Mixed Constrained Image Filter Design Using Particle Swarm Optimization

Zhiguo Bao and Takahiro Watanabe

Graduate School of Information Production and Systems, Waseda University
Kitakyushu-shi, Japan
(Email: baozhiguo@moegi.waseda.jp)

Abstract: This paper describes evolutionary image filter design for noise reduction using particle swarm optimization (PSO), where mixed constraints on the circuit complexity, power and signal delay are optimized. First, the evaluating values about correctness, complexity, power and signal delay are introduced to the fitness function. Then PSO autonomously synthesizes a filter. To verify the validity of our method, an image filter for noise reduction is synthesized. The performance of resultant filter by PSO is similar to that of Genetic Algorithm (GA), but the running time of PSO is 10% shorter than that of GA.

Keywords: PSO, Evolutionary Design, Image Filter.

I. INTRODUCTION

The idea of evolutionary hardware design was introduced at the beginning of 1990s in papers [1, 2], and it is usually defined as an approach in which the Genetic Algorithm (GA) is utilized to search for a suitable configuration of a reconfigurable device in order to meet a given specification.

The image filter design problem is often approached by means of evolutionary design techniques. In addition to an optimization of filter coefficients (for example, [3]), evolutionary approaches are applied to find a complete structure of image filters. In [4], Gaussian noise filters were evolved using a variant of Cartesian Genetic Programming in which target filters were composed of simple digital components, such as logic gates, adders and comparators. Later, image filters for other types of noise and edge detectors were evolved using the same technique [5]. But they did not discuss any circuit constraints such as complexity, power consumption and signal delay, while we proposed mixed constrained design optimization using GA for some combinational circuits [6, 7]. The proposed method could synthesize good circuits about complexity, power and signal delay, but it took the large running time for GA process. As another optimization method, particle swarm optimization (PSO) [8, 9, 10] was proposed and evaluated, and it is promising to find a good solution in shorter time, compared to GA.

This paper applies PSO to mixed constrained image filter design for noise reduction, shown in Fig. 1. The circuit complexity, power and signal delay which are caused by both logic gates and wires, are optimized. In this design, first, the evaluating value about correctness, complexity, power and signal delay are introduced to the fitness function. Then PSO autonomously synthesizes an image filter which is simpler and has better performance than the conventional design. To verify the validity of our method, an image filter for reducing noise is experimentally synthesized.

The organization of this paper is as follows: a brief overview of PSO is described in the next section. Section III describes design optimization for an image filter using PSO. Section IV shows the experimental results. Finally, Sect. V concludes this paper.

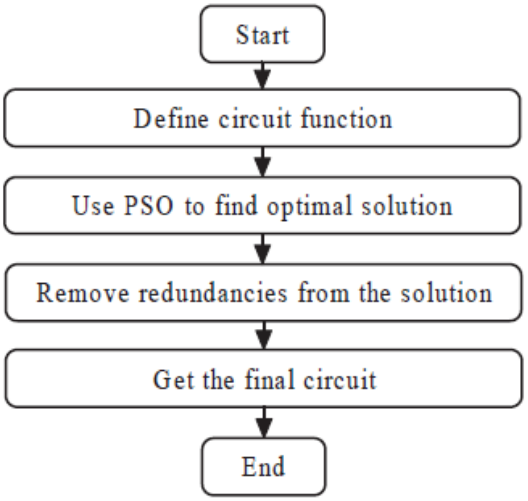

Fig. 1. The overview of our method

II. Particle swarm optimization

Particle swarm optimization (PSO) is an algorithm model on swarm intelligence that finds a solution to an optimization problem in a search space, shown in Fig. 2.

In PSO, a particle represents a candidate solution to the problem. Each particle is treated as a point in the *D*-dimensional problem space. The *i*-th particle is represented as $X_i = (x_{i1}, x_{i2}, \dots, x_{iD})$. The best previous position (the position giving the best fitness value) of the *i*-th particle is recorded and represented as $P_i = (p_{i1}, p_{i2}, \dots, p_{iD})$. The index of the best particle among all the particles in the population is represented by the symbol *g*. The rate of the position change (velocity) for particle *i* is represented as $V_i = (v_{i1}, v_{i2}, \dots, v_{iD})$. The particle is updated according to the following equations:

$$v_{id}^{(t+1)} = w \times v_{id} + c_1 \times rand() \times (p_{id} - x_{id}^{(t)}) + c_2 \times Rand() \times (p_{gd} - x_{id}^{(t)}). \quad (1)$$

$$x_{id}^{(t+1)} = x_{id}^{(t)} + v_{id}^{(t+1)}. \quad (2)$$

where,

$0 <= i <= (n-1)$, $1 <= d <= D$.

n : number of particles in a group.

D : number of members in a particle.

t: pointer of iterations (generations).

w : inertia weight factor.

c1, *c2* : acceleration constant.

rand(), *Rand()*: uniform random value in the range [0,1].

$v_{id}^{(t)}$: velocity of particle *i* at iteration *t*, $V_d^{\min} \le v_{id}^{(t)} \le V_d^{\max}$.

$x_i^{(t)}$: current position of particle *i* at iteration *t*.

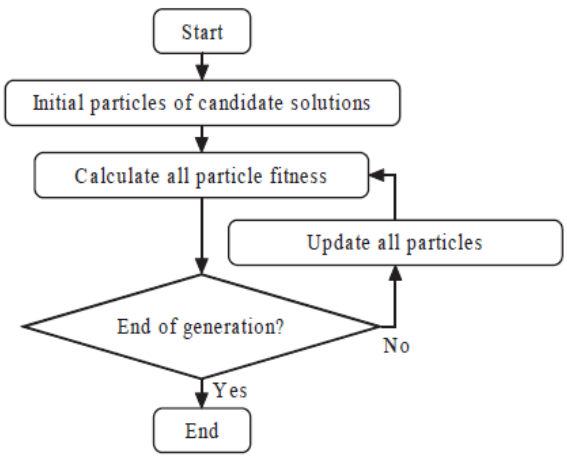

Fig. 2. The evolutionary process of PSO

The inertia weight factor *w* is employed to control the impact of the previous history of velocities on the current velocity, thereby influencing the trade-off between global (wide-ranging) and local (fine-grained) exploration abilities of the "flying points". A larger inertia weight facilitates global exploration (searching new areas), while a smaller inertia weight tends to facilitate local exploration to free tune the current search area. Suitable selection of the inertia weight provides a balance between global and local exploration abilities and thus requires lesser iterations on average to find the optimum. Good values of *w* are usually slightly less than 1 [10]. It could be randomly initialized for each particle. Or a high value of *w* at the beginning of the run facilitates global search, while a small *w* tends to localize the search.

c_1 and c_2 are constants that say how much the particle is directed towards good positions. They represent a "cognitive" and a "social" component, respectively, in that they affect how much the particle's personal best and the global best (respectively) influence its movement. Usually we take $c_1 = c_2 = 2$ [10].

III. Image filter design using PSO

PSO is applied to search good solutions to optimize the image filter design.

The target image filter is to provide identical functional behavior with less complexity, less power and less signal delay.

1. Image filter

Every image operator is considered as a digital circuit with nine 8-bit inputs and a single 8-bit output, which processes gray-scaled (8-bit/pixel) images.

As shown in Fig. 3, every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbors in the processed image [5].

2. Reconfigurable processing array for image filter

Similarly to [11], the reconfigurable image filter is implemented as a Virtual Reconfigurable Circuits (VRC) (Fig. 4). As a new pixel value is calculated using nine pixels, the VRC has got nine 8-bit inputs and a single 8-bit output. The VRC consists of two-input CLBs (Configurable Logic Blocks in FPGA) placed in a 4*4 array. Any input of each CLB may be connected to either a primary circuit input or the output of a CLB in the preceding column. Any CLB can be programmed to

implement one of the functions given in Table 1 [5]. All these functions operate with 8-bit operands and produce 8-bit results. Table 1 also gives value of complexity (FC), power (FP) and signal delay (SD) for each function in a CLB.

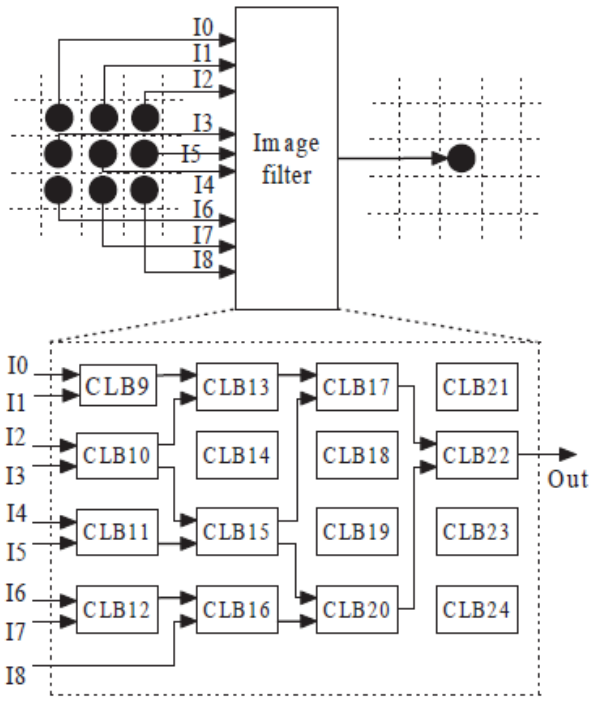

Fig. 3. A candidate image filter

In Fig. 4, there are position of inputs and output of each logic block. Therefore, we add values of wire about complexity, power and signal delay in Table 1.

3. Genetic encoding

The chromosome (particle) is a string of integers where each three continuous integers constitute a logic block. Each triplet in the chromosome encodes the two inputs and the function type of a logic block, respectively, such as:

(*Input_1*, *Input_2*, *Function type*).

A typical chromosome then can be a sequence of triplets [6, 7], such as:

$(IN_1^1, IN_2^1, F_{type}^1)$ … $(IN_1^i, IN_2^i, F_{type}^i)$ …

Here, IN_1^i and IN_2^i mean positions of the corresponding signal. F_{type}^i means *Function_type*. For primary input from I_0 to I_8, the range of IN^i is ($0 <= IN^i <= 8$). For input from output of a logic block CLB_m, that is, CLB_9 to CLB_{24} in Fig. 4, $IN^i = m$. Function in a CLB is defined as shown in Table 1.

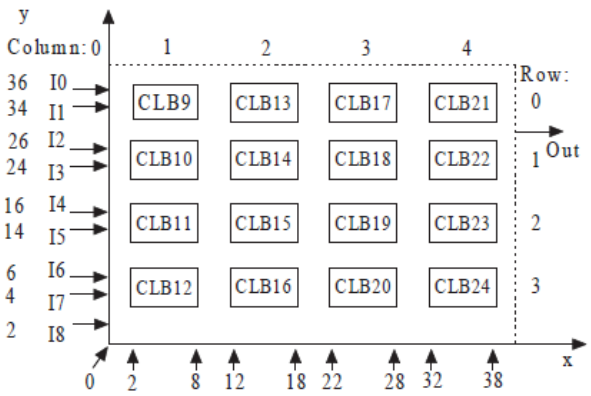

Fig. 4. A reconfigurable processing array

Table 1. Functions implements in a CLB

ID	Function	Description	*FC*	*FP*	*SD*
0	255	Constant	8	8	1
1	x	Identity	16	16	2
2	$255 - x$	Inversion	24	24	3
3	$x \vee y$	Bitwise OR	32	32	3
4	$\overline{x} \vee y$	Bitwise $\overline{x}$ OR y	40	40	4
5	$x \wedge y$	Bitwise AND	32	32	3
6	$not(x \wedge y)$	Bitwise NAND	40	40	4
7	$x \oplus y$	Bitwise XOR	64	64	4
8	$x \gg 1$	Right shift by 1	15	15	2
9	$x \gg 2$	Right shift by 2	14	14	2
10	$(x \ll 4) \vee (y \gg 4)$	Swap	16	16	2
11	$x + y$	+ (addition)	358	358	18
12	$x +^s y$	+ with saturation	367	367	19
13	$(x + y) \gg 1$	Average	350	350	18
14	$max(x, y)$	Maximum	240	240	16
15	$min(x, y)$	Minimum	240	240	16
-	(wire)	(wire)	16	16	2

FC: function complexity.
FP: function power.
SD: signal delay.

4. Fitness function

The pixels of corrupted image c_i are used as inputs of VRC. Pixels of filtered image f_i are generated, which are compared to the pixels of original image o_i.

The design objective is to minimize the difference between the filtered image and the original image. The image size is *nc*nr* pixels, but only the area of *(nc-2)*(nr-2)* pixels is considered, because the pixel values at the borders are ignored, and thus remain unfiltered. The fitness value of a candidate filter is obtained as follows:

(1) the VRC is configured using a candidate chromosome

(2) the created circuit is used to produce pixel values in the image f_i and

(3) the fitness value is calculated as

$$Fitness = (-1) \times (F_1 \times \beta + F_2). \quad (3)$$

Where, F_1 and F_2 are defined as follows and β is the weight on F_1.

$$F_1 = \sum_{i=1}^{nc-2} \sum_{j=1}^{nr-2} (|fi(i, j) - oi(i, j)|). \quad (4)$$

Where,

nc: the number of columns of the pixels in the image.

nr: the number of rows of the pixels in the image.

$f_i(i; j)$: the pixel (i, j) in filtered image f_i, the value range is [0,255].

$o_i(i ; j)$: the pixel (i, j) in original image o_i, the value range is [0,255].

$$F_2 = SD \times \alpha_{sd} + Pg \times \alpha_{pg} + Cg \times \alpha_{cg} + Pw \times \alpha_{pw} + Cw \times \alpha_{cw}. \quad (5)$$

Where,

SD: signal delay of a circuit individual, determined by a critical path.

α_{sd}: the weight on signal delay in F_2.

Pg: power of logic blocks in a circuit, calculated by summation of all logic block's power.

α_{pg}: the weight on power of logic blocks in F_2.

Cg: complexity of logic blocks in a circuit, calculated by summation of all logic block's complexity.

α_{cg}: the weight on complexity of logic blocks in F_2.

Pw: power of all wires in a circuit, calculated by summation of all wire's power.

α_{pw}: the weight on power of wires in F_2.

Cw: complexity of wires in a circuit, calculated by summation of all wire's complexity.

α_{cw}: the weight on complexity of wires in F_2.

The priority of evaluating values in Eq. (3) is: $F_1 > F_2$. In this experiment, β is set to $0.1*10^9$. The priority of evaluating values in Eq. (5) is: $SD > Pg > Cg > Pw > Cw$. In this experiment, α_{sd} is set to $0.1*10^6$, α_{pg} is set to $1*10^3$, α_{cg} is set to $0.1*10^3$, α_{pw} is set to 10, and α_{cw} is set to 1. All α's and β are empirically assigned in our experiment.

IV. Experimental results

Table 2 shows the parameters of the evolution of PSO used in this experiment. Some preliminary experiments were performed in advance to decide parameters suitable for our experiment.

The proposed method was implemented in Eclipse SDK 3.1.1 with jre 1.6.0; and tested on a PC with Inter(R) Core(TM) 2 CPU at 2.67 GHz and 2.0 GB RAM.

Table 2. Conditions for evolution

Number of Generation : 100.
Population Size : 600.
Inertia weight factor w : [0.4, 1.0]. Limit of change in velocity of each member in an individual: $V_d^{max} = 0.5 * P_d^{max}, V_d^{min} = -0.5 * P_d^{max}$. Acceleration constant : c1 = 2, c2 = 2.

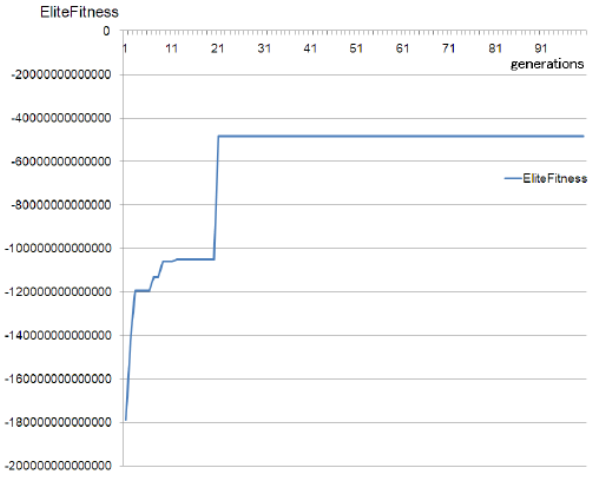

Fig. 5. Elite fitness of PSO (Y-axis) vs. the number of generations (X-axis)

The image filter is evolved for a 512*512 Lena image corrupted by 5% salt-and-pepper noise, shown in Fig. 7.

Fig. 5 shows the elite fitness of PSO with w = 0:9 vs. the number of generations during the image filter evolution. The elite fitness is increasing during evaluation time.

Table 3 shows the results of PSO with different w. For each PSO, we use results over 10 independent trials. "max" means the best elite fitness value from 10 trials; "average" means the average fitness value of 10 individuals; "time" means the average running time (minutes) of one trial. The larger the fitness is, the better the image filter is. The less the ratio is, the better the image filter is.

From the results, we can see that PSO (0.9) produces better solutions than others, from the point of the best elite fitness.

An example of chromosome by PSO (0.9) with fitness -48550809287267 is as follows:

(0, 0, 0)(1, 7, 15)(3, 5, 14)(8, 5, 15)(0, 0, 0)
(0, 0, 0)(0, 0, 0)(11, 12, 14)(0, 0, 0)(0, 0, 0)(0, 0, 0)
(4, 16, 15)(10, 20, 14)(0, 0, 0)(0, 0, 0)(0, 0, 0)

Table 3: Fitness values of PSO with different w, and GA.

Item	Max		Average		Running time	
	Fitness	Ratio	Fitness	Ratio	Time(m)	Ratio
$PSO(0.4)$	-80005207479684	1.65	-99745634920892	1.11	42.80	0.99
$PSO(0.5)$	-80535402393364	1.66	-94873624466988	1.06	43.60	1.01
$PSO(0.6)$	-79181905512328	1.63	-97375325953839	1.09	43.20	0.99
$PSO(0.7)$	-56076103257304	1.15	-94746895485425	1.06	42.60	0.98
$PSO(0.8)$	-57981907477527	1.19	-91487054845692	1.02	41.70	0.96
$PSO(0.9)$	-48550809287267	1.00	-89652125806386	1.00	43.00	1.00
$PSO(1.0)$	-85478004803230	1.76	-102629344653216	1.14	40.00	0.92
$PSO(w1)$	-82558108959223	1.70	-93726626111394	1.05	42.80	0.99
$PSO(w2)$	-63176503079148	1.30	-91741585167119	1.02	42.00	0.97
GA	-47595302921595	0.98	-64678393342856	0.72	49.70	1.10

$PSO(w)$: PSO with $w = 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$.
$PSO(w1)$: PSO with $w = 1.0 - 0.6 * generation/generation\_size$.
$PSO(w2)$: PSO with $w = 0.4 + 0.6 * random()$.
GA: Number of Generation, Population Size is same to that of PSO; Elite Size = 10, Crossover Probability (Pc) = 0.3, Mutation Probability (Pm) = 0.1, (ref. [7]).

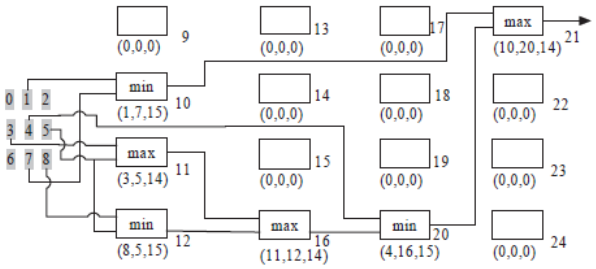

Fig. 6. The optimized image filter by PSO (0.9)

The graphical representation of this chromosome is shown in Fig. 6.

As the image is relatively large, we can assume that the evolved filter is general. The filter is able to remove the same type of noise also from other images. The image filter was evolved using Lena image and tested on other images.

Fig. 7, 8 and 9 show the input images with 5% salt-and-pepper noise, the Mean difference per pixel (MDPP) value of these images are 6.42, 6.29 and 6.35, respectively. Fig. 10, 11 and 12 show the output images by the image filter in Fig. 6, the MDPP value of these images are 1.87, 2.37 and 2.51, respectively. Obviously, this image filter could reduce noise for all cases.

V. Conclusions

This paper proposed the use of PSO (Particle Swarm Optimization) for mixed constrained image filter design for noise reduction. The complexity, power and signal delay both of the CLBs (Configurable Logic Blocks) and wires are considered. An image filter for removing noise is experimentally synthesized using PSO, to verify the validity of our method. By evolution, quality of the optimized image filter by PSO (0.9) is almost same as that of GA, but the running time of PSO is 10% shorter than that of GA.

Acknowledgements

This research was supported by the "Ambient SoC Global COE Program of Waseda University" of the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan and partially by the Intellectual Cluster project of MEXT, Japan and the Core Research for Evolutional Science and Technology (CREST) of Japan Science and Technology Agency (JST).

REFERENCES

[1] T. Higuchi, T. Niwa, T. Tanaka, et al. (1993), "Evolving hardware with genetic learning: a first step towards building a Darwin machine," Proc. the Second International Conference on Simulated Adaptive Behaviour, MIT Press, pp. 417-424.
[2] H. de Garis (1993), "Evolvable hardware: Genetic programming of a Darwin machine," Proc. International Conference on Artificial Neural Networks and Genetic Algorithms, Innsbruck, Austria, Springer.

Figure 7: The input image (Lena) with noise, (MDPP: 6.42).

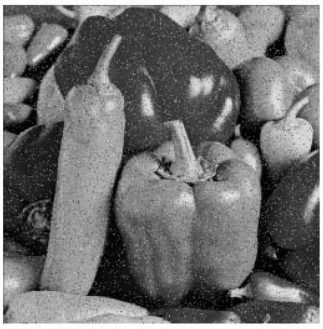

Figure 8: The input image (Peppers) with noise, (MDPP: 6.29).

Figure 9: The input image (Goldhill) with noise, (MDPP: 6.35).

Figure 10: The output image (Lena), (MDPP: 1.87).

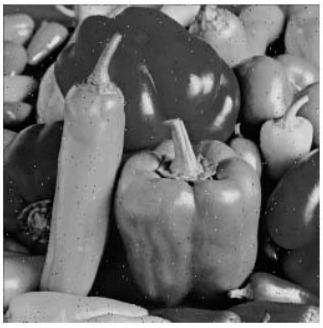

Figure 11: The output image (Peppers), (MDPP: 2.37).

Figure 12: The output image (Goldhill), (MDPP: 2.51).

[3] J. Dumoulin, J. Foster, J. Frenzel, et al. (2000), "Special Purpose Image Convolution with Evolvable Hardware," Real-World Applications of Evolutionary Computing, vol. 1803 of LNCS, Springer, 2000, pp. 111-125.
[4] L. Sekanina (2002), "Image Filter Design with Evolvable Hardware," Applications of Evolutionary Computing, (the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing, EvoIASP 2002), vol. 2279 of LNCS, Springer, 2002, pp. 255-266.
[5] Z. Vasicek and L. Sekanina (2007), "Evaluation of a New Platform for Image Filter Evolution," Proc. the Second NASA/ESA Conference on Adaptive Hardware and Systems, 2007 (AHS 2007), Aug. 2007, pp. 577-586.
[6] Z. Bao and T. Watanabe (2009), "A Novel Genetic Algorithm with Cell Crossover for Circuit Design Optimization," Proc. IEEE International Symposium on Circuits and Systems 2009 (ISCAS 2009), Taipei, Taiwan, May 2009, pp. 2982-2985.
[7] Z. Bao and T. Watanabe (2009), "Evolutionary Design for Image Filter using Genetic Algorithm," Proc. IEEE TENCON 2009, Singapore, Nov. 2009, (accepted).
[8] J. Kennedy and R. Eberhart (1995), "Particle swarm optimization," Proc. IEEE International Conference on Neural Networks, Vol. 4, 1995, pp. 1942-1948.
[9] Russell C. Eberhart and Yuhui Shi (1998), "Comparison between genetic algorithms and particle swarm optimization," Evolutionary Programming VII, Lecture Notes in Computer Science, Springer, Volume 1447, 1998, pp. 611 - 616.
[10] Jacob Robinson and Yahya Rahmat-Samii (2004), "Particle swarm optimization in electromagnetic," IEEE Transactions on Antennas and Propagation, Volume 52, Issue 2, Feb. 2004, pp. 397 - 407.
[11] T. Martinek and L. Sekanina (2005), "An Evolvable Image Filter: Experimental Evaluation of a Complete Hardware Implementation in FPGA," Evolvable Systems: From Biology to Hardware, vol. 3637 of LNCS, Springer, 2005, pp. 76-85.