Evaluation of a New Platform For Image Filter Evolution

Zdenek Vasicek
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
E-mail: vasicek@fit.vutbr.cz

Lukas Sekanina
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
E-mail: sekanina@fit.vutbr.cz

Abstract

This paper describes a new FPGA implementation of a system for evolutionary image filter design. Three parallel search algorithms are compared. An optimal mutation rate and the quality of three pseudo-random number generators are investigated. The efficiency of proposed system is demonstrated on the problem of removing the salt-and-pepper noise with intensity of 5%, 10% and 20% and designing an edge detector which works with input images corrupted by the salt-and-pepper noise.

1 Introduction

The image filter design problem is often approached by means of evolutionary design techniques. In addition to an optimization of filter coefficients (see, for example, [1]), evolutionary approaches are applied to find a complete structure of image filters. Sekanina evolved Gaussian noise filters using a variant of Cartesian Genetic Programming in which target filters were composed of simple digital components such as logic gates, adders and comparators [9]. Later, image filters for other types of noise and edge detectors were evolved using the same technique [10].

In order to speed up the evolutionary design process, an FPGA-based accelerator was proposed [6]. The accelerator uses the so-called *virtual reconfigurable circuit* to quickly evaluate candidate circuits. The accelerator implements a complete evolvable system in a single FPGA, i.e. the search engine, virtual reconfigurable circuit and fitness calculation unit are implemented as digital circuits using user logic available in the FPGA. This approach has been further developed by many authors [16, 4, 3].

For applications in the area of embedded systems, Xilinx has introduced PowerPC processors into the families Virtex 2, Virtex 4 and Virtex 5. As illustrated by Glette and Torresen for a two-bit multiplier design problem [2], the PowerPC processor can be used to implement a flexible search algorithm which, then, might be more sophisticated and efficient than a hardwired search algorithm. Recent paper [14] described a new FPGA implementation of a system for evolutionary image filter design in which genetic operations are carried out in the PowerPC processor. The main benefit of this architecture is that it allows the user to easily tune the search algorithm for a given problem while keeping the process of evolution on a single chip, i.e. very fast in comparison with a common PC.

This paper deals with an analysis of suitable parameters of various parallel search algorithms (random search, hill climbing and genetic algorithm), an optimal mutation rate and the quality of pseudo-random number generators for this new platform. The problem of interest is (1) removing the salt-and-pepper noise with intensity of 5%, 10% and 20% and (2) designing an edge detector which is able to deal with input images corrupted by the salt-and-pepper noise. Except the 5%-salt-and-pepper noise, other problems were not approached so far by means of evolutionary design techniques in literature.

2 Xilinx Virtex II Pro FPGA

We will use the Virtex II Pro FPGA which contains 23,616 slices, 49,788 flip flops, 852 IO blocks and 232 Block RAM modules [15]. Moreover, it contains the IBM PowerPC 405 core which is able to operate at 400 MHz. This core is equipped with a 5-stage pipeline, a virtual-memory-management unit, separate instruction-cache and data-cache units, 3 programmable timers, on-chip memory controller (OCM) and variety of interfaces, including processor local bus (PLB) interface, device control register (DCR) interface and JTAG port interface.

The chip can be configured either externally or internally, using the so-called Internal Configuration Access Port (ICAP). Although the port can operate at 66 MHz, it is not used for evolutionary filter design because of its difficult control and throughput insufficient for evolvable hardware [2, 6].

3 Evolvable systems in FPGAs

In order to produce an evolvable system, three main components have to be implemented: a genetic unit, an array of reconfigurable elements and a fitness calculation unit.

The FPGA-based implementations of evolvable hardware systems can be divided into two groups: (1) The FPGA serves for the fitness calculation only. The evolutionary algorithm (which is usually executed on a personal computer) sends configuration bitstreams representing candidate circuits to the FPGA in order to obtain their fitness values. (2) The entire evolvable system is implemented in an FPGA. Instead of using ICAP, virtual reconfigurable circuits (VRC) have been used for evolvable hardware in the recent years [3, 2, 6, 8]. The VRC is a second configurable layer developed on the top of an FPGA in order to obtain a fast reconfiguration scheme and application-specific programmable elements. While its implementation cost is relatively high, it directly enables to connect the chromosome of evolutionary algorithm (EA) with the configuration memory of reconfigurable array.

The problem domain determines the type and number of reconfigurable elements. In some cases the evolutionary design is performed directly with reconfigurable cells of an FPGA [11, 13]; in other cases a kind VRC is applied [3, 16, 2, 6, 8]. An evolutionary optimization of coefficients stored in registers represents the simplest example [12]. The EA and fitness calculation unit can be implemented either as an application specific circuit [12, 6, 8] or as a program. This program is running either in a personal computer [11] or in an embedded processor which is integrated into the FPGA. The embedded processor is typically available as a hard core (e.g., PowerPC in Virtex II Pro FPGA [2]) or as a soft core (e.g., the MicroBlaze core [13]).

4 Proposed architecture

4.1 Image filters in VRC

The proposed architecture was described in [14]. Every image operator is considered as a digital circuit of nine 8-bit inputs and a single 8-bit output, which processes gray-scaled (8-bits/pixel) images (see Fig. 1).

Fig. 2 shows a corresponding VRC which consists of 2-input Configurable Logic Blocks (CFBs), denoted as E_i, placed in a grid of 8 columns and 4 rows. Any input of each CFB may be connected either to a primary circuit input or to the output of a CFB, which is placed anywhere in the preceding column. Any CFB can be programmed to implement one of functions given in Table 1. All these functions operate with 8-bit operands and produce 8-bit results. These functions were recognized as useful for this task in [10]. The reconfiguration is performed column by column.

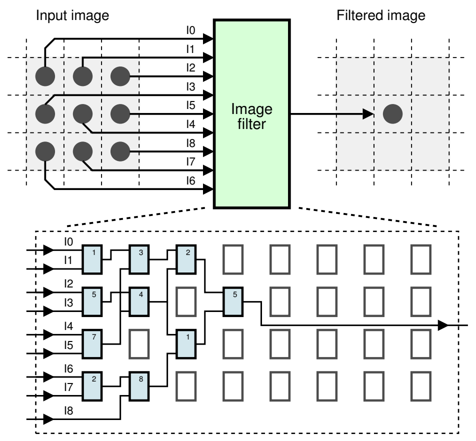


Figure 1. A candidate image filter

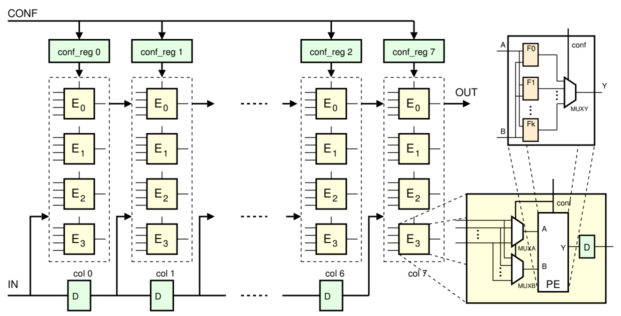


Figure 2. VRC for image filter evolution

The computation is pipelined; a column of CFBs represents a stage of the pipeline. Registers (denoted D) are inserted between the columns in order to synchronize the input pixels with CFB outputs. The configuration bitstream of VRC which is stored in a register array *conf_reg* consists of 384 bits. A single CFB is configured by 12 bits, 4 bits are used to select the connection of a single input, 4 bits are used to select one of the 16 functions. Evolutionary algorithm directly operates with configurations of the VRC; simply, a configuration is considered as a chromosome.

Table 1. Functions implemented in each CFB

code	function	description	code	function	description
0	255	constant	8	$x \gg 1$	right shift by 1
1	x	identity	9	$x \gg 2$	right shift by 2
2	$255 - x$	inversion	A	$swap(x, y)$	swap nibbles
3	$x \vee y$	bitwise OR	B	$x + y$	$+$ (addition)
4	$\bar{x} \vee y$	bitwise $\bar{x}$ OR y	C	$x +^S y$	$+$ with saturation
5	$x \wedge y$	bitwise AND	D	$(x + y) \gg 1$	average
6	$\overline{x \wedge y}$	bitwise NAND	E	$max(x, y)$	maximum
7	$x \oplus y$	bitwise XOR	F	$min(x, y)$	minimum

4.2 Search Algorithm

The proposed system allows the use of various parallel search algorithms. The algorithms, that we tested, will be described in Section 5. These algorithms utilize a population of candidate solutions and a single genetic operator — mutation, which inverts k bits of the chromosome (i.e. of the configuration). No crossover operator is employed because it is currently unknown how to design it to be more efficient than the mutation operator. The PowerPC processor implements the genetic operations.

4.3 Fitness Calculation

The fitness calculation is carried out by the Fitness Unit (FU). The pixels of corrupted image u are loaded from external SRAM1 memory and forwarded to inputs of VRC. Pixels of filtered image v are sent back to the Fitness Unit, where they are compared with the pixels of original image w which is stored in another external memory, SRAM2. Filtered image is simultaneously stored into the third external memory, SRAM3. The design objective is to minimize the difference between the filtered image and the original image, i.e. the fitness value is calculated for $M \times N$-pixel image (note that border pixels are ignored) as

$$fitness = \sum_{i=1}^{M-2} \sum_{j=1}^{N-2} |v(i,j) - w(i,j)|. \tag{1}$$

As the 3×3 pixels of the image window are not stored at neighboring addresses of SRAM, the hardware implementation of the fitness unit utilizes three first-in-first-out raw buffers, special addressing circuits and comparators to extract the filtering window from memory. The FU can be considered as an extension of the VRC pipeline. Hence, in each clock cycle, a temporary fitness value is updated by a new pixel difference.

4.4 Top level entity

As Fig. 3 shows, the proposed architecture (except the SRAM memories) is completely implemented in a single FPGA. All components (except the VRC) are connected to the LocalBus which is attached to the FPGA via a PCI bus. Now it remains to describe the Control Unit (CU), Processor and Memory Interface (PMI) and the PowerPC integration into the system.

In order to maximize the overall performance, the CU plays the role of master and controls the entire system. In particular, it starts/stops the evolution, determines the number of generations and other parameters of search algorithm and generates control signals for the remaining components.

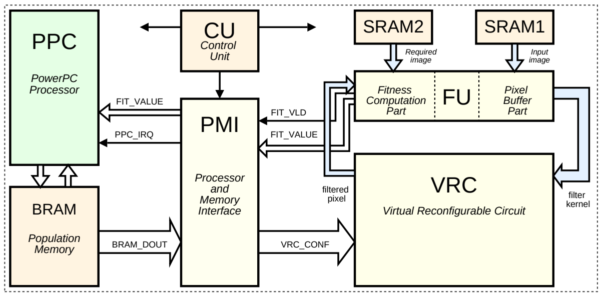


Figure 3. The image filter evolver in FPGA

Upon the request, the PowerPC generates a new candidate individual, i.e. it is idle in its main loop. The instruction memory of the PowerPC is implemented using on-chip Block RAM (BRAM) memories and connected to the LocalBus in order to send/read programs to/from an external PC. However, since our program is short, it can completely be stored in an instruction cache.

The population of candidate configurations is stored in on-chip BRAM memories. The population memory is divided into banks; each of them contains a single configuration bitstream of VRC. An additional bit (associated with every bank) determines data validity; only valid configurations can be evaluated. In order to overlap the evaluation of a candidate configuration with generating a new candidate configuration, at least two memory banks have to be utilized. While a circuit is evaluated, a new candidate configuration is generated. A new configuration is used immediately after completing the evaluation of the previous circuit. If b banks are utilized, the PowerPC processor has b-times more time to generate a new candidate circuit (i.e. EA can be more complicated). The proposed implementation utilizes eight banks.

The PMI component consists of two subcomponents working concurrently. The first subcomponent, controlled by the CU, reconfigures the VRC using configurations stored in the population memory. The second subcomponent is responsible for sending the fitness value to the PowerPC processor. This process is controlled by the FU. The PMI component also provides an interface to the population memory via LocalBus. The evaluation works as follows:

1. When a valid configuration is available, the CU initiates the reconfiguration of VRC. This process is controlled by PMI.

2. As soon as the first column of CFBs has been reconfigured, CU initiates the fitness calculation process performed by the FU.

3. When the last column of CFBs has been reconfigured, a corresponding memory bank is invalidated and the bank counter is incremented.

4. Three clock cycles before the end of evaluation the FU indicates the forthcoming end of evaluation.

5. The CU initiates a new configuration of VRC and repeats the sequence 1-4 again.

6. As soon as the fitness value is valid, it is sent (together with a corresponding bank number) to the PowerPC. An interrupt (IRQ) is generated to activate a service routine of the PowerPC. In this routine, a new candidate configuration is generated for the given bank. The PowerPC processor acknowledges the interrupt (IRQACK) and sets up the validity bit.

These steps are pipelined in such manner as there are no idle clock cycles. Therefore, time of evolution can be expressed as

$$t_{evol} = Q(M-2)(N-2)\frac{1}{f}$$

where Q is the number of evaluations, $N \times M$ is the number of pixels and f is the operation frequency.

4.5 Results of synthesis

In order to implement the proposed system, we used a COMBO6X card equipped with Virtex II Pro 2VP50ff1517 FPGA [5]. Results of synthesis are summarized in Table 2. While the PowerPC works at 300 MHz, the logic supporting the PowerPC works at 150 MHz. The remaining FPGA logic (including VRC and FU) works at 50 MHz. Experimental results show that approximately 3,000 candidate filters can be evaluated per second ($N = M = 128$).

Table 2. Results of synthesis

VRC	IO blocks	BRAM	Slices	DFF
Available	852	232	23 616	49 788
4×8 CFBs	602	12	4 591	3 638
used	70%	5%	20%	7%
No VRC	602	12	1 240	2 479
used	70%	5%	5%	5%

5 Description of search algorithms

Three parallel search algorithms are evaluated: a random search, a hill-climbing algorithm and a genetic algorithm. **Random search (RS):** This algorithm operates with p individuals that are generated randomly at the beginning of the evolution. Then an offspring is created using a bit-mutation operator from each parent and evaluated. If the offspring is equal or better than its parent then the offspring replaces the parent in the new population. In fact, p standard random search algorithms run in parallel. This algorithm was implemented in [6] as a special circuit. Fig. 4 shows concurrent operations of several processes running in hardware and the PowerPC processor (including the configuration of the VRC, evaluation of candidate filters and generation of candidate configurations). These processes are synchronized in such a way that no clock cycle is lost because of waiting on some resources. Note that only two banks are considered in this example.

Hill Climbing search (HC): This algorithm operates with p individuals that are generated randomly at the beginning of the evolution. After their evaluation, r offspring configurations are generated for each parent using a bit-mutation operator. The best offspring of the r offspring configurations replaces the corresponding parent; however, only in case that its fitness value is equal or better than the parent's fitness value. Again, in fact, p standard hill climbing algorithms run in parallel.

Genetic algorithm (GA): The initial population of p individuals is generated randomly. Then, r offspring are generated from each parent using a bit-mutation operator. A new population consisting of p individuals is formed from p parents and their $p.r$ offspring. We used a deterministic selection in which p-best scored individuals are selected as new parents.

6 Experimental results

Experiments were arranged to find a suitable mutation rate and an efficient pseudo-random number generator. We also compared the three search algorithms. The objective was to (1) remove the salt-and-pepper noise with intensity of 5%, 10% and 20% from real-world images and (2) design an edge detector which is able to deal with input images corrupted by the salt-and-pepper noise. A visual quality of filtered images is expressed in *mdpp* which stands for the mean difference per pixel between the filtered image and original image.

6.1 The mutation rate

Our strategy is to estimate the suitable mutation rate using not so many evaluations (less than 100,000 evaluations allowed) and then to utilize the discovered mutation rate in long-time experiments. Figure 5 shows average *mdpp* calculated from the best *mdpp* values at the end of 32 independent runs of the RS algorithm ($p = 8$) for each of $k = 1 - 127$ inverted bits in the chromosome. Two methods are used: exactly k bits are always inverted (denoted as "fix" in Fig. 5) and a randomly chosen number of bits is inverted; however, limited by k (denoted as "rnd" in Fig. 5).

We can observe in Figure 5 that the mutation rate which allows minimizing the *mdpp* is usually 20 bits per chromo-

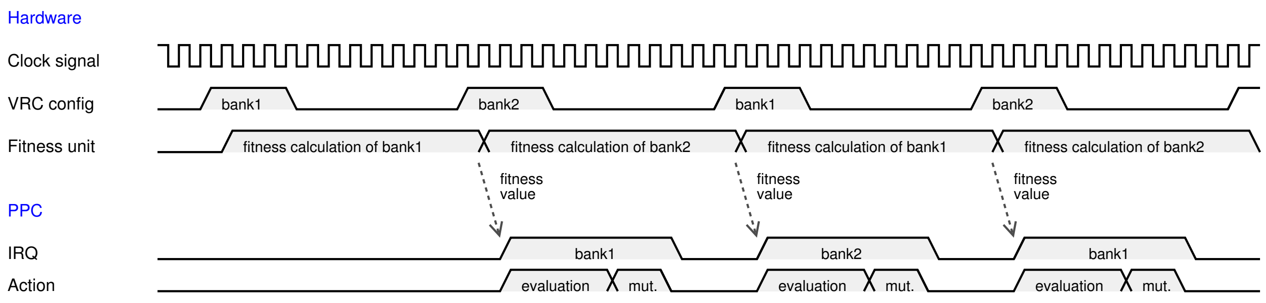


Figure 4. Example of timing for 2 banks: the reconfiguration of VRC costs 4 clock cycles, the evaluation costs 12 clock cycles and the interrupt routine requires 8 clock cycles.

some, i.e. 5.2%. It is also more efficient to invert exactly 20 bits than to randomly generate a number from interval $1 - 20$.

6.2 Pseudorandom number generators

As the outputs of pseudorandom number generators (PRNG) only approximate some of the properties of random numbers, we have to determine a suitable one for the proposed architecture. The following three PRNGs were evaluated:

Linear congruential generators represent the oldest and best-known pseudorandom number generator algorithms. It is, however, well known that the properties of this class of generators are far from ideal. The applied linear congruential generator operates according to formula

$$V_{j+1} = (1103515245 \times V_j + 12345) \bmod 2^{32}.$$

Linear Feedback Shift Register (LFSR) is a shift register whose input bit is driven by the exclusive-or (xor) of some bits of the overall shift register value. As for this PRNG is also known that output bits do not pose a good distribution we used a parallel LFSR consisting of 32 independent and different LFSRs seeded identically.

Mersenne Twister algorithm is a twisted generalized feedback shift register that avoids many of the problems with earlier generators. It has the colossal period of $2^{19937} - 1$ iterations, is proven to be equidistributed in (up to) 623 dimensions (for 32-bit values). A standard implementation of Mersenne Twister was utilized [7].

Figure 6 shows average *mdpp* and corresponding standard deviations obtained from 40 independent runs (after 12,288 evaluations in each run) using the RS algorithm ($p = 8$, "fix" mutation applied on 20 bits). The three generators are compared on two problems: removing 10% salt-and-pepper noise from Lena image (the first image shows *mdpp*, the second image shows a standard deviation) and edge detector design (the third image shows *mdpp*, the fourth image shows a standard deviation). Surprisingly, there are not any significant differences in the quality of obtained results.

6.3 Comparison of search algorithms

Table 3 provides parameters of experiments arranged in order to compare the three algorithms — RS ($p = 8$), HC ($p = 8, r = 2$) and GA ($p = 8, r = 2$). As a training image we used a 128×128-pixel version of Lena image (XLena) which contains a given type of noise in some regions. Table 4 summarizes obtained results. We can observe that while the best **average** *mdpp* is always obtained by means of the RS algorithm, the GA always produces a filter with the smallest *mdpp* at all. Recall that the number of evaluations is **identical**; however, RS always produces **more** generations than the GA. Figures 7 and 8 give examples of images filtered using the best-evolved filters. Table 5 compares *mdpp* of the best-evolved filters and conventional filters (median and Sobel operator) on a set of 256×256-pixel test images.

Table 3. Parameters of experiments

corrupted image	target image	bits mutated	runs	evaluations per run
5%-noise XLena	XLena	18	64	160,000
10%-noise XLena	XLena	24	349	320,000
20%-noise XLena	XLena	20	139	320,000
5%-noise XLena	Edges in XLena	18	389	160,000

7 Discussion

The proposed FPGA implementation of image filter evolution can generate a solution approx. 22 times faster than a PC with Celeron 2.4GHz (i.e. 3000 evaluations/s). This allows us to perform a detailed evaluation of various aspects of used algorithms in a reasonable time. Experimental results show that the use of a parallel RS algorithm is a good

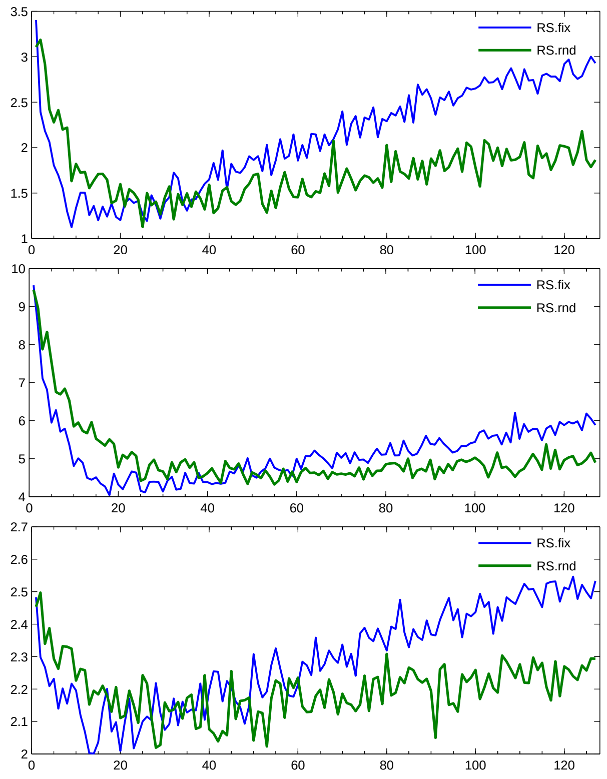


Figure 5. The average *mdpp* (y-axis) calculated from 32 independent runs for a given number of mutated bits (x-axis). Top: for Lena with 5%-salt-and-pepper noise. Middle: for Lena with 10%-salt-and-pepper noise. Down: for Lena with 5%-salt-and-pepper noise with the goal to obtain an edge detector

choice in this case; the RS produces the best results in average. However, we compared the number of evaluations. If the number of generations were compared, the GA is able to find a suitable solution much faster than the RS. Therefore, there is a tradeoff between the number of generations and the number of evaluations. If an average solution is required, it is better to run the RS which provides an average filter quickly. However, if a perfect filter is a must and more generations can be produced, the GA should be utilized.

In comparison with [6] the proposed implementation requires almost identical amount of logic on the chip. In addition, the PowerPC processor is employed. However, the proposed solution offers the possibility to easily change the search algorithm which is impossible in the former one.

The images filtered by evolved filters are not as smudged as the images filtered by median filters. Moreover, evolved filters occupy only approx. 70% of the area needed to implement the median filter on the same FPGA.

Table 4. Comparison on four test problems

noise type	**search algorithm**	**mean difference per pixel**			
		min	**max**	**mean**	**std.dev.**
5% salt-and-pepper noise	RSplf	0.410	3.190	0.967	0.581
	HC2plf	0.432	3.320	1.060	0.615
	GA2plf	0.333	3.450	2.010	1.240
10% salt-and-pepper noise	RSplf	0.982	3.280	1.720	0.337
	HC2plf	0.913	48.01	4.370	3.730
	GA2plf	0.828	7.390	2.650	2.190
20% salt-and-pepper noise	RSplf	1.870	4.350	2.850	0.510
	HC2plf	1.650	4.190	2.880	0.587
	GA2plf	0.870	12.10	2.680	1.330
5% noise, edge detection	RSplf	1.100	2.660	1.910	0.419
	HC2plf	1.380	2.960	2.310	0.421
	GA2plf	1.070	2.660	2.400	0.453

Table 5. Results for tests images

test	**5% noise**		**10% noise**		**edge detection**	
image	**evolved**	**median**	**evolved**	**median**	**evolved**	**Sobel**
Airplane	0.338	3.536	0.874	3.843	0.988	2.902
Bird	0.147	1.514	0.389	1.648	0.467	2.827
Bridge	0.657	7.830	1.386	8.165	1.688	2.856
Camera	0.627	4.413	0.850	4.746	1.108	2.786
Goldhill	0.451	5.870	0.962	6.134	1.161	2.812
Lena	0.367	3.577	0.863	3.893	1.022	2.832

The use of various pseudo-random generators has no effect on the quality of evolved filters and speed of evolution. This result is also surprising. Is there any relation to the fact that the parallel random search exhibits the best performance? This is an open question for future research.

8 Conclusions

We evolved image filters for three types of noise which were not approached by means of evolutionary design so far: the salt-and-pepper noise with intensity of 10% and 20% and 5%-salt-and-pepper noise existing in the image in which edges should be detected. Evolved filters are at least comparable with a conventional solution which is based on the median filter. As Fig. 8bf shows, in contrast to evolved filters, images filtered by the median filter are smudged. The proposed platform can be considered as an efficient “designer” of image filters which can be utilized in sophisticated filtering schemes for real-world applications.

Acknowledgements

This research was partially supported by the Grant Agency of the Czech Republic under No. 102/07/0850 *Design and hardware implementation of a patent-invention machine* and the Research Plan No. MSM 0021630528 – *Security-Oriented Research in Information Technology*.

References

[1] J. Dumoulin, J. Foster, J. Frenzel, and S. McGrew. Special Purpose Image Convolution with Evolvable Hardware. In *Real-World Applications of Evolutionary Computing*, volume 1803 of *LNCS*, pages 1–11. Springer Verlag, 2000.

[2] K. Glette and J. Torresen. A flexible on-chip evolution system implemented on a xilinx virtex-ii pro device. In *Evolvable Systems: From Biology to Hardware*, volume 3637 of *LNCS*, pages 66–75. Springer, 2005.

[3] D. Gwaltney and K. Dutton. A VHDL Core for Intrinsic Evolution of Discrete Time Filters with Signal Feedback. In *Proc. of the 2005 NASA/DoD Conf. on Evolvable Hardware*, pages 43–50, Washington D.C., USA, 2005. IEEE CS.

[4] P. N. Kumar, S. Suresh, and J. R. P. Perinbam. Digital image filter design using evolvable hardware. In *Proc. of the Fourth Annual ACIS Int. Conf. on Computer and Information Science ICIS05*, pages 483–488. IEEE CS, 2005.

[5] Liberouter project, 2005. URL: http://www.liberouter.org.

[6] T. Martinek and L. Sekanina. An evolvable image filter: Experimental evaluation of a complete hardware implementation in fpga. In *Evolvable Systems: From Biology to Hardware*, volume 3637 of *LNCS*, pages 76–85. Springer, 2005.

[7] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[8] R. Salomon, H. Widiger, and A. Tockhorn. Rapid Evolution of Time-Efficient Packet Classifiers. In *IEEE Congress on Evolutionary Computation*, pages 2793–2799, Vancouver, Canada, 2006. IEEE CIS.

[9] L. Sekanina. Image Filter Design with Evolvable Hardware. In *Applications of Evolutionary Computing – Proc. of the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'02*, volume 2279 of *LNCS*, pages 255–266. Springer, 2002.

[10] L. Sekanina. *Evolvable components: From Theory to Hardware Implementations*. Springer Verlag, 2004.

[11] A. Thompson, P. Layzell, and S. Zebulum. Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution. *IEEE Transactions on Evolutionary Computation*, 3(3):167–196, 1999.

[12] G. Tufte and P. Haddow. Evolving an adaptive digital filter. In *The Second NASA/DoD workshop on Evolvable Hardware*, pages 143–150. IEEE Computer Society, 2000.

[13] A. Upegui and E. Sanchez. Evolving hardware with self-reconfigurable connectivity in Xilinx FPGAs. In *The 1st NASA/ESA Conference on Adaptive Hardware and Systems(AHS-2006)*, pages 153–160, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[14] Z. Vasicek and L. Sekanina. An evolvable hardware system in Xilinx Virtex II Pro FPGA. *Int. J. Innovative Computing and Applications*, 1(1):63–73, 2007.

[15] Xilinx Virtex-II Pro Platform FPGAs, 2005.

[16] Y. Zhang, S. Smith, and A. Tyrrell. Digital Circuit Design using Intrinsic Evolvable Hardware. In *Proc. of the 2004 NASA/DoD Conf. on Evolvable Hardware*, pages 55–62. IEEE Computer Society, 2004.

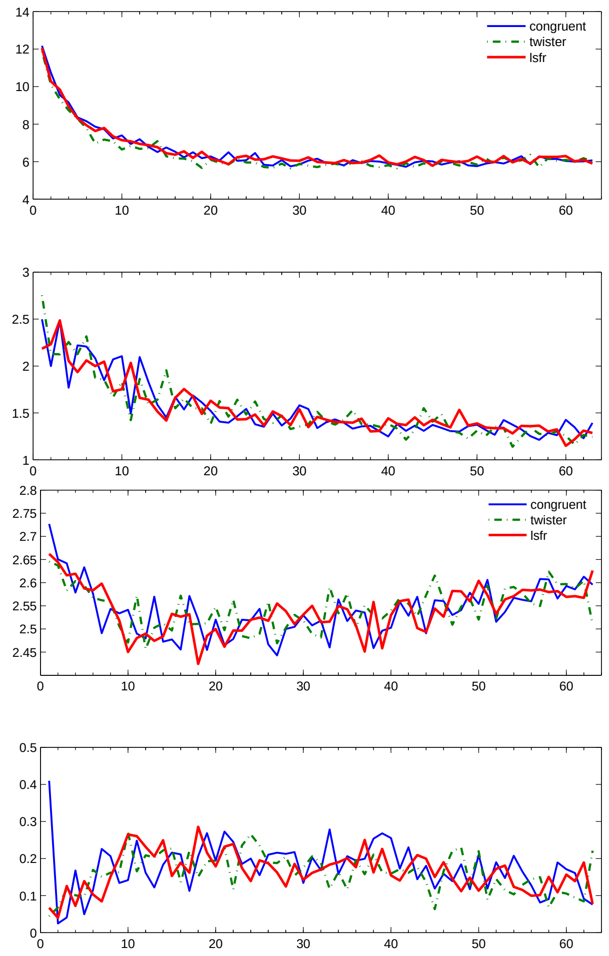
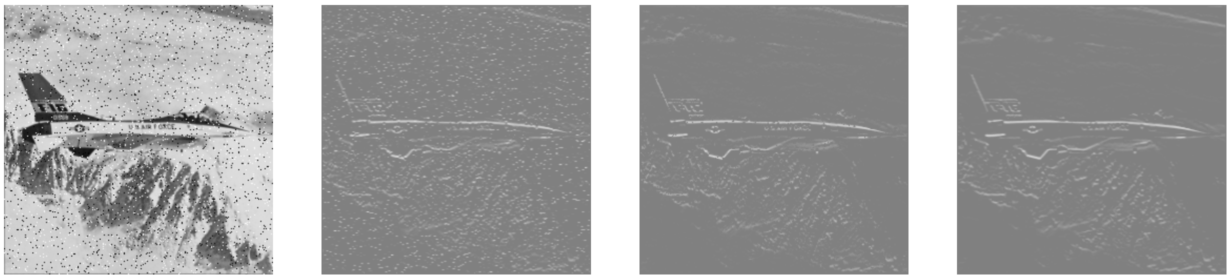
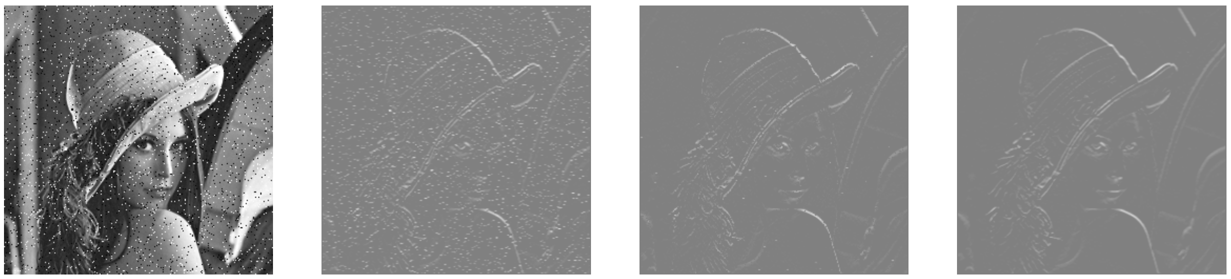


Figure 6. Comparison of linear congruential generator (congruent), LFSR and Mersenne Twister on two problems. In the first and third plot, the *mdpp* values (y-axis) are plotted against the number of mutated bits (x-axis). In the second and fourth plot, the corresponding standard deviations (y-axis) are plotted against the number of mutated bits (x-axis).

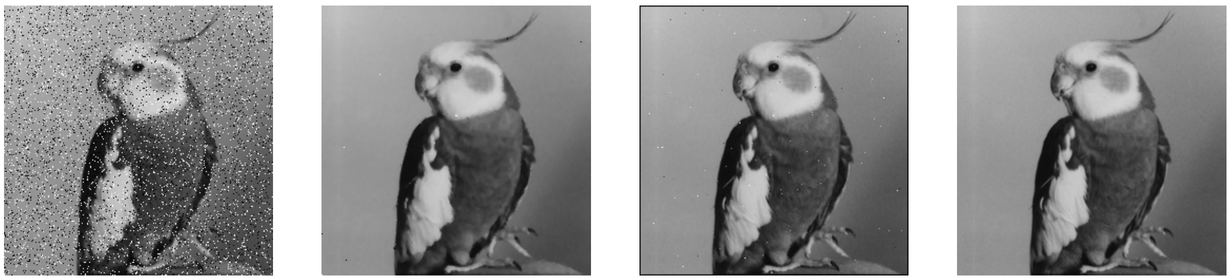


(a) Corrupted image (b) Conventional filtering (c) Evolved filtering (d) Desired output

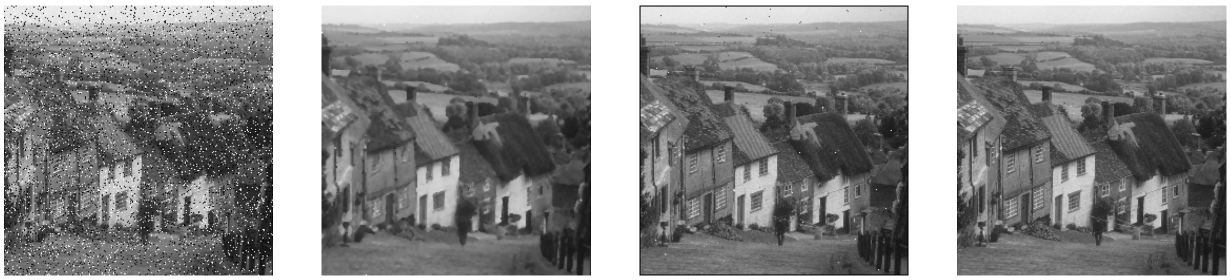


(e) Corrupted image (f) Conventional filtering (g) Evolved filtering (h) Desired output

Figure 7. The Airplane and Lena images from test set. Edge detection in images corrupted by the 5% salt-and-pepper noise.



(a) Corrupted image (b) Conventional filtering (c) Evolved filtering (d) Desired output



(e) Corrupted image (f) Conventional filtering (g) Evolved filtering (h) Desired output

Figure 8. The Bird and Goldhill images from test set. The 10% salt-and-pepper noise removal task