An Efficient Selection Strategy for Digital Circuit Evolution

Zbyšek Gajda and Lukáš Sekanina

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

Abstract. In this paper, we propose a new modification of Cartesian Genetic Programming (CGP) that enables to optimize digital circuits more significantly than standard CGP. We argue that considering fully functional but not necessarily smallest-discovered individual as the parent for new population can decrease the number of harmful mutations and so improve the search space exploration. This phenomenon was confirmed on common benchmarks such as combinational multipliers and the LGSynth91 circuits.

1 Introduction

Cartesian Genetic Programming (CGP) exhibits many interesting features, especially for circuit design. When CGP is applied to reduce the number of gates in digital circuits it starts with the fitness function which evaluates the circuit behavior only. Once one of candidate circuits conforms to the behavioral specification the number of gates becomes important and reflected in the fitness value. This method which will be called the *standard CGP* in this paper, is widely adopted in literature [1, 2, 3, 4].

We have shown in our previous work [5] that area-efficient digital circuits can be evolved even if the requirement on the gate reduction is not specified explicitly. The method is based on modifying the selection mechanism and fitness function of the standard CGP. In this paper, we provide further experimental evidence for this phenomenon. In addition to testing the method using popular benchmarks such as multipliers we will perform experimental evaluation using the LGSynth91 benchmark circuits. We hypothesize that the neutral search and redundancy of encoding of CGP (as demonstrated in [6, 7, 8]) are primarily responsible for this phenomenon. We argue that considering fully functional but not necessarily smallest-discovered individuals as parents improve the search space exploration in comparison with the standard CGP.

The rest of the paper is organized as follows. Section 2 surveys the basic (standard) version of CGP. Benchmark problems are presented in Section 3. Proposed modification of CGP is formulated in Section 4. The results of experiments are summarized in Section 5. Section 6 deals with the analysis of results on the basis of measurement of non-destructive mutations. Finally, conclusions are given in Section 7.

2 Cartesian Genetic Programming

Cartesian Genetic Programming is a widely-used method for evolution of digital circuits [9, 1]. In CGP, a candidate entity (circuit) is modeled as an array of n_c (columns) $\times$ n_r (rows) of programmable nodes (gates). The number of inputs, n_i, and outputs, n_o, is fixed. Each node input can be connected either to the output of a node placed in previous l columns or to one of the program inputs. The l-back parameter, in fact, defines the level of connectivity and thus reduces/extends the search space. For example, if l=1 only neighboring columns may be connected; if $n_r = 1$ and $l = n_c$ then full connectivity is enabled. Feedback is not allowed. Each node is programmed to perform one of n_a-input functions defined in the set Γ (n_f denotes $|\Gamma|$). Each node is encoded using $n_a + 1$ integers where values $1 \ldots n_a$ are the indexes of the input connections and the last value is the function code. Every individual is encoded using $n_c.n_r.(n_a + 1) + n_o$ integers. Figure 1 shows an example of a candidate circuit and its chromosome.

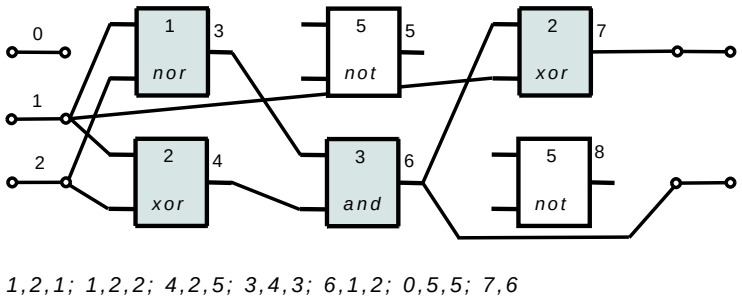


Fig. 1. An example of a candidate circuit in CGP and its chromosome: $l = 3$, $n_c = 3$, $n_r = 2$, $n_i = 3$, $n_o = 2$, $n_a = 2$, Γ = {NOR (1), XOR (2), AND (3), NAND (4), NOT (5)}.

CGP operates with the population of $1+\lambda$ individuals (typically, λ is between 1 and 20). The initial population is constructed either randomly or by a heuristic procedure. Every new population consists of the best individual of the previous population and its λ offspring. The offspring individuals are created using a point mutation operator which modifies h randomly selected genes of the chromosome, where h is the user-defined value.

There is one important rule for selection of the parent. In case when two or more individuals can serve as the parent, the individual which has not served as the parent in the previous generation will be selected as the new parent. This strategy is important because it ensures the diversity of population [7]. The algorithm is terminated when the maximum number of generations is exhausted or a sufficiently working solution is obtained.

Because we will deal with digital circuit evolution, let us consider the fitness function for that case only. The goal is to obtain a perfectly working circuit

(all assignments to the inputs have to be tested) with the number of gates as low as possible. Additional criteria can be included; however, we will not deal with them in this paper. The most effective strategy to the fitness calculation proposed so far is as follows: The fitness value of a candidate circuit is defined as [3]:

$$fit1 = \begin{cases} b & when \quad b < n_o 2^{n_i}, \\ b + (n_c n_r - z) & otherwise, \end{cases} \tag{1}$$

where b is the number of correct output bits obtained as response for all possible assignments to the inputs, z denotes the number of gates utilized in a particular candidate circuit and $n_c.n_r$ is the total number of available gates. It can be seen that the last term $n_c n_r - z$ is considered only if the circuit behavior is perfect, i.e. $b = b_{max} = n_o 2^{n_i}$. We can observe that the evolution has to discover a perfectly working solution firstly while the size of circuit is not important. Then, the number of gates is optimized.

The encoding used in CGP is redundant since there may be genes that are entirely inactive. These genes do not influence the phenotype, and hence the fitness. This phenomenon is often referred to as neutrality. The role of neutrality has been investigated in detail [10, 6, 7]. For example, it was found that the most evolvable representations occur when the genotype is extremely large and in which over 95% of the genes are inactive [7]. But for example, Collins has shown that for some specific problems the neutrality-based search is not the best solution [11]. Miller has also identified that the problem of bloat is insignificant for CGP [12].

3 Benchmark Problems

Design of small multipliers is the most popular benchmark problem for the gate level circuit evolution. Because the direct CGP approach is not scalable it works only for 4-bit multipliers (i.e. 8-input/8-output circuits) and smaller. Table 1 summarizes the best known results for various multipliers according to [1, 2]. CGP was used with two-input gates, $l = n_c$, $\lambda = 4$, $h = 3$, remaining parameters are given in Table 1. CGP was seeded using conventional designs. The fitness function was constructed according to equation 1.

CGP is capable of creating innovative designs for this class of circuits. However, it is important to carefully initialize CGP parameters. For example, in order to reduce the search space the function set should contain just the logic functions that are important for multipliers (the solutions denoted as Best CGP in Table 1 were obtained using $\Gamma = \{x\ AND\ y,\ x\ XOR\ y,\ (not\ x)\ AND\ y\}$). However, the gate $(not\ x)\ AND\ y$ is not usually considered as a single gate in digital design. Its implementation is constructed using two gates: AND and NOT. Hence we also included 'Recalc. CGP' to Table 1 which is the result recalculated when one considers $(not\ x)\ AND\ y$ as two gates in the multipliers shown in [2].

For further comparison of the standard CGP and proposed method we have selected 16 circuits from the LGSynth91 benchmark suite [13] (see Table 4). In

Table 1. The number of two-input gates in multipliers according to [1, 2].

Multiplier	Best conv.	Best CGP	Recalc. CGP	$n_r \times n_c$	Max. gener.
2b×2b	8	7	9	1×7	10k
3b×2b	17	13	14	1×17	200k
3b×3b	30	23	25	1×35	20M
4b×3b	47	37	44	1×56	200M
4b×4b	64	57	67	1×67	700M

this case we have utilized CGP in the postsynthesis phase, i.e. CGP is employed to reduce the number of gates in already synthesized circuits. In this paper, we have used the ABC tool to perform (conventional) synthesis [14]. Each circuit is represented as a netlist of gates in the BLIF format (Berkeley Logic Interchange Format).

4 The Proposed Modification of CGP

From the perspective of this paper, the fitness function and selection strategy are the most interesting features of the standard CGP. Because $(1 + \lambda)$ strategy is used, the highest-scored individual p (whose fitness value will be denoted f_p) is always preserved. The result of evolution is then just the highest-scored individual of the last generation in the standard CGP.

Consider a situation in which a fully working circuit has already been obtained ($b = b_{max}$) and the number of gates is optimized now. If the mutation operator creates an individual x with the fitness value f_x and $f_x \geq f_p$ then x will become a new parental solution p (assuming that there is no better result of mutation in the population). However, if the mutation operator creates individual y with the fitness value f_y and $(f_y < f_p) \wedge (f_y \geq b_{max})$ then p will be selected as parent for the new population and y will be discarded (assuming that the fitness values of other solutions are lower than f_y). In this way, many new fully functional solutions, however slightly worse than the parent, are lost. We will demonstrate in Section 5 that considering individual y for which the property $(f_y < f_p) \wedge (f_y \geq b_{max})$ holds as a new parent is beneficial for the efficient search process.

The new selection strategy and fitness function is proposed *only* for the situation when the number of gates is optimized, i.e. the fitness value of the best individual is higher than or equal to b_{max}. Otherwise, the algorithm works as the standard CGP. As the best individual found so far will not be copied to the new population automatically, it is necessary to store it in an auxiliary variable. Let β denote the best discovered solution and let f_β be its fitness value. In the first population, β is initialized using p.

Assume that $x_1 \ldots x_\lambda$ are individuals (with fitness values $f_{x_1} \ldots f_{x_\lambda}$) created from the parental solution p using the mutation operator and $f_\beta \geq b_{max}$ (i.e. we are in the gate reduction phase now). Because the best individual β and parental individual p are not always identical we have to determine their new instances

β' and p' separately. The best-discovered solution is defined as:

$$\beta' = \begin{cases} \beta & when \quad f_\beta \geq f_{x_i}, i = 1 \ldots \lambda, \\ x_j & otherwise, \end{cases} \tag{2}$$

where x_j is the highest-scored individual for which $f_{x_j} > f_\beta$ holds. If multiple individuals exist that have higher fitness than f_β in $\{x_1 \ldots x_\lambda\}$, randomly choose the best one of them. The new parental individual is defined as:

$$p' = \begin{cases} p & when \quad \forall i, i = 1 \ldots \lambda : f_{x_i} < b_{max} \\ x_j & otherwise, \end{cases} \tag{3}$$

where x_j is a randomly selected individual from those in $\{x_1 \ldots x_\lambda\}$ which obtained the fitness score higher than or equal to b_{max}. In other words, the new parent must be a fully functional solution; however, the number of gates is not important for its selection. Note that the result of evolution is no longer p but β. The proposed strategy will be denoted fit2.

5 Results

5.1 Experimental Setup

CGP is used according to its definition in Section 2. In this paper, we always use $n_r = 1$ and $l = n_c$. The initial population is generated either randomly or using a solution obtained from a conventional synthesis method.

If CGP is applied as a postsynthesis optimizer then the number of gates of the result of conventional synthesis is denoted as m (it is assumed that each of the gates has up to γ inputs). Then CGP will operate with the parameters $n_c = m, n_r = 1, l = n_c, n_a = \gamma$.

In all experiments $\lambda = 14$, $\gamma = 2$ and h is between 1 and 14 (the mean value is 7). We have used $\Gamma' = \{and, or, not, nand, \ nor, xor, identity, const_1, const_0\}$ where *not* and *identity* are unary functions (taking the first input of the gate) and $const_k$ is constant generator with the value k.

Each experiment is repeated ten times with the 100 million generation limit. In all experiments the standard fitness function of CGP (denoted fit1) is compared with the method presented in Section 4 (denoted fit2).

5.2 Evolution From a Random Population

In the first experiment, we have evolved multipliers with up to four-bit operands from randomly generated initial population. According to recommendations of [7], we intentionally allowed relatively long chromosomes to be used by CGP. The n_c values were set on the basis of ABC synthesis (see Table 3, the seed).

Table 2 summarizes the number of gates (the best and mean values), mean number of generations to reach b_{max} and the success rate for fit1 and fit2. As design of 2b×2b and 3b×2b multipliers is easy for CGP, we will mainly analyse

the results for larger problem instances (here and in next sections). It can be seen that fit2 gives better results than fit1. However, the mean number of generations is higher for fit2. We have obtained almost identical minimum number of gates when compared with [2] (also in Table 1, Best CGP) even when CGP is randomly initialized and a non-problem specific set of gates is utilized.

Table 2. The best-obtained and mean number of gates for the multiplier benchmarks when CGP starts from randomly generated initial population.

Circuit	Alg.	n_c	gates (best)	gates (mean)	mean # gener.	succ. runs
2b × 2b	fit1	7	7	7	2 738	100%
	fit2		7	7	2 777	100%
3b × 2b	fit1	16	13	13	651 297	100%
	fit2		13	13	741 758	100%
3b × 3b	fit1	57	25	27.7	476 812	100%
	fit2		23	23.4	625 682	100%
4b × 3b	fit1	125	46	52.7	2 714 891	100%
	fit2		37	43.1	4 271 179	100%
4b × 4b	fit1	269	110	128.3	29 673 418	90%
	fit2		60	109.4	37 573 311	70%

5.3 Post-synthesis Optimization

The second set of experiments compares fit1 and fit2 when CGP is applied to reduce the number of gates in already functional circuits. We compared three approaches to seeding the initial population in case of multipliers. The resulting multipliers of the ABC tool are taken as seeds in the first group of experiments (denoted 'seed:ABC' in Table 3). The second group of experiments is seeded using the best multipliers reported in paper [2] (denoted 'seed:Tab. 1' in Table 3). The seeds of the third group of experiments are created manually as combinational carry save multipliers according to [15] (denoted 'seed: CM' in Table 3). Table 3 shows that fit2 can produce more compact designs (see the 'best' column) than fit1. The mean number of gates is given in generation 1M, 2M, 5M, 10M, 20M, 50M and 100M (M=10^6). It can be seen that the best solution is improving over time.

The best-evolved multiplier (4b × 4b) is composed of 56 gates (taken from Γ' which does not consider the AND gate with one input inverted as a single gate). The best circuit presented in [2] consists of 57 gates taken from Γ (i.e., 67 gates when Γ' is used). We can also express the implementation cost in terms of transistors used. While the 56-gate multiplier is composed of 400 transistors the multiplier reported in [2] consists of 438 transistors. It is assumed that the number of transistors required to create a particular gate is as follows: nand (4 tr.), nor (4 tr.), or (6 tr.), and (6 tr.), not (2 tr.) and xor (10 tr.) [15].

Table 3. The best-obtained and mean number of gates in generations 1M...100M for the multiplier benchmarks when CGP is seeded by functional solutions of different type.

seed: ABC	Alg.	seed	best	1M	2M	5M	10M	20M	50M	100M
2b × 2b	fit1	17	7	7	7	7	7	7	7	7
	fit2		7	7	7	7	7	7	7	7
3b × 2b	fit1	16	13	13	13	13	13	13	13	13
	fit2		13	13	13	13	13	13	13	13
3b × 3b	fit1	57	26	38.2	36.1	34.3	32.6	31	29.8	28.7
	fit2		23	31.5	28.8	27.2	25	24.5	24.2	23.5
4b × 3b	fit1	125	54	93.2	88.3	79.3	75.6	71.6	66.6	64.4
	fit2		37	80	68	55.9	49.9	46.9	44.1	41.1
4b × 4b	fit1	269	140	212.4	190.6	178.9	170.9	165.2	158.5	152.4
	fit2		68	218.2	182.2	151.3	136.5	121.2	107	93.3
seed: Tab. 1		seed	best	1M	2M	5M	10M	20M	50M	100M
2b × 2b	fit1	9	7	7	7	7	7	7	7	7
	fit2		7	7	7	7	7	7	7	7
3b × 2b	fit1	14	13	13	13	13	13	13	13	13
	fit2		13	13	13	13	13	13	13	13
3b × 3b	fit1	25	23	25	25	24.7	23.9	23.5	23.2	23.1
	fit2		23	25	25	24.7	24.4	24.2	23.5	23.1
4b × 3b	fit1	44	36	38.5	37.8	37.1	36.8	36.8	36.4	36.3
	fit2		35	37.9	37.1	36.5	36.4	36.2	36.2	36.1
4b × 4b	fit1	67	57	59.6	58.8	58	57.8	57.5	57.3	57.1
	fit2		56	59.5	59.2	58.7	58.3	57.2	56.8	56.8
seed: CM		seed	best	1M	2M	5M	10M	20M	50M	100M
2b × 2b	fit1	8	7	7	7	7	7	7	7	7
	fit2		7	7	7	7	7	7	7	7
3b × 2b	fit1	17	13	13	13	13	13	13	13	13
	fit2		13	13	13	13	13	13	13	13
3b × 3b	fit1	30	23	28	28	28	27.8	27.6	26.5	25.8
	fit2		23	28	28	27.6	26.8	25	24.4	23.4
4b × 3b	fit1	45	37	43	43	43	42.4	41.9	40.6	39.2
	fit2		37	43	43	42.6	42.2	41.5	39.9	38.4
4b × 4b	fit1	64	59	62.9	62.6	62.6	62.3	61.5	60.6	60.2
	fit2		59	62.9	62.9	62.8	62.4	62	61.3	60.8

Table 4 gives the best-obtained and mean number of gates for the LGSynth91 benchmark circuits when CGP is seeded by already working circuits. The working circuits (of the size given by n_c) were obtained using ABC initialized with the original LGSynth91 circuits (in the BLIF format) and mapped on two-input gates of Γ'. The 'exp. gates' is the estimated number of two-input gates (after the conventional synthesis) given in [13]. It can be seen that fit2 is more successful than fit1. In general, CGP gives better results than 'exp. gates' because it does not employ any deterministic synthesis algorithm; all the optimizations are being done implicitly, without any structural biases.

Table 4. The best-obtained and mean number of gates for the LGSynth91 benchmarks when CGP starts from the initial solution (of size n_c) synthesized using ABC.

Circuit	n_i	n_o	exp. gates	n_c seed	gates fit1 (best)	gates fit2 (best)	gates fit1 (mean)	gates fit2 (mean)
9symml	9	1	43	216	53	23	68.5	25.5
C17	5	2	6	6	6	6	6	6
alu2	10	6	335	422	134	73	149	89.4
alu4	14	8	681	764	329	274	358	279
b1	3	4	13	11	4	4	4	4
cm138a	6	8	17	19	16	16	16	16
cm151a	12	2	33	34	24	23	24	23
cm152a	11	1		24	22	21	22.1	21.8
cm42a	4	10	17	20	17	17	17	17
cm82a	5	3	27	12	10	10	10	10
cm85a	11	3	38	41	23	22	24.1	22
decod	5	16	22	34	30	26	30	26.1
f51m	8	8	43	146	29	26	32.9	27.3
majority	5	1	9	10	8	8	8	8
x2	10	7	42	60	27	27	29.6	27.4
z4ml	7	4	20	40	15	15	15	15

Figure 2a shows the number of gates of the parent individual p in every 1000th generation during the progress of evolution of the 4b×4b multiplier using fit1 and fit2 (taken from the best runs; seeded by ABC). It can be seen that the parent is different from the best-obtained solution for fit2 (the curve is not monotonic). We can also observe that fit1 provides better result than fit2 in the early stages of the evolution. However, fit2 outperforms fit1 when more generations are allowed for evolution. Figure 2b shows the mean number of gates of the best-obtained individuals (averaged from 10 independent runs).

6 Analysis

We have seen so far that selecting of the parent individual on the basis of its functionality solely (and so neglecting the number of gates) provides slightly

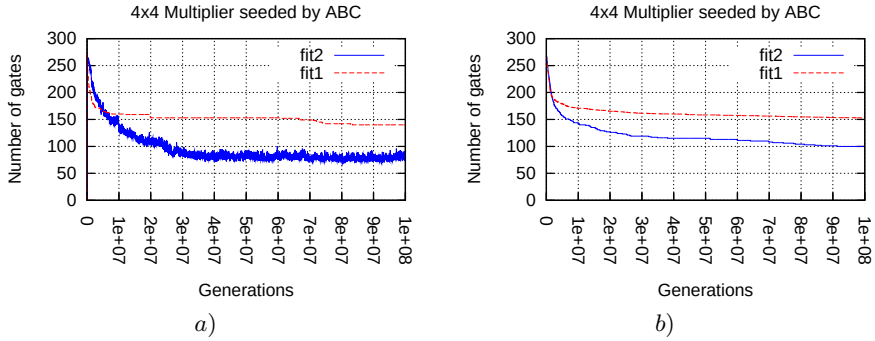


Fig. 2. a) The number of gates of the parent individual (from the best run for 4b×4b multiplier). b) The mean number of gates of the best-obtained individuals β (from 10 runs for 4b×4b multiplier)

better results at the end of evolution (when the goal is to reduce the phenotype size) than the standard CGP. How is it possible that the approach really works? Recall that the fitness landscape is rugged and neutral in case of digital circuit evolution using CGP [6, 8]. Hence relatively simple mutation-based search algorithms are more successful than sophisticated search algorithms and genetic operators such as those developed in the field of genetic algorithms and estimation of distribution algorithms. In the standard CGP, generating the offspring individuals is biased to the best individual that has been discovered so far. The best individual is changed only if a better or equally-scored solution is found. In the proposed method, the changes of the parent individual are more frequent because the only requirement for a candidate individual to qualify as the parent is to be fully functional. Hence we consider the proposed algorithm as more *explorative* than the standard CGP.

Our hypothesis is that if a high degree of redundancy is present in the genotype the proposed method will generate more functionally correct individuals than the standard CGP. And because the fitness landscape is rugged and neutral the proposed method is more efficient in finding compact circuit implementations than the standard CGP. In order to verify this hypothesis we have measured the number of mutations that lead to functionally correct circuits. When CGP is seeded with a working circuit, we have in fact measured the number of neutral and useful mutations. Figure 3 compares the results for fit1 and fit2 in the experiments that are reported in Table 2 and Table 3. The y-axis is labeled as MNM which stands for 'Millions of Non-destructive Mutations'. For small multipliers (2b×2b, 3b×2b) fit1 always yields higher MNM which contradicts with our hypothesis. However, we have already declared that these really small multipliers are not interesting because the problem is easy and an optimal solution can be discovered very quickly. In case of more difficult circuits, fit2 provides higher MNM in most cases, especially when sufficient redundancy is available

(see Fig. 3 a, d). When the best resulting multipliers of paper [2] are used to seed the initial population, fit1 is always higher than fit2 (see Fig. 3 b). It corresponds with a theory that CGP (with almost the zero redundancy in the genotype) has got stuck at a local extreme and fit2 does not have a space to work.

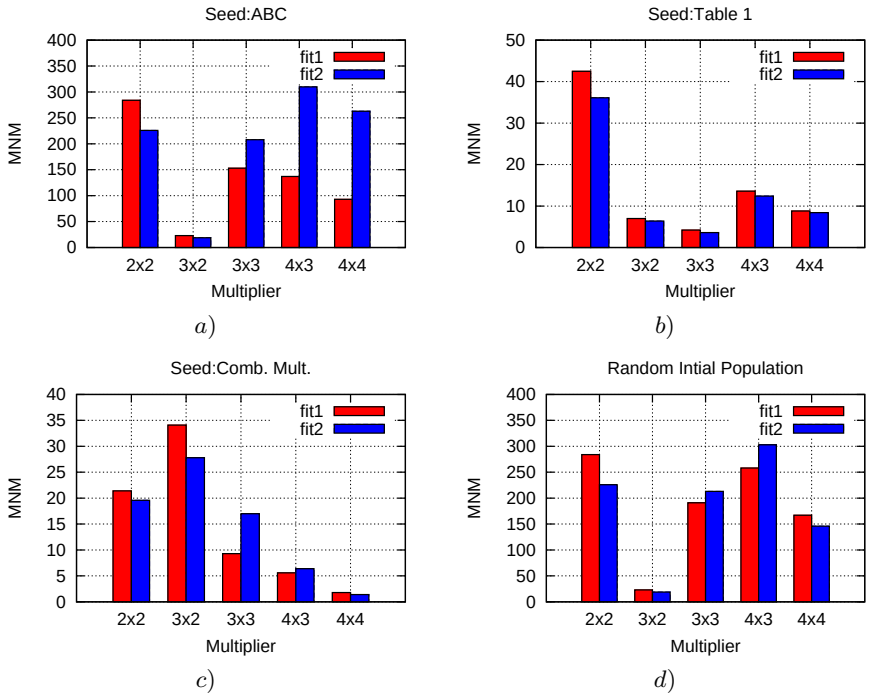
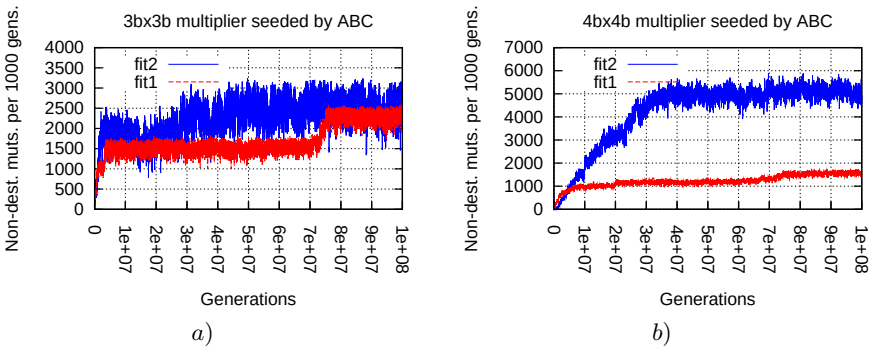


Fig. 3. Millions of Non-destructive Mutations (MNM) for different experiments (mean values given)

The number of non-destructive mutations was counted in every 1000 generations and the resulting value was plotted as a single point to Fig. 4a (3b×3b multiplier) and Fig. 4b (4b×4b multiplier). The best run seeded using ABC is shown in both cases. It is evident that significantly more correct individuals have been generated for fit2 on average. It can also be seen that while fit1 tends to create a relatively stable number of correct individuals in time (the dispersion is approx. 200 individuals for the 4b×4b multiplier), great differences are observable in the number of correct individuals for fit2 (the dispersion is approx. 1000 individuals for the 4b×4b multiplier). That also supports the idea of biased search of fit1.



a) *b*)

Fig. 4. The number of non-destructive mutations per 1000 generations for: a) 3b×3b multiplier; b) 4b×4b multiplier

7 Conclusions

In this paper, we have shown that the selection of the parent individual on the basis of its functionality instead of compactness leads to smaller phenotypes at the end of evolution. The method is especially useful for the optimization of nontrivial circuits when a sufficient redundancy is available in terms of available gates and a sufficient time is allowed for evolution. In the future work we plan to test the proposed method to reduce the size of phenotype in symbolic regression problems.

Acknowledgments

This work was partially supported by the grant Natural Computing on Unconventional Platforms GP103/10/1517, the BUT FIT grant FIT-10-S-1 and the research plan Security Oriented Research in Information Technology MSM 0021630528.

References

[1] Miller, J.F., Job, D., Vassilev, V.K.: Principles in the Evolutionary Design of Digital Circuits – Part I. Genetic Programming and Evolvable Machines **1**(1) (2000) 8–35

[2] Vassilev, V., Job, D., Miller, J.: Towards the Automatic Design of More Efficient Digital Circuits. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, IEEE Computer Society (2000) 151–160

[3] Kalganova, T., Miller, J.F.: Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In: The First NASA/DoD Workshop on Evolvable Hardware, IEEE Computer Society (1999) 54–63

[4] Gajda, Z., Sekanina, L.: Reducing the number of transistors in digital circuits using gate-level evolutionary design. In: 2007 Genetic and Evolutionary Computation Conference, ACM (2007) 245–252
[5] Gajda, Z., Sekanina, L.: When does cartesian genetic programming minimize the phenotype size implicitly? In: Genetic and Evolutionary Computation Conference, ACM – Accepted (2010)
[6] Vassilev, V.K., Miller, J.F.: The advantages of landscape neutrality in digital circuit evolution. In: ICES '00: Proceedings of the Third International Conference on Evolvable Systems. Volume 1801 of LNCS., Springer-Verlag (2000) 252–263
[7] Miller, J.F., Smith, S.L.: Redundancy and Computational Efficiency in Cartesian Genetic Programming. IEEE Transactions on Evolutionary Computation **10**(2) (2006) 167–174
[8] Miller, J.F., Job, D., Vassilev, V.K.: Principles in the Evolutionary Design of Digital Circuits – Part II. Genetic Programming and Evolvable Machines **1**(3) (2000) 259–288
[9] Miller, J., Thomson, P.: Cartesian Genetic Programming. In: Proc. of the 3rd European Conference on Genetic Programming EuroGP2000. Volume 1802 of LNCS., Springer (2000) 121–132
[10] Yu, T., Miller, J.F.: Neutrality and the evolvability of boolean function landscape. In: EuroGP '01: Proceedings of the 4th European Conference on Genetic Programming. Volume 2038 of LNCS., Springer-Verlag (2001) 204–217
[11] Collins, M.: Finding needles in haystacks is harder with neutrality. In: GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM (2005) 1613–1618
[12] Miller, J.: What bloat? cartesian genetic programming on boolean problems. In: 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers. (2001) 295–302
[13] Yang, S.: Logic Synthesis and Optimization Bechmarks User Guide, Version 3.0. (1991)
[14] Berkley Logic Synthesis and Verification Group: (ABC: A System for Sequential Synthesis and verification)
[15] Weste, N., Harris, D.: CMOS VLSI Design: A Circuits and Systems Perspective (3rd edition). Addison Wesley (2004)