Comparison of CGP and Age-Layered CGP Performance in Image Operator Evolution

Karel Slaný

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
slany@fit.vutbr.cz

Abstract. This paper analyses the efficiency of the Cartesian Genetic Programming (CGP) methodology in the image operator design problem at the functional level. The CGP algorithm is compared with an age layering enhancement of the CGP algorithm by the means of achieved best results and their computational effort. Experimental results show that the Age-Layered Population Structure (ALPS) algorithm combined together with CGP can perform better in the task of image operator design in comparison with a common CGP algorithm.

1 Introduction

Cartesian Genetic Programming (CGP) was introduced by J. F. Miller and P. Thomson in 1999 [8]. When comparing with a standard genetic programming approach, CGP represents solution programs as bounded $(c \times r)$-node directed graphs. It utilizes only a mutation operator which is operating in small populations.

The influence of different aspects of the CGP algorithm have been investigated; for example the role of neutrality [2, 14], bloat [6], modularity [13] and the usage of search strategies [7]. In order to evolve more complicated digital circuits, CGP has been extended to operate at the functional level [9]. Gates in the nodes were replaced by high-level components, such adders, shifters, comparators, etc. This approach has been shown to suite well for the evolution of various image operators such as noise removal filters and edge detectors. In several papers it has been reported, that the resulting operators are human-competitive [5, 10].

The literature describes many techniques designed to preserve diversity in population as pre-selection [1], crowding [3], island models [11], etc. Escaping the local optima can only be achieved by the introduction of new, randomly generated individuals into the population. The simplest method, how to implement this technique, is restarting the evolution multiple times with different random number generator seeds. This increases the chances to find the global optima. But the evolutionary algorithm must have enough time to find a local optima. More sophisticated methods do not restart the process from scratch, but periodically introduce randomly generated individuals into the existing population. The algorithm has to ensure, that the new incomes are not easily discarded by existing better solutions and receive enough time to evolve. Such an algorithm is the Age-Layered Population Structure (ALPS) algorithm introduced by G. Hornby in 2006 [4].

This paper deals with the comparison of a standard CGP based algorithm with a modification of ALPS in the case of image operator evolution. The performance of these algorithms is measured by comparing the achieved fitness during the evolution process.

2 Image Filter Evolution

As introduced in [9, 10], an image operator can be considered as a digital circuit with nine 8-bit inputs and a single 8-bit output. This circuit can then process grey-scaled 8-bit per pixel encoded images. Every pixel value of the filtered image is calculated using a corresponding pixel and its neighbours in the processed image.

2.1 CGP at the Functional Level

In CGP a candidate graph (circuit) solution is represented by an array of c (columns) $\times$ r (rows) interconnected programmable elements (gates). The number of the circuit inputs, n_i, and outputs, n_o, is fixed through the entire evolutionary process. Each element input can be connected to the output of any other element, which is placed somewhere in the previous l columns. Feedback connections are not allowed. The l parameter defines the interconnection degree and influences the size of the total search space. In case of $l = 1$ only neighbouring columns are allowed to connect; on the other hand, if $l = c$ then a element block can connect to any element in any previous column. Each programmable element is allowed to perform one function of the functions defined in the function set Γ. The functions stored in Γ influence the design level of the circuit. The set Γ can represent a set of gates or functions defined at a higher level of abstraction. Every candidate solution is encoded into a chromosome, which is a string of $c \times r \times (e_i + 1) + n_o$ integers as shown in fig. 1. The parameter e_i is the number of inputs of the used programmable elements. If we use two-input programmable elements, then $e_i = 2$.

CGP, unlike genetic programming (GP), operates with a small population of λ elements, where usually $\lambda = 5$. The initial population is randomly generated. In each generation step the new population consists of a parent, which is the fittest individual from the previous generation, and its mutants . In case of multiple individuals sharing the best fitness, the individual, which was not selected to be the parent in the previous generation, is selected to be the parent of the next generation. This is used mainly to ensure diversity of the small population. The mutants are created by a mutation operator, which randomly modifies genes of an individual. Crossover operator is not used in CGP. In various applications, which utilize CGP, crossover has been identified to have rather destructive effect. In the particular case of image filter evolution at the functional level the crossover

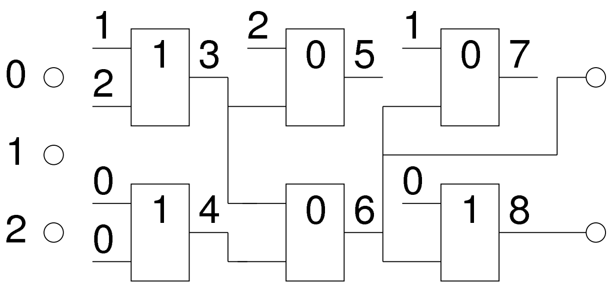

Fig. 1. An example of a 3-input circuit. Parameters are set to $l = 3$, $c = 3$, $r = 2$, $\Gamma = \{+(0), -(1)\}$. Gates 2 and 7 are not used. the chromosome looks like 1,2,1, 0,0,1, 2,3,0, 3,4,0 1,6,0, 0,6,1, 6, 8. The last two numbers represent the connection of the outputs.

operator does not demonstrate a very destructive behaviour [12]. However no benefits of utilizing crossover operators have been shown.

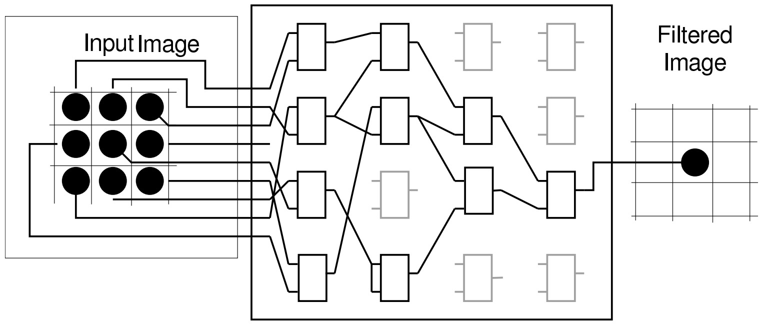

Fig. 2. Example of an image filter consisting of a 3×3 input and output matrix connected to the image operator circuit.

In image operator evolution the goal of CGP is to find a filter which minimizes the difference between the filtered image I_f and the reference image I_r, which must be present for a particular input image I_i. If the input and the reference image are of the size $K \times L$ pixels and a square 3×3 input and output matrix is used, then the filtered image has the size of $(K-2) \times (L-2)$ pixels. Because of the shape of the matrices the pixels at the edge of the filtered images can be read but they cannot be written. For an 8-bit grey-scale image the fitness value f_v can be defined by the following expression:

$$f_v = \sum_{i=1}^{K-2} \sum_{j=1}^{L-2} |I_f(i,j) - I_r(i,j)| \,. \tag{1}$$

The expression (1) summarizes the differences of corresponding pixels in the filtered and the reference image. When f_v drops to $f_v = 0$ then it means that the images I_f and I_r of the size $K \times L$ pixels are indistinguishable from each other (except the pixels on the edges). Papers [10, 5] show that this approach leads to good image filters. The results are satisfiable even in cases, when only a single image in the fitness function is used.

2.2 ALPS Paradigm for CGP

Premature convergence has always been a problem in genetic algorithms. One way to deal with this problem is to increase mutation probability. This will keep the diversity high. But this can also very likely destroy good alleles, which are already present in the population. When the mutation rate is set too high, then the genetic operator cannot explore narrow surroundings of a particular solution. Large population sizes can also be a solution to the diversity problem, but then more time is needed to search for a good solution.

The Age-Layered Population Structure (ALPS) [4] algorithm adds time tags into a genetic algorithm. These tags represent the age of a particular candidate solution in the population. The candidate solutions are only allowed to mutually interact and compete in groups, which contain solutions with similar age. By structuring the population into age-based groups, the algorithm ensures that a newly generated solution cannot easily be outperformed by a better solution, which is already present in the population. Also, new, randomly generated solutions are added in regular periods. These are the two main parts which maintain population diversity in the ALPS algorithm.

The age measure is the count of how many generations the candidate solution has been evolving in the population. Newly generated solution start with the age of 0. Individuals which were generated by an genetic operator such as mutation or crossover receive the age value of the oldest parent increased by 1. Every time a candidate solution is taken to be a parent, its age increases by 1. In cases a candidate solution is used multiple times to be a parent during a generation cycle, its age is still increased by 1 only once.

The population is defined as a structure of age layers, which restrict the competition and breeding among candidate solutions. Each layer, except for the last layer, has a maximum age limit. This limit allows only solutions with the age below its value to be in the layer. The last layer has no maximal age restriction, so that any best solution can stay in this layer for an unlimited time. The structure of the layers can be defined in various ways. Different systems for setting the age limits, which can be used, are shown in tab. 1. The limit values are multiplied by an *age-gap* parameter, thus obtaining maximum age of an individual in each layer.

The ALPS algorithm was designed for maintaining diversity in difficult GP based problems. Its main genetic operator is crossover with tournament selection. But crossover is not used in CGP, instead only mutation and elitism are utilized. Some modifications need to be done in order to make the ALPS algorithm work

aging scheme	maximum age in layer					
	0	1	2	3	4	5
linear	1	2	3	4	5	6
Fibonacci	1	2	3	5	8	13
polynomial	1	2	4	9	16	25
exponential	1	2	4	8	16	32
factorial	1	2	6	24	120	720

Table 1. Various aging scheme distributions, which can be used in the ALPS algorithm.

with CGP in the case of image filter evolution. These changes mainly involve removing the crossover operator from the algorithm.

During every generation cycle each layer interacts with other layers by sending individuals to the next layer or by receiving new individuals from the previous layer. The original ALPS algorithm starts with randomly initialized first layer. Other layers are empty and will be filled during the process of evolution. The individuals grow older and move to next layers or are discarded. In regular intervals the bottom layer is replaced by newly generated individuals. Let us call the parameter describing this behaviour *randomize-period*. The value of the parameter stands for the number of generations between two randomization of the bottom layer.

Whenever the age of a member in a particular layer exceeds the age limit for this layer, then such a member is moved to the next layer. This formulation can cause trouble in implementation of the algorithm. Just imagine the case, when new members have to be moved into a fully occupied higher layer. In this case, the layer, which has to accept members or offspring from a previous layer, is divided into halves. The first half is used for generating new members from the original layer and the second half is populated by the newly incomes. After this step both halves behave again as a single layer.

Also elitism, similar to CGP, is used. Each layer keeps its best evolved member and only replaces it with an individual with a better or a least the same fitness. If this individual is selected to be a parent, its age is not increased in order to keep it in the current layer.

During the process of evolution the size limits of the population do not change, but the number of individuals in the layers may vary. This is caused by the fact, that in the initial phase the algorithm starts with only one populated layer. Also in certain situations a layer can lead into extinction, when current layer members and its offspring are moved into next layer and no newcomers have arrived.

3 Experimental Set-up

In order to evolve image operators a set of function has been adopted from [10].

The CGP and modified ALPS-CGP algorithms are used to find a random shot noise filter and a Sobel filter using a set of three pictures as training data.

ID	Function	Description	ID	Function	Description
0	$x \vee y$	binary or	4	$x +_{sat} y$	saturated addition
1	$x \wedge y$	binary and	5	$(x + y) >> 1$	average
2	$x \oplus y$	binary xor	6	$Max(x, y)$	maximum
3	$x + y$	addition	7	$Min(x, y)$	minimum

Table 2. Function set used in the experiments. All functions have 8-bit inputs and outputs.

The evolved image operators are connected to a 3×3 input and output mask. The size of the pictures is 256×256 pixels.

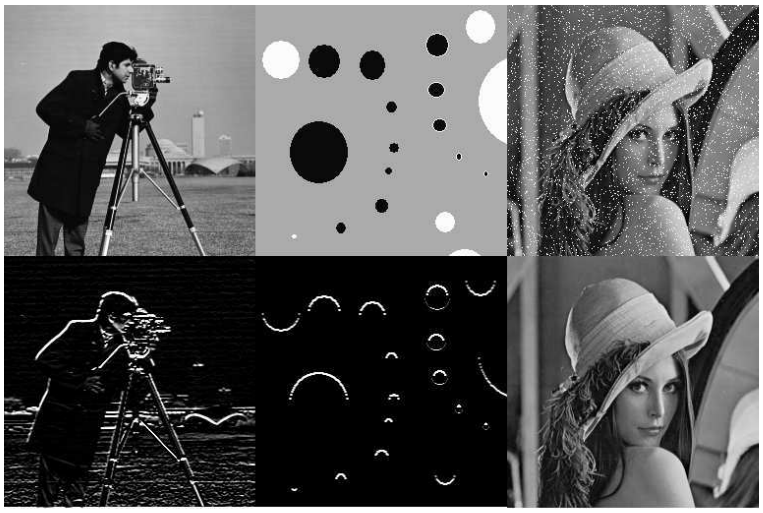

Fig. 3. Images used in described experiments. Top row contains input images entering the evolved image operators. Bottom row shows reference images used for fitness evaluation. The evolution searches for a Sobel operator and for a random shot noise removal filter.

The experiment consists of two main groups which differ in the way how the evolved image operators is defined. In the first set of experiments the image operator shall consist of 8 columns $\times$ 6 rows of programmable elements with the interconnection parameter $l = 1$. The value $l = 1$ allows only interconnection of neighbouring columns. This is because of an easy implementation as a pipelined filter in hardware. The second set of experiments utilizes chromosomes which consist of only a single row with 48 programmable elements with the interconnection parameter set to $l = 48$. This value ensures unlimited interconnection, except that the output of an evolved circuit cannot connect directly to its input.

The CGP algorithm uses population size of 8 individuals. Mutation probability is set to 8% in both algorithms. The ALPS-CGP algorithm uses 5 layers. Each layer can hold up to 8 individuals. The polynomial aging scheme is used with *age-gap* = 20. The bottom layer is regenerated with random individuals every *randomize-period* = 5 cycles. Each evolutionary process of 10000 generations is repeated 100 times. Average data are used to compare both of the two algorithms.

The measured data are compared according to the evaluation number. That means the fitness values are plotted against the number of evaluations which the algorithm has performed rather than to the generation it has reached. This is done because of the fact, that the ALPS-CGP algorithm uses larger populations. Thus it has a greater chance of exploring larger amounts of search-space in a single generation cycle than the CGP algorithm.

3.1 Results

In the first set of experiments image operators with rectangular arrangement of programmable elements with the interconnection parameter $l = 1$ were evolved.

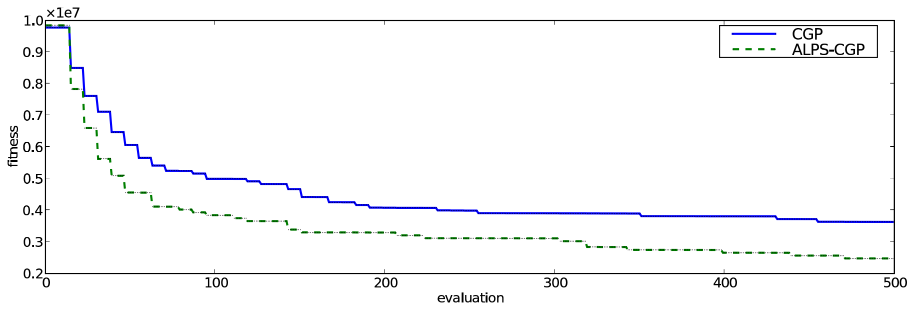

Fig. 4. The progression of the fitness value during the first 500 evaluations when evolving the Sobel filter by using the camera pictures. The image operator consists of 8×6 elements.

The graphs in fig. 4, 5 and 6 show that the ALPS-CGP algorithm is behaving slightly better than the standard CGP algorithm. In average we have obtained similar or better image operators in less time than using only the CGP algorithm. The average fitness values, measured after reaching final generation number, are summarized in tab. 3.

In the second set of experiments the evolutionary process searched for image operators with a linear structure with high interconnection parameter, allowing more complex structures to be designed. This also increases the search space of all possible solutions. The graphs in fig. 7, 8 and 9 show the behaviour of the algorithms in the first 500 evaluations. Again we have obtained similar results.

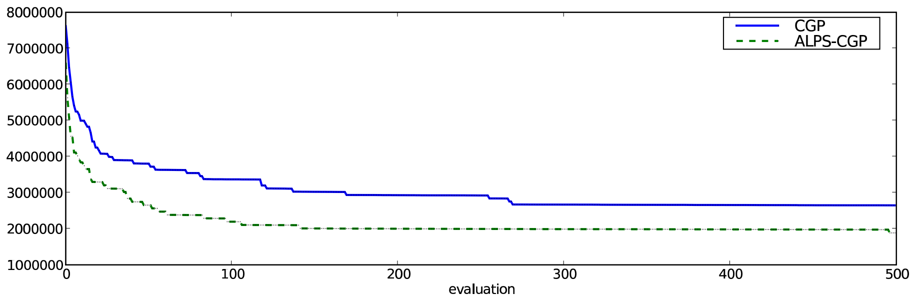

Fig. 5. The progression of the fitness value during the first 500 evaluations when evolving the Sobel filter by using the circle pictures. The image operator consists of 8×6 elements.

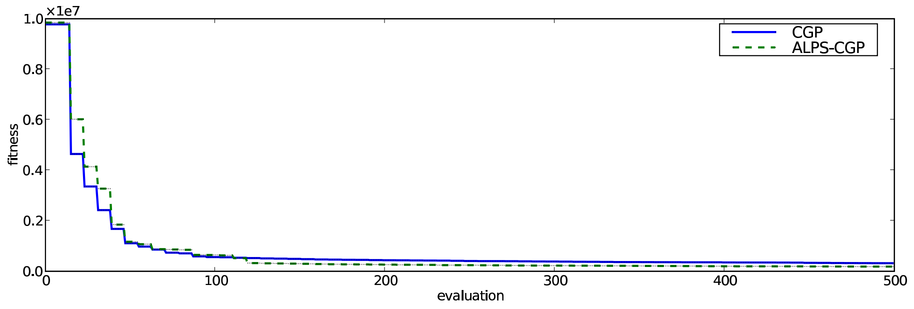

Fig. 6. The progression of the fitness value during the first 500 evaluations when evolving the random shot noise removal filter by using the Lena pictures. The image operator consists of 8×6 elements.

4 Discussion

In the first group of experiments we can observe a performance gain, when using the ALPS-CGP algorithm. In the initial phase, when the algorithms are started, the ALPS-CGP algorithm converges faster to local optima than the simple CGP algorithm. But then still keeps improving the fitness value of the evolved image operators. The ALPS algorithm in average achieves better fitness values.

In the second group of experiments the algorithms show similar behaviour as in group one. Again ALPS has achieved better fitness values than the CGP algorithm. Only in the case of the random shot noise filter the algorithms show approximately the same performance. This may be because the random shot filter is easier to construct from the function set Γ. Also the second group of experiments showed, that the evolved image operators achieve slightly worse results, as in the first case. The explanation may be the fact, that in the second case the elements, which the image operator consists of, are allowed to connect

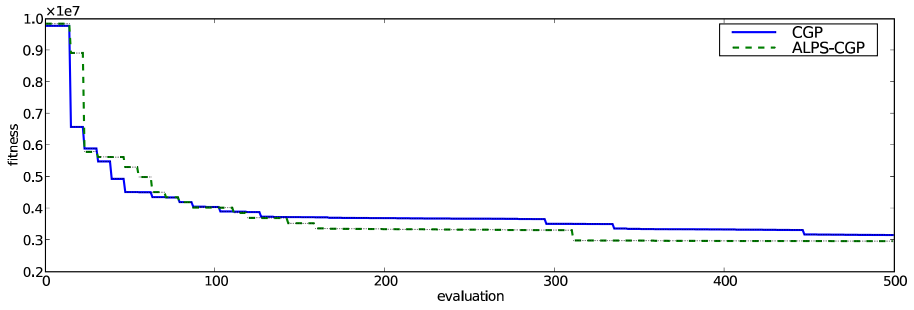

Fig. 7. The progression of the fitness value during the first 500 evaluations when evolving the Sobel filter by using the camera pictures. The image operator consists of 1×48 elements.

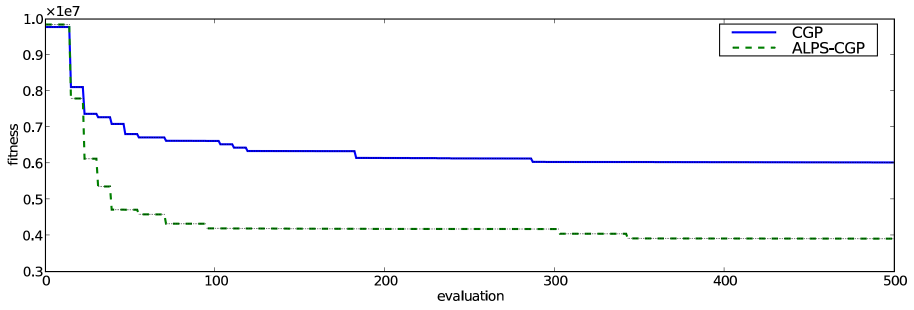

Fig. 8. The progression of the fitness value during the first 500 evaluations when evolving the Sobel filter by using the circle pictures. The image operator consists of 1×48 elements.

more freely in comparison with the first experimental set. The search space is much greater. Another explanation may be the fact, that the configuration of the image operator is taken from [10], and might be more optimized for the role of an image operator.

The whole system is implemented in SW. To finish all runs in the first or second set of experiments 6 days of computation time are needed when using a two 1800 MHz dual-core AMD Opteron system. This is mainly caused by the time-consuming fitness function - every pixel in the training images has to be computed separately. The slow performance is also a drawback, when the optimal performance parameters need to be found. The ALPS-CGP algorithm is not much more complex than the CGP algorithm. In fact the main differences are the time-tags and the consequent restriction. These are not difficult to implement in hardware. Therefore the next step will be implementing the system in a FPGA. This will give a greater chance of evaluating the ALPS-CGP and

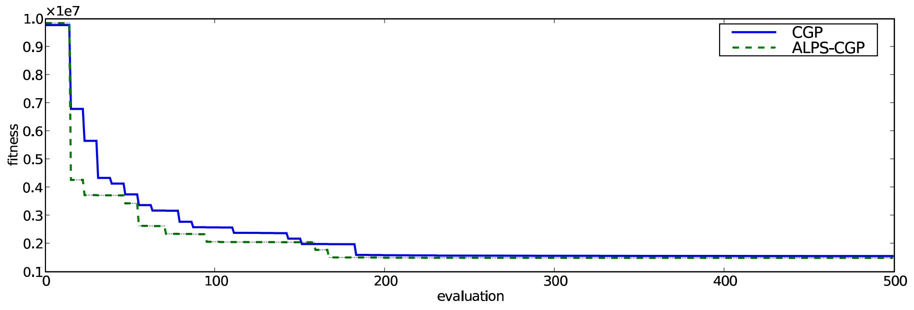

Fig. 9. The progression of the fitness value during the first 500 evaluations when evolving the random shot noise removal filter by using the Lena pictures. The image operator consists of 1×48 elements.

set of experiments	image set	average fitness CGP	average fitness ALPS
1	camera Sobel	2 034 740	1 371 492
1	circle Sobel	1 948 039	1 345 093
1	Lena impulse	52 980	31 681
2	camera Sobel	2 323 204	1 893 629
2	circle Sobel	2 176 023	1 580 389
2	Lena impulse	47 557	31 284

Table 3. The average achieved fitness after finishing 10000 generations.

CGP performance. Also larger input masks can be used, which can lead to better image operators.

5 Conclusions

An analysis of the performance of a standard CGP approach and an ALPS enhanced CGP algorithm in the task of image operator evolution was performed. The performance of the algorithms was measured in six cases of system settings. Experiments have shown that the ALPS-CGP algorithm performs better than the standard CGP algorithm. However in more difficult cases the performance of the ALPS-CGP algorithm appears not to be much superior. Further experiments, including hardware implementation, are needed to receive a better comparison of the two algorithms.

Acknowledgements

This work was 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 intention No. MSM 0021630528 – Security-Oriented Research in Information Technology.

References

[1] Daniel J. Cavicchio. *Adaptive search using simulated evolution*. PhD thesis, University of Michigan, 1970.

[2] Mark Collins. Finding needles in haystacks is harder with neutrality. *Genetic Programming and Evolvable Machines*, 7(2):131–144, 2006.

[3] Kenneth Alan DeJong. *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[4] Gregory Scott Hornby. Alps: the age-layered population structure for reducing the problem of premature convergence. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 815–822, New York, NY, USA, 2006. ACM.

[5] Tomáš Martínek and Lukáš Sekanina. An evolvable image filter: Experimental evaluation of a complete hardware implementation in fpga. *Lecture Notes in Computer Science*, 2005(3637):76–85, 2005.

[6] Julian Francis Miller. What bloat? cartesian genetic programming on boolean problems. In *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 295–302, 2001.

[7] Julian Francis Miller and Stephen L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Trans. Evolutionary Computation*, 10(2):167–174, 2006.

[8] Julian Francis Miller and Peter Thomson. Cartesian genetic programming. In *Proceedings of the 3rd European Conference on Genetic Programing*, Lecture Notes in Computer Science, pages 121–132, Berlin, 1999. Springer Verlag.

[9] Lukáš Sekanina. Image filter design with evolvable hardware. *Lecture Notes in Computer Science*, 2002(2279):255–266, 2002.

[10] Lukáš Sekanina. *Evolvable Components: From Theory to Hardware*. Springer-Verlag, Berlin Heidelberg, 2004.

[11] Zbigniew Skolicki and Kenneth Alan DeJong. Improving evolutionary algorithms with multi-representation island models. In *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, pages 420–429. Springer, 2004.

[12] Karel Slaný and Lukáš Sekanina. Fitness landscape analysis and image filter evolution using functional-level cgp. *Lecture Notes in Computer Science*, 2007(4445):311–320, 2007.

[13] James Alfred Walker and Julian Francis Miller. Investigating the performance of module acquisition in cartesian genetic programming. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1649–1656, New York, NY, USA, 2005. ACM.

[14] Tina Yu and Julian Francis Miller. Neutrality and the evolvability of boolean function landscape. In *EuroGP '01: Proceedings of the 4th European Conference on Genetic Programming*, pages 204–217, London, UK, 2001. Springer-Verlag.