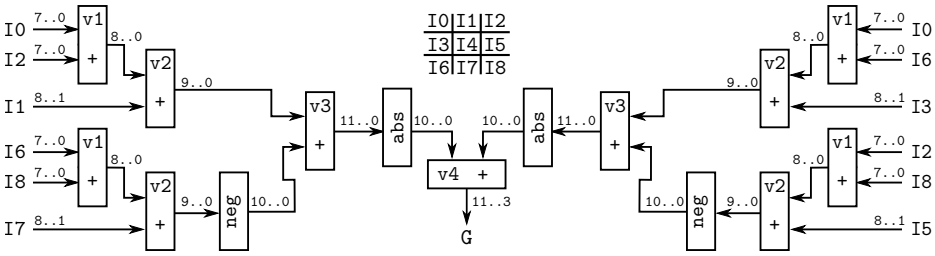


Fig. 1. A hardware implementation of Sobel edge detector. Symbols v denote the depth of addition. $I0, \ldots, I8$ are input pixels.

adders, and three 11-bit adders. It will be used as a reference implementation in this paper.

However, hardware implementations are often optimized to save valuable resources on a chip. In this case, the multiplication by two is implemented by arithmetic shifting. Subtraction is composed of adders v_3 and a set of inverters (neg). The absolute value is obtained by an inversion controlled by the most significant bit representing a negative sign. In other words, the inversion is implemented by an array of XOR gates in which one input of each XOR gate is connected to the most significant bit. The 9-bit and 11-bit adders were replaced by 8-bit adders. Furthermore, the multiplication of one operand is replaced with a division for the other operand, i.e. $(i_0+i_2)+2i_1$ was replaced by $(i_0+i_2)/2+i_1$. The size of operands of v_2 adders was decreased to 8 bits. For the other adders, the size of the operands was reduced by excluding less significant bits. This optimized version of the Sobel edge detector produces an insignificant error with respect to a solution operating exactly according to formula 2. In total, this edge detector contains sixteen NOT gates, sixteen XOR gates, and eleven 8-bit adders.

2.2 Evolutionary Computing in Edge Detector Design

Evolutionary computing has been utilized to design edge detectors since the nineties [4]. The current research on evolutionary computing for edge detection aims at evolving either edge detectors or edge features, where the features are functions of pixel values used in the process of classifying pixels as edge points or non-edge points [5]. Advanced concepts such as multi-objective methods [6] and Bayesian programs for features definition [5] have been integrated in to the EA-based design approaches. Evolutionary computing was also employed to evolve other computational models that can subsequently be employed for edge detection, for example, cellular automata [7].

Edge detectors and other image operators have been designed by Cartesian genetic programming (CGP). The method and representative case studies are surveyed in [8]. In the case of edge detector evolution, CGP evolves a solution using elementary two-input functions such as minimum, maximum and addition

operating over pixel values. The objective is to minimize the error function which is usually defined as a mean absolute error between the image generated by a candidate solution and a "golden image" in which all edges are ideally marked. The golden image is in practice obtained by a suitable conventional edge detector such as Canny or Sobel operator. The evolved circuit, in fact, approximates the conventional solution using hardware-friendly components.

In order to develop low-cost and efficient hardware implementations, edge detectors were evolved using gates and other hardware friendly components as building blocks. For example, an evolvable hardware approach was taken for low level edge detector design in [9] and genetic programming was used to search for digital transfer function of image edge detector [10]. The image filter evolution has been accelerated using specialized hardware such as graphics processing units [11] and field programmable gate arrays [12].

2.3 Approximate Computing in Image Processing

In recent years, approximate computing was established to investigate how computer systems can be made better—more energy efficient, faster, and less complex—by relaxing the requirement that they are exactly correct. One of the approximation techniques is functional approximation whose purpose is to implement a slightly different function to the original one providing that the error is acceptable and power consumption, performance or other parameters are improved. The functional approximation can be conducted at the level of software as well as hardware.

Image operators (including edge detectors) are good candidates for approximations because occasional errors (pixels showing undesired values) are not often recognized by users. On the other hand, the approximate implementations can lead to a reduction in power consumption or processing time.

In [13], a software module implementing the Sobel edge detector was replaced by trained neural network and the module has been later accelerated in a specialized hardware.

Using Axilog, which is a set of language annotations that provide the necessary syntax and semantics for approximate hardware design and reuse in Verilog [14], a conventional implementation of Sobel detector counting 143 lines of code was manually annotated (details not provided in the paper) and an approximate implementation was obtained. In both cases, appropriate papers report the quality of resulting images for a few target errors.

Image operators can also be approximated by approximating selected arithmetic components (adders or multipliers) that are present in conventional implementations, see denoising filters [15, 16]. Finally, approximations of median-outputting filters based on simplifying of the compare and swap components can be found in [17].

2.4 Evolutionary Circuit Design

After publishing a seminal paper introducing the field of evolvable hardware [18], new methods based on EAs have been proposed for circuit design. A considerable success have been achieved by Cartesian genetic programming which enabled to improve results of conventional circuit synthesis and optimization algorithms for small combinational circuits and discover new implementations of important circuit components such as filters, classifiers and predictors [19]. More complex circuits were then evolved by means of decomposition techniques [20], functional level evolution, developmental encodings, and advanced fitness evaluation methods utilizing the principles of formal verification [21].

Digital circuits can naturally be approximated by means of CGP. The methods can be classified as

- Error-oriented: In the first phase, CGP tries to evolve a circuit showing a predefined error. In the second phase, the resources are optimized.
- Resources-oriented: Resources (e.g. the number of gates) are constrained and CGP is used to minimize the circuit error with available resources.
- Multi-objective: All criteria are optimized together using a multi-objective EA such as NSGA-II.

Examples of approximate circuits obtained by CGP are approximate medians, 8-bit adders and 8-bit multipliers [1, 22]. All these circuits were approximated without any decomposition. In this work, we propose to approximate selected components of the whole circuit and analyze the impact of the approximation on the circuit behavior.

3 Adopting CGP for Circuit Approximation

CGP will be used to approximate selected components of conventional or evolved edge detection circuits. This section deals with the principles of CGP when applied to circuit evolution and approximation.

3.1 Cartesian Genetic Programming

Circuit Representation in the Chromosome: A candidate circuit is modeled as a grid of processing nodes arranged in n_c columns and n_r rows. Each processing node performs specific operation g from the set of functions Γ. In evolutionary circuit design, this function set usually contains logic gates or elementary arithmetic functions. The number of circuit inputs n_i and outputs n_o are fixed. Parameter l-back defines a degree of interconnection between the columns. For example, if $l = 1$ the interconnection is minimal because only neighboring columns may be connected; if $l = n_c$ the circuit interconnection is maximal. Nodes of the same column can not be connected.

The circuit connection is encoded into a chromosome. Each gate (processing node) is represented by a triplet (i_1, i_2, α), where α is a code of operation taken

from the function set Γ. Symbols i_1 and i_2 are pointers to nodes (or primary inputs) to which a given gate is connected to, providing that primary inputs are labeled $0 \dots n_i - 1$ and the nodes are labeled $n_i - 1 \dots n_i + n_c n_r - 1$. Finally, the chromosome contains n_o genes determining the nodes or logic constants (0 or 1) where the primary outputs are connected to.

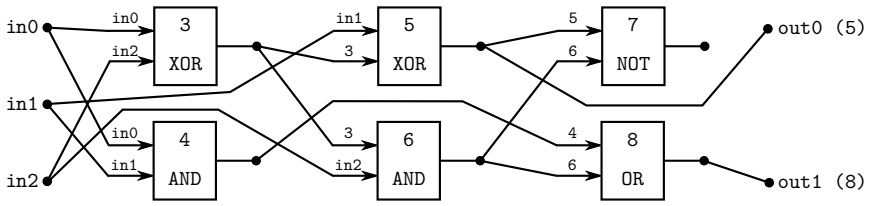


Fig. 2. Full adder represented by CGP with parameters: $n_i = 3, n_o = 2, n_c = 3$, $n_r = 2, l = 3, \Gamma = \{0^{AND}, 1^{OR}, 2^{XOR}, 3^{NOT}\}$. Chromosome: (0, 2, 2) (0, 1, 0) (1, 3, 2) (3, 2, 0) (5, 6, 3) (4, 6, 1) (5, 8)

Figure 2 provides an example of the CGP encoding. One important feature of CGP is that not all the nodes have to be included in the phenotype. In this case, NOT (node 7) is disconnected.

Fitness Function: As our goal is to approximate arithmetic circuits using the resources-oriented method, the fitness function is defined as a sum of absolute differences (SAD)

$$f_{SAD} = \sum_{j=1}^{K} |y(j) - t(j)|,$$

where y is candidate circuit's response, t is target response and K is the number of fitness cases. Because target circuits are arithmetic circuits, we have to evaluate circuit responses for all possible combinations of operands, i.e. $K = 2^{n_i}$. This definition of the fitness function is preferred over the Hamming distance as suggested in [23].

The proposed approximation method is constructed as a resources-oriented method, in which a good compromise is sought between the number of gates and the error. Resources (gates) can be constrained either by constraining the product ($n_c \times n_r < k$) or by constraining the number active gates in a potentially big array of gates. The first approach was utilized in the literature [22, 23]. In this work, the second approach is adopted as it enables CGP to operate with highly redundant arrays of gates which is beneficial for an efficient search [24]. Let the number of available gates be n_n and the number of gates in phenotype be n_{pn}. The fitness function is defined:

$$f = \begin{cases} f_{SAD}, & \text{if } n_{pn} \leq n_n \\ \infty, & \text{if } n_{pn} > n_n \end{cases}$$

Search Algorithm: CGP uses a (1+λ) search method consisting of the following steps:

1. An initial population of the size $1 + \lambda$ is created.
2. Each candidate circuit is evaluated by fitness function f.
3. The highest-scored candidate circuit is selected as a new parent. The parent from previous generation is never selected as the new parent if there is another individual with the same fitness value.
4. By applying a point mutation, λ offspring individuals are generated from the parent. Parameter h defines the number of genes (integers) that undergo a mutation.
5. Steps 2—4 are repeated until the termination condition is not satisfied.

Heuristic Population Seeding: In many cases a conventional exact solution (circuit C_0, with z gates) is available and can be used in the initial population of CGP. According to [22], a simple method can be employed in order to obtain the first approximation of C_0. We create $2z$ circuits in such way that every single gate of C_0 is independently replaced by a wire which connects gate's first or second input with its output. The circuit producing the smallest error out of these $2z$ approximations is taken as the first parent of CGP.

3.2 Resources-oriented Approximation

The proposed method should produce a Pareto front showing the best obtained compromises between the number of gates and the error. As our target circuits are relatively small (tens of gates), it makes sense to execute CGP multiple times and constraint the number of gates in each run to $z-1, z-2, \ldots, z_m$, where z is the number of gates in the exact solution C_0 and z_m is the smallest reasonable approximation of C_0. Each CGP run begins with the best approximate circuit obtained from the previous approximation and the objective is to minimize the circuit error for a given amount of gates.

4 Experimental Results

The adders are key components of edge detection circuits. Firstly, results of adder approximations are summarized in this section. Then, performance and cost of various edge detectors utilizing the approximate adders are reported.

4.1 Evolutionary Approximation of Adders

Computational requirements of CGP can significantly be reduced if the initial population is seeded by a conventional solution. In the case of adders, the carry ripple adder and the Kogge-Stone adder have been employed. The exact 8-bit carry ripple adder is composed of one half adder and seven full adders. In total, it consists of 37 two input gates. However, the carry propagates through 15

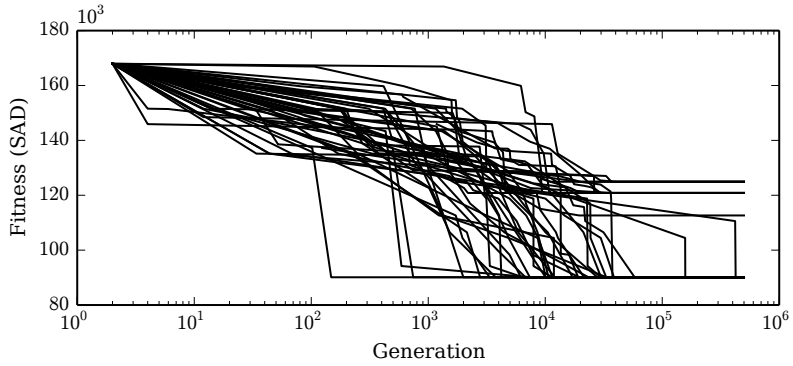


Fig. 3. Convergence curves for the adder composed of 41 gates in all 50 evolutionary runs

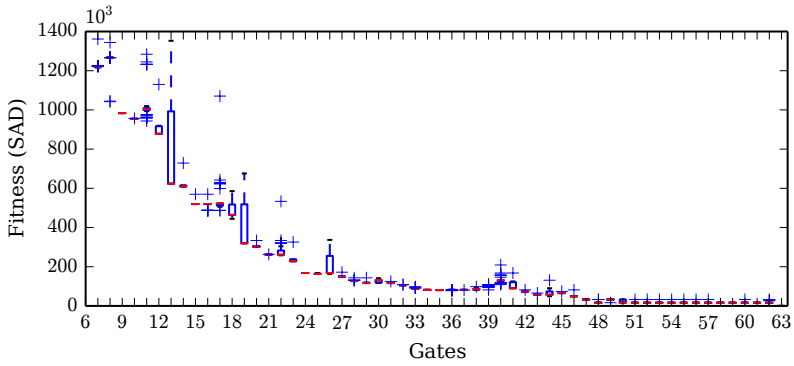


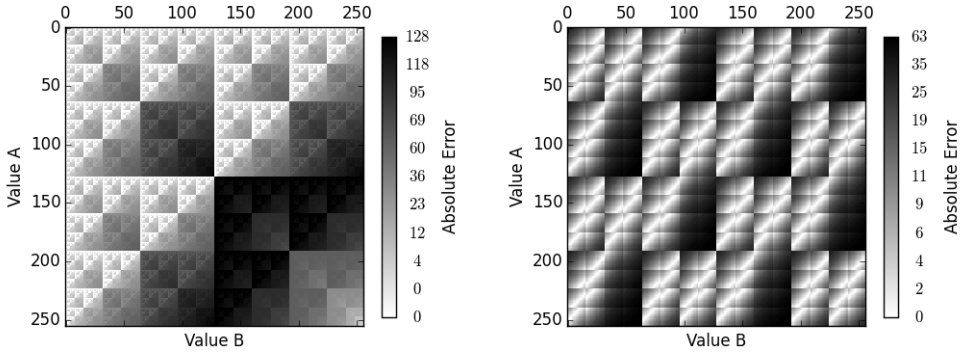
Fig. 4. Statistical evaluation of the evolutionary approximation of 8-bit Kogge-Stone adders using 6 – 63 gates.

gates and the corresponding delay of 15Δ (where Δ is delay of one gate) is undesirable for many applications. The 8-bit Kogge-Stone adder [25] exploits the carry lookahead logic. As the carry bits are computed in parallel, the resulting delay is only 7Δ. The cost is, however, higher – 73 gates.

The objective is to approximate the 8-bit Kogge-Stone adder. In order to keep the delay less or equal to 7Δ, CGP is used with $n_c = 7$. CGP is executed multiple-times with constrained resources to obtain a Pareto front. The CGP parameters are initialized as follows: $n_r = 13, n_c = 7$, $l = n_c$, and $\Gamma = \{$BUF, NOT, AND, OR, XOR, NAND, NOR, XNOR$\}$, where BUF stands for an identity function. The first runs are seeded with the circuit obtained by removing one gate from the exact adder, i.e. $n_n = 72$.

The parameters of evolution are set as follows: $h = 5\%, \lambda = 4$, and $n_g = 500000$. After 50 runs, the number of allowed gates n_n is decremented.

In order to demonstrate the progress of evolution, an adder constrained to utilize up to 41 gates is considered. Figure 3 shows convergence curves obtained



(a) Conventional Design (b) Evolutionary Design

Fig. 5. Error plots of approximate adders containing 9 gates.

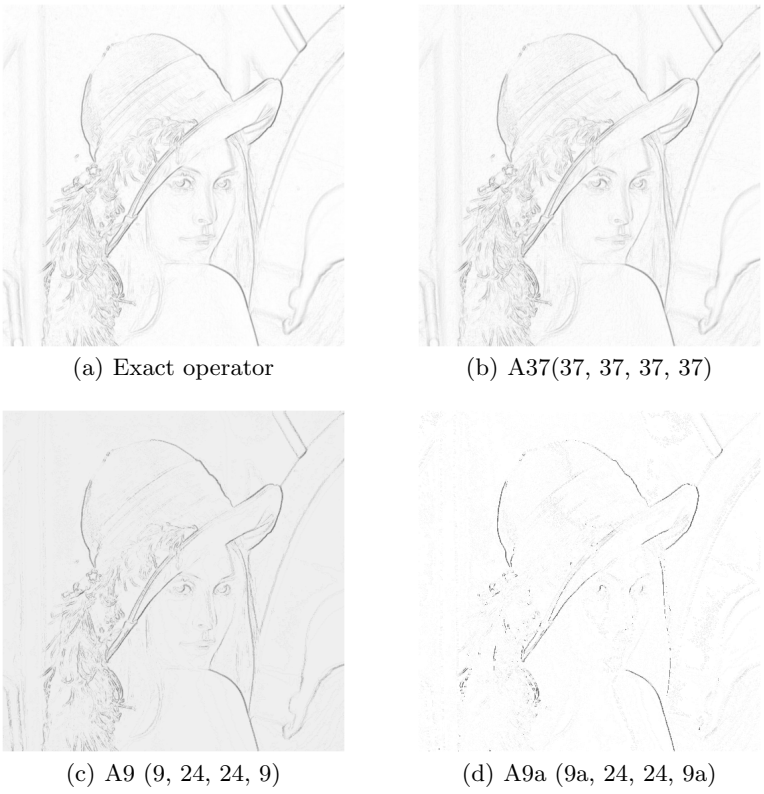
		Absolute Error					
Gates	Δ	Avg.	MIN	Q1	Q2	Q3	MAX
9^a	1	47.87	1	16	47	88	128
9	4	15.02	1	6	12	22	63
13	5	9.51	1	4	9	14	32
24	7	2.56	1	1	3	4	8
37	6	1.23	1	1	2	3	7
43	6	0.85	1	1	1	2	3
62	7	0.00	-	-	-	-	-

Table 1. Parameters of evolved approximate 8-bit adders and one conventional approximate 8-bit adder (9^a).

from 50 independent runs. Every run starts with the error $f = 167 \cdot 10^3$ and ends up with the error $f = 90 \cdot 10^3$ in most cases.

Boxplots summarizing the whole experiment ($n_n = 6, 7, \ldots, 63$ gates enabled) are plotted in Figure 4. The adders with less than 6 gates were omitted due their large error. It can be seen from the boxplots that the evolution converges to a single value in most cases. Moreover, we discovered a fully functional implementation of the 8-bit adder which contains fewer gates ($n_n = 62$) and features same latency as the 8-bit Kogge-Stone adder. A detailed analysis of the results revealed that a solution composed of 37 gates has delay 6Δ , which is more than two times smaller than the delay of the Carry Ripple Adder. The average error of this approximate adder is only 1.23 (Table 1).

In common conventional approximations, an 8-bit adder is often approximated by a very cheap implementation consisting of 8 OR gates and one AND gate (for the carry bit) [26]. This approximation can be compared with an evolved approximate 8-bit adder of the same implementation cost (i.e. 9 gates). The conventional adder exhibits bigger maximal, median, and average errors (Ta-



(a) Exact operator (b) A37(37, 37, 37, 37)

(c) A9 (9, 24, 24, 9) (d) A9a (9a, 24, 24, 9a)

Fig. 6. Edge detection by approximate Sobel detectors

ble 1). Figure 5 also shows that more significant errors occur in the case of larger operands. On the other hand, in the case of evolved approximate adder, more significant errors are spread out in the space of operand combinations.

4.2 Approximation of Sobel Edge Detector

Several approximations of the reference edge detector implementation from Fig. 1 are proposed in this Section. Firstly, some adders of the reference circuit were replaced by approximate 8-bit adders. The replacement is performed for a set of adders occupying the same depth (v_1, v_2, v_3, and v_4 in Fig. 1) of the adder tree. Let us denote an approximate edge detector by An_1 (n_1, n_2, n_3, n_4), where each element n_i represents the number of gates in adders of a given depth, e.g. A50 $(50, 50, 50, 50)$ is an edge detector containing at all levels approximate adders consisting of 50 gates.

The impact of approximate adders on edge detection is demonstrated in Figure 6. For example, by using the adders composed of 37 gates, we obtained a low-cost edge detector A37 $(37, 37, 37, 37)$ that produces a very small error

Table 2. MAE and other properties of approximate Sobel edge detectors

Edge Detector			Mean Absolute Error
Name	Gates	Δ	0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60
Eq. 2	813	33	
A62	714	30	
A37	439	26	
A13	291	27	
A9	221	24	
A9a	221	18	

Table 3. RMSE and other properties of approximate Sobel edge detectors.

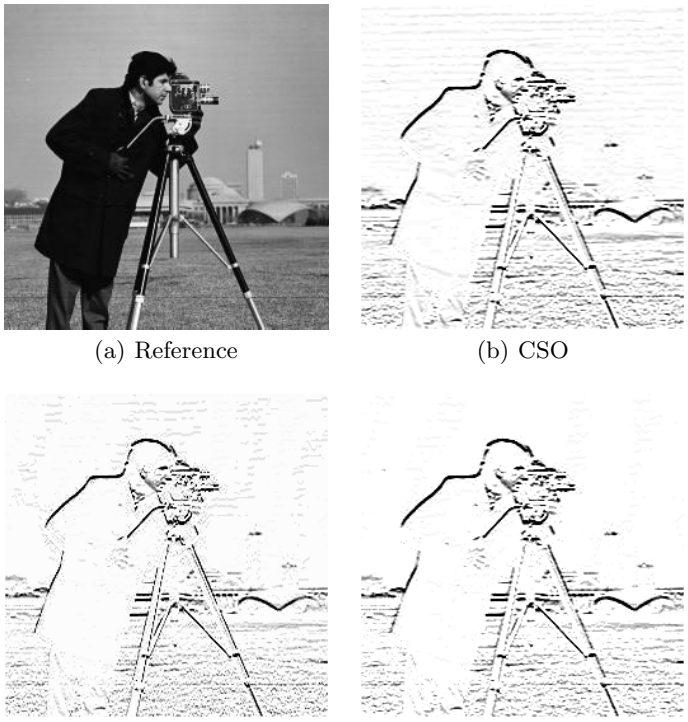
Edge detector			Root Mean Square Error
Name	Gates	Δ	0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60
Eq. 2	813	33	
A62	714	30	
A37	439	26	
A13	291	27	
A9	221	24	
A9a	221	18	

which is indiscernible to human eye. This approximate edge detector contains the same amount of gates as the edge detector composed of fully functional 8-bit carry ripple adders. However, the approximate detector is more than two times faster.

Figure 6 compares edge detector A9 $(9, 24, 24, 9)$ with A9a $(9, 24, 24, 9)$. Both solutions utilize the same number of gates. A9 (containing the adders evolved by CGP) shows more precise edge detection than A9a which employs the adders approximated conventionally. On the image of Lenna, A9 produces the mean absolute error (MAE) 19.84 per pixel which is bigger than the error of A9a — 13.53 per pixel. This result is also manifested by darker background of the image produced by A9. If the root mean square error (RMSE) is used as an error metric, the result is 20.84 for A9 which is better result than A9a (RMSE = 22.41).

We tested approximate edge detection circuits on a dataset containing 200 images. Tables 2 and 3 demonstrate the differences in terms of MAE and RMSE between a fully working detector operating exactly according to formula 2, a solution in which the adders containing operands having more than 8 bits are replaced with accurate 8-bit adders (A62), and other approximate detectors – A9, A9a, A13 (13, 24, 34, 43), and A37.

It is difficult to perform a direct comparison with approximate edge detectors published in the literature. The reason is that many circuit parameters and details of experiments are not published. A brief comparison can be done with



(a) Reference (b) CSO

(c) ESO24 (d) A24

Fig. 7. Edge detection by approximate Sobel Operators

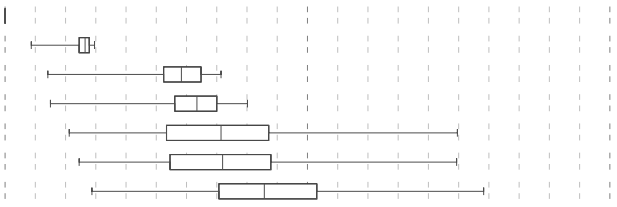
Axilog. The authors of Axilog reported a 1.82× area reduction for Sobel detector with RMSE=10% for a single image. The percentage value of RMSE was computed on the normalized image with pixels in the range $(0, 1)$. Under this metric, A9 detector obtained 3.89× area reduction with maximal RMSE = 6.2% using the dataset of 200 images [27]. Moreover, A9 detector reduced the area three times in comparison with A62 and almost twice in comparison with A37.

4.3 Approximation of Evolved Edge Operator

Paper [12] presents Sobel operator (CSO) implemented as:

```
uint8 CSO(uint8 kernel[9]) {
  int i;
  i = kernel[0] + 2*kernel[1] + kernel[2];
  i = i - (kernel[6] + 2*kernel[7] + kernel[8]);
  i = max(i, 0);
  i = min(i, 255);
  return i;
}
```

Table 4. Properties of approximate edge operator CSO and approximate evolved operator ESO.

Sobel operator			Root Mean Square Error
Name	Gates	Δ	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
CSO	411	30	
CSO62	358	28	
CSO37	233	25	
CSO24	188	24	
ESO62	506	73	
ESO37	381	68	
ESO24	336	69	

where kernel is the 3×3-pixel convolution window. Paper [12] also presents an edge detector (ESO), completely evolved by CGP. This operator used the image shown in Fig. 7 as a golden image in the fitness function. The evolved code of ESO is given below.

```
uint8 ESO(uint8 kernel[9]){
  uint i14, i17, i19, i22, i27, i29;
  i14 = min((kernel[1] + kernel[7]) >> 1, kernel[7]);
  i17 = i14 ^ kernel[7];
  i19 = min(i14 + (255 - kernel[1]), 255);
  i22 = 255 - i19;
  i27 = min(i22, (i17 + i19) >> 1);
  i29 = min(i22 + i27, 255);
  return (i27 + i29) & 0xff;
}
```

CSO operator employs just a part of the Sobel edge detector and it thus computes the horizontal derivative. The cost of CSO is five adders and an array of NOT gates. In order to obtain edges as shown in Figure 7, we saturated the output to 0 or to 255, if the output value is negative or greater than 255, respectively.

Both CSO and ESO can be approximated using the adders presented in Section 4.1. As CSO has only three levels of adders, we denote the approximate conventional operator CSOn_1 (n_1, n_2, n_3) where n_i represents the number of gates used in approximate adders at the level v_i.

Evolved operator ESO contains five adders which can be replaced by their approximate versions. Approximate detectors will be denoted ESOn_1 (n_1, n_2, n_3, n_4, n_5) where n_i represents the number of gates used in a given approximate adders.

ESO contains more gates than CSO mainly because there were no requirements on the area minimization in paper [12]. Table 4 gives RMSE calculated using 200 images for all approximate edge detectors.

It can be seen that RMSE is growing when CSO62 $(62, 62, 62)$ is compared with approximate CSO37 $(37, 37, 37)$ and CSO24 $(24, 24, 34)$. But for example, RMSE boxplots are almost identical in the case of ESO62 $(62, 62, 62, 62, 62)$, ESO37 $(37, 37, 37, 37, 37)$, and ESO24 $(24, 24, 24, 30, 28)$. It seems that the evolved solution is more robust to the approximation than conventional CSO. It also turns out that it is hard to predict the impact of approximations on the overall circuit behavior.

5 Conclusions

In this work, various approaches to the approximation of edge detectors based on the Sobel operator were proposed and evaluated. We replaced exact adders in conventional as well as evolved edge detectors by their approximate versions. The approximate adders were obtained using CGP. Results were reported in terms of the error (MAE and RMSE obtained using 200 test images) and the implementation cost given as the number of gates.

We showed that evolved approximate 8-bit adder composed of 9 gates has smaller average error than a commonly used approximation consisting of the same number of gates. Moreover, evolved inaccurate adder containing 37 gates and producing a very small average absolute error has $3\times$ smaller delay than a fully functional carry ripple adder composed of the same amount of gates. In the case of edge detection, we demonstrated a circuit showing $3.89\times$ area reduction with maximal RMSE=6.2%. It seems that evolved edge detectors are more resilient to approximations than conventional edge detectors.

Our future work will be devoted to a detailed analysis of the approximations not only at the circuit level but also at the whole system level.

Acknowledgements. This work was supported by the Czech science foundation project GA16-17538S.

References

1. Vasicek, Z., Sekanina, L.: Circuit approximation using single- and multi-objective cartesian GP. In: Proceedings of the 18th European Conference on Genetic Programming – EuroGP. LNCS 9025, Springer (2015) 217–229
2. Sonka, M., Hlavac, V., Boyle, R.: Image Processing: Analysis and Machine Vision. Thomson-Engineering (1999)
3. Ttofis, C., Hadjitheophanous, S., Georghiades, A., Theocharides, T.: Edge-directed hardware architecture for real-time disparity map computation. IEEE Transactions on Computers **62**(4) (2013) 690–704
4. Harris, C., Buxton, B.: Evolving edge detectors with genetic programming. In: Proceedings of the First Annual Conference on Genetic Programming. (1996) 309–314
5. Fu, W., Johnston, M., Zhang, M.: Genetic programming for edge detection using multivariate density. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, ACM (2013) 917–924

6. Zhang, Y., Rockett, P.I.: Evolving optimal feature extraction using multiobjective genetic programming: a methodology and preliminary study on edge detection. In: In: GECCO 05: Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM (2005) 795–802
7. Priego, B., Bellas, F., Souto, D., Lopez-Pena, F., Duro, R.: Evolving cellular automata for detecting edges in hyperspectral images. In: 2012 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), IEEE (2012) 1–6
8. Sekanina, L., Harding, L.S., Banzhaf, W., Kowaliw, T.: Image processing and CGP. Cartesian Genetic Programming (2011) 181–215
9. Hollingworth, G., Tyrrell, A., Smith, S.: Simulation of Evolvable Hardware to Solve Low Level Image Processing Tasks. In: Proc. of the Evolutionary Image Analysis, Signal Processing and Telecommunications Workshop. Volume 1596 of Lecture Notes in Computer Science., Springer-Verlag (1999) 46–58
10. Golonek, T., Grzechca, D., Rutkowski, J.: Application of genetic programming to edge detector design. In: Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, IEEE (2006) 4683–4686
11. Harding, S., Banzhaf, W.: Genetic programming on gpus for image processing. Int. J. High Perform. Syst. Archit. **1**(4) 231–240
12. Vasicek, Z., Sekanina, L.: An evolvable hardware system in Xilinx Virtex II Pro FPGA. International Journal of Innovative Computing and Applications **1**(1) (2007) 63–73
13. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D.: Neural acceleration for general-purpose approximate programs. Commun. ACM **58**(1) 105–115
14. Yazdanbakhsh, A., Mahajan, D., Thwaites, B., Park, J., Nagendrakumar, A., Sethuraman, S., Ramkrishnan, K., Ravindran, N., Jariwala, R., Rahimi, A., Esmaeilzadeh, H., Bazargan, K.: Axilog: Language support for approximate hardware design. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2015, IEEE (2015) 812–817
15. Kulkarni, P., Gupta, P., Ercegovac, M.D.: Trading accuracy for power in a multiplier architecture. J. Low Power Electronics **7**(4) (2011) 490–501
16. Shi, K., Boland, D., Stott, E., Bayliss, S., Constantinides, G.: Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs. In: 51st ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE (2014) 1–6
17. Monajati, M., Fakhraie, S., Kabir, E.: Approximate arithmetic for low-power image median filtering. Circuits, Systems, and Signal Processing **34**(10) (2015) 3191–3219
18. Higuchi, T., Niwa, T., Tanaka, T., Iba, H., de Garis, H., Furuya, T.: 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 (1993) 417–424
19. Miller, J.F.: Cartesian Genetic Programming. Springer-Verlag (2011)
20. Torresen, J.: A scalable approach to evolvable hardware. Genetic Programming and Evolvable Machines **3**(3) (2002) 259–282
21. Vasicek, Z., Sekanina, L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genetic Programming and Evolvable Machines **12**(3) (2011) 305–327
22. Vasicek, Z., Sekanina, L.: Evolutionary approach to approximate digital circuits design. IEEE Transactions on Evolutionary Computation **19**(3) (2015) 432–444
23. Sekanina, L., Vasicek, Z.: Approximate circuits by means of evolvable hardware. In: 2013 IEEE International Conference on Evolvable Systems. Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE CIS (2013) 21–28

24. Miller, J.F., Smith, S.L.: Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evolutionary Computation **10**(2) (2006) 167–174
25. Kogge, P.M., Stone, H.S.: A parallel algorithm for the efficient solution of a general class of recurrence equations. IEEE Transactions on Computers (Aug 1973) 786–793
26. Nepal, K., Li, Y., Bahar, R.I., Reda, S.: Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In: Proceedings of the Conference on Design, Automation and Test in Europe. DATE '14, EDA Consortium (2014) 1–6
27. Martin, D., Fowlkes, C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: Proc. 8th Int'l Conf. Computer Vision. Volume 2. (July 2001) 416–423