Bent Function Synthesis by Means of Cartesian Genetic Programming

Radek Hrbacek and Vaclav Dvorak

Brno University of Technology,
Faculty of Information Technology
Bozetechova 2, 61266 Brno, Czech republic
ihrbacek@fit.vutbr.cz, dvorak@fit.vutbr.cz
http://www.fit.vutbr.cz/~ihrbacek/

Abstract. In this paper, a new approach to synthesize bent Boolean functions by means of Cartesian Genetic Programming (CGP) is proposed. Bent functions have important applications in cryptography due to their high nonlinearity. However, they are very rare and their discovery using conventional brute force methods is not efficient enough. We show that by using CGP we can routinely design bent functions of up to 16 variables. The evolutionary approach exploits parallelism in both the fitness calculation and the search algorithm.

1 Introduction

Evolutionary Algorithms (EAs) have been recently used in many engineering areas as design and optimization methods. Thanks to the innovation introduced into the design process, they are able to outperform conventional approaches in particular problems. Several types of EAs have been successfully employed in the task of evolutionary circuit design. Besides Genetic Programming (GP) heavily used by John Koza [1] to automatically design analog circuits, regulators, optical systems or antennas, excellent results have been achieved with the use of Cartesian Genetic Programming (CGP) [2]. The applications of CGP include combinational circuits design [3] and optimization [4], digital image filter design [5, 6], artificial neural networks design [7] and many others.

The evolutionary design is often very computationally demanding approach. In order to reduce the design time, various application specific accelerators as well as evolutionary algorithm modifications have been proposed. While the former case typically involves parallel fitness function implementation based on FPGA accelerators [6, 8] or running on multicore CPUs, GPUs [9] or even computer clusters [3] and exploiting parallelism at various levels (instruction, data, thread or process), the latter one includes genotype representation or search algorithm modifications. In the past, spatially structured evolutionary algorithms have been intensively studied and variety of approaches differing in the used evolutionary algorithm or communication topology has emerged [10–12].

While the use of computers and communication networks is becoming more and more popular, one has to seriously deal with the security of the data being stored or transferred. In cryptography, the two most fundamental techniques to achieve security in systems are *confusion* and *diffusion* [13]. Confusion refers to making a complex relationship between the ciphertext and the key. Thanks to diffusion, the statistical structure of the plain text is dissipated over significant part of the ciphertext, which prevents from reconstructing the original statistical information. In real cryptographic systems, the cipher key is much shorter than the message being encrypted and thus the key has to be reused in some way, often by applying a Boolean function to the key all over again. To avoid decryption by an attacker, the key sequence has to be random. If the Boolean function used to generate the key stream is close to linear, the message can be possibly deciphered. By using functions that are as far from linear as possible, one can build more secure cryptographic systems [14]. These functions, called *bent* functions, are very rare.

The state of the art methods for finding them operate usually on the brute force principle although exploiting some properties of the functions in order to reduce the size of the search space [15]. The number of Boolean functions grows exponentially with the number of variables, while the relative frequency of bent functions decreases. Therefore, for higher number of variables (the literature reports only functions of no more than 8 variables), these methods are not efficient enough. Another approach based on genetic algorithm (GA) is very limited as well. Even though the GA seems to be suitable for this purpose, the proposed approach is not scalable enough [16].

Inspired by the evolutionary design of combinational circuits by means of CGP, we propose a CGP based synthesis of bent Boolean functions. The parallelism at the data, thread and process level has been applied in order to take advantage of modern processor architectures and computer clusters. The scalability of this approach has been earlier verified in the task of evolutionary design of combinational adders and multipliers [3].

The paper is organized as follows. Section 2 introduces bent Boolean functions from the mathematical perspective. CGP is discussed in Section 3 and the proposed evolutionary approach to synthesize bent functions is described in Section 4. Section 5 is dedicated to experiments and the achieved results, final conclusions can be found in Section 6.

2 Bent Boolean functions

Boolean functions are of great importance for various cryptographic algorithms. Special attention is paid to the design of nonlinear Boolean functions due to their resistance to linear cryptanalysis [17]. This section presents necessary mathematical definitions for the purpose of introduction of bent functions [14, 15].

Definition 1. *A* ***Boolean function*** *is a function of the form* $f : \mathrm{D}^n \rightarrow \mathrm{D}$*, where* $\mathrm{D} = \{0, 1\}$ *is a Boolean domain and* $n \geq 0$ *is the arity of the function. For a function* f*, let* $f_0 = f(0, 0, \ldots, 0)$*,* $f_1 = f(0, 0, \ldots, 1)$*, ...,* $f_{2^n-1} = f(1, 1, \ldots, 1)$*.* $\mathrm{TT}_f = (f_{2^n-1} \cdots f_1 f_0)$ *is the* ***truth table representation*** *of the function* f*.*

Definition 2. *A* ***linear*** *(Boolean) function is either the constant 0 function or the exclusive OR (XOR) of one or more variables. An* ***affine*** *(Boolean) function is a linear function or the complement of a linear function.*

Definition 3. *The* ***Hamming distance*** $d(f, g)$ *between two functions* f *and* g *is the number of truth table entries with different values.*

Definition 4. *The* ***nonlinearity*** NL_f *of a function* f *is the minimum Hamming distance between the function* f *and an affine function.*

Definition 5. *Let* f *be a Boolean function of even arity* n, f *is a* ***bent function*** *iff its nonlinearity* NL_f *is maximum among* n*-variable functions.*

Affine functions are not suitable for the use in cryptography, since they are susceptible to a linear attack. Therefore, one seeks functions that are as far away (in the Hamming distance) as possible from all the affine functions – these are the bent functions. The nonlinearity of a bent function f of n variables is $\mathrm{NL}_f = 2^{n-1} - 2^{\frac{n}{2}-1}$ [18]. This constraint is not applicable for functions of odd

Table 1. Examples of 4-variable Boolean functions and their nonlinearities.

	function f	truth table TT_f	nonlinearity NL_f
linear	0	0000000000000000	0
	x_0	1010101010101010	0
	x_1	1100110011001100	0
	$x_1 \oplus x_0$	0110011001100110	0
	x_2	1111000011110000	0
	$x_2 \oplus x_0$	0101101001011010	0
	$x_2 \oplus x_1$	0011110000111100	0
	$x_2 \oplus x_1 \oplus x_0$	1001011010010110	0
	x_3	1111111100000000	0
	$x_3 \oplus x_0$	0101010110101010	0
	$x_3 \oplus x_1$	0011001111001100	0
	$x_3 \oplus x_1 \oplus x_0$	1001100101100110	0
	$x_3 \oplus x_2$	0000111111110000	0
	$x_3 \oplus x_2 \oplus x_0$	1010010101011010	0
	$x_3 \oplus x_2 \oplus x_1$	1100001100111100	0
	$x_3 \oplus x_2 \oplus x_1 \oplus x_0$	0110100110010110	0
nonlinear	$x_3 x_0$	1010101000000000	4
	$x_2 x_1 x_1 \oplus x_3 \oplus x_0$	1101010100101010	2
	$x_3 x_0 \oplus x_1$	0110011011001100	4
	$x_3 x_2 \oplus x_1 \oplus x_0$	0110011011001100	4
bent	$x_3 x_2 \oplus x_1 x_0$	0001000100011110	6
	$x_3 x_0 \oplus (x_2 \oplus x_0) x_1 \oplus x_2 \oplus x_0$	1011100000010010	6

arity that can, in general, have greater nonlinearity. This paper deals only with functions of even number of variables.

Examples of Boolean functions of 4 variables can be seen in Table 1. In the first 16 rows, all linear functions are listed, followed by several nonlinear and bent functions, the maximum nonlinearity of 4-variable functions is $\mathrm{NL}_f = 2^{4-1} - 2^{\frac{4}{2}-1} = 6$.

The number of different Boolean functions grows exponentially with the number of variables: $N_f(n) = 2^{2^n}$. However, the relative frequency of bent functions decreases very fast (see Table 2) and thus, for $n \geq 6$, identifying them is like looking for a needle in a haystack.

Table 2. Relative frequency of n-variable bent functions [14].

variables n	2	4	6	8
Boolean functions	2^4	2^{16}	2^{64}	2^{256}
bent functions	2^3	$\approx 2^{9.8}$	$\approx 2^{32.3}$	$\approx 2^{106.3}$
relative frequency	2^{-1}	$\approx 2^{-6.2}$	$\approx 2^{-31.7}$	$\approx 2^{-149.7}$

Recently, various approaches based on the properties of bent functions have been proposed, effectively reducing the number of the Boolean functions needed to be verified in order to identify bent functions by means of a brute force search [16, 15]. In some special cases, bent functions can be constructed directly [17].

3 Cartesian Genetic Programming

Cartesian genetic programming - a branch of genetic programming - has been introduced by Miller [2] and since then it has been successfully applied to a number of challenging real-world problems [19]. In contrast with GP which uses tree representation, an individual in CGP is represented by a directed acyclic graph. This dissimilarity enables the candidate solution to automatically reuse intermediate results and have multiple outputs, which makes CGP very suitable for design of various kinds of digital circuits, digital filters, etc.

A candidate program in CGP consist of the cartesian grid of $n_\mathrm{r} \times n_\mathrm{c}$ programmable nodes interconnected by a feed-forward network, as it can be seen in Figure 1. Node inputs can be connected either to one of n_i primary inputs or to a node in preceding l columns, each node has usually a fixed number of inputs $n_\mathrm{ni} = 2$. Each node can perform one of n_ni-input functions from the set Γ. Each of n_o primary circuit outputs is connected either to a primary input or a node output, the output connectivity can be additionally restricted by the o-back parameter. By changing the grid size and the l-back parameter, one can control the area and delay of the circuit.

Thanks to the fixed topology of CGP programs, each chromosome can be encoded using an fixed-sized array of $n_\mathrm{r} \cdot n_\mathrm{c} \cdot (n_\mathrm{ni} + 1) + n_\mathrm{o}$ integers (n_ni inputs

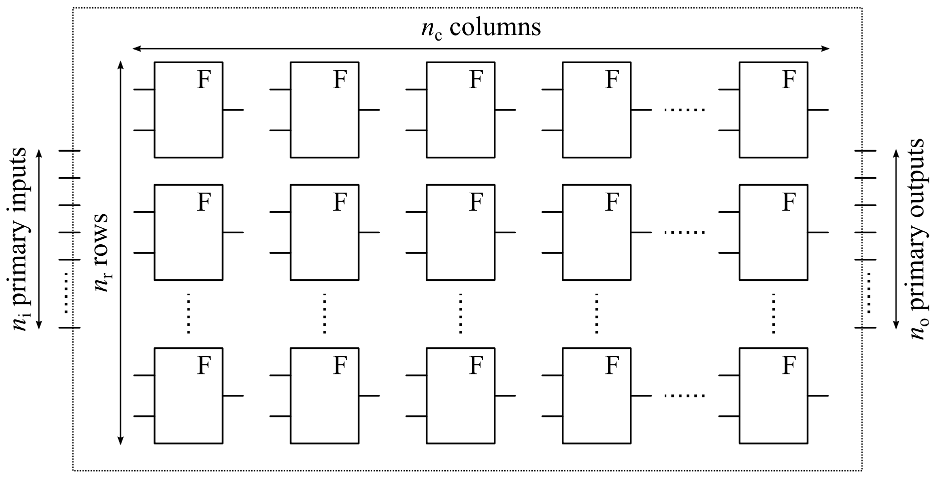

Fig. 1. Cartesian genetic programming scheme.

and one function per each node). Each primary input is assigned a number from $\{0, ..., n_i - 1\}$ and the nodes are assigned numbers from $\{n_i, ..., n_i + n_r \cdot n_c - 1\}$. Unlike the genotype, the phenotype is of variable length depending on the number of inactive nodes (i.e. nodes whose output is not used by any other node or primary output), which implies the existence of individuals with different genotypes but the same phenotypes. The existence of individuals with different genotypes but with the same fitness value is usually referred to as neutrality. For certain problems, the neutrality significantly reduces the computational effort and helps to find more innovative solutions [20].

CGP uses a simple mutation based $(1 + \lambda)$ evolutionary strategy as a search mechanism, the population size $1+\lambda$ is mostly very small, typically, λ is between 1 and 15. The initial population is constructed randomly in most cases, however, it can be seeded with a known solution as well (evolutionary optimization) [4]. In each generation, the best individual or a sibling with the same fitness value is passed to the next generation unmodified along with its λ offspring individuals created by means of point mutation operator. The mutation rate m is usually set to modify up to 5 % randomly selected genes. Usually, no crossover operator is used in CGP, however, for particular problems (e.g. symbolic regression), special crossover operators have been investigated [21]. None of them has been confirmed as useful for other problem classes so far.

In the case of combinational circuit design, the fitness function is given by the number of correct output bits compared to a specified truth table. All combinations of input values (2^{n_i} test vectors for a circuit with n_i inputs and n_o outputs) have to be fetched to the primary inputs in order to obtain a fully working circuit. $n_o \cdot 2^{n_i}$ output bits have to be verified so as to compute the fitness value.

4 Bent function synthesis by means of CGP

The principle of bent function synthesis by means of CGP is very similar to the case of combinational circuit design, since every Boolean function can be implemented by a combinational circuit. The difference lies in the fitness function. Unlike combinational circuits having fitness value equal to the total number of wrong output bits, the fitness value of a bent function candidate is its nonlinearity, i.e. the lowest Hamming distance from a linear function. Despite the fact, that bent Boolean functions have single output comparing to combinational circuits having arbitrary many outputs, the fitness calculation is computationally more intensive, since the number of linear functions being compared with the candidate individual grows exponentially with the number of variables.

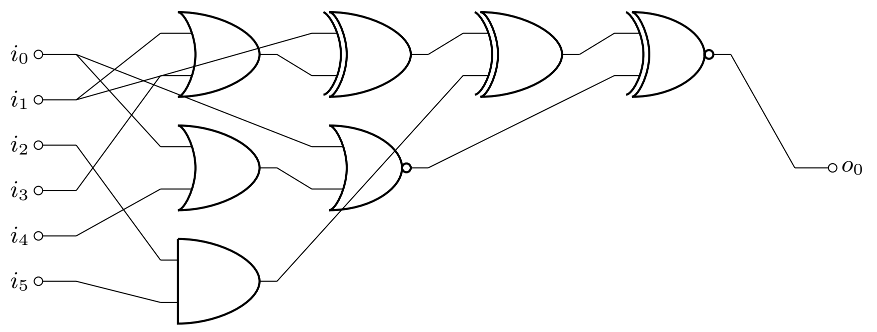

Fig. 2. Example of an CGP individual representing the Boolean function $f(i_5, \ldots, i_0) = o_0 = \overline{((i_1 \oplus (i_1 + i_3)) \oplus i_2 i_5) \oplus \overline{i_0 + (i_0 + i_4)}}$ with the truth table $\mathrm{TT}_f =$ 0011110001101001001100110110011011110000101001011111111110101010. This function has nonlinearity $\mathrm{NL}_f = 28$ and thus it is bent.

Figure 2 depicts an example of an CGP individual representing a Boolean function. Note that the representation is not optimal in terms of area or delay, since the only significant property is the truth table.

While evaluating an individual's fitness value, all active genes of the chromosome need to be traversed and their output values need to be calculated. The single output is then compared against all linear functions simply by XORing the values and counting the number of ones. There is no need to compare the values to the remaining affine functions (the complements of linear functions), since the following always holds true:

$$d(f, g) + d(f, g_c) = 2^n, \tag{1}$$

where f, g are arbitrary n-variable Boolean functions and g_c is complementary to g.

The entire evolutionary design process can be accelerated in the same way as it has been done in the case of combinational circuits [3]. The test vectors can be fed to the CGP individual in parallel, from 64 test vectors within a standard x86-64 register up to 256 test vectors using AVX extension. Moreover, the population can be split over a number of threads, each thread handling a portion of the population. Nevertheless, the number of threads is substantially limited by the population size, which is usually very small in CGP. In order to take advantage of multicore processors or even computer clusters, additional level of parallelism has to be exploited. By introducing spatially structured EA principle, one can scale the evolutionary process onto arbitrary sized computer cluster. Unfortunately, the absence of crossover operator in CGP is a very limiting factor, since most parallel algorithms are based on combining genotypes from different spatially isolated populations. Thus, simple isolated islands model with periodical exchange of the best individual is used [3].

5 Experimental results

In this section, experiments regarding the ability of the proposed approach to synthesize bent functions are presented. All the experiments were performed on a computer cluster of 112 nodes with the following hardware configuration: 2× 8-core Intel E5-2670, 128 GB RAM, 2× 600 GB 15 k scratch hard disks, connected by gigabit Ethernet and Infiniband links.

The performance of the CGP based approach has been examined in terms of the evolution time. The CGP parameters were set as follows on the basis of previous experiments with combinational circuits [3]: the functions set $\Gamma = \{\texttt{BUF}, \texttt{NOT}, \texttt{AND}, \texttt{OR}, \texttt{XOR}, \texttt{NAND}, \texttt{NOR}, \texttt{XNOR}\}$, population of 5 individuals, mutation

Table 3. Bent Boolean functions CGP based synthesis times.

n	nodes	hosts/	time [s]		
	$n_r \times n_c$	threads	mean	median	std
6	1×50	1/1	0.000819	0.000685	0.000668
8	1×100	1/1	0.00470	0.00343	0.00410
10	1×150	1/1	0.0602	0.0442	0.0483
12	1×200	1/1	2.0443	1.4057	1.9579
		1/4	1.1291	0.8392	1.0610
		4/1	0.8240	0.6267	0.5405
		40/1	0.3859	0.3618	0.1080
14	1×250	1/1	133.202	91.765	146.839
		1/4	76.040	54.954	72.808
		4/1	44.680	35.700	34.165
		40/1	15.806	15.255	4.853
16	1×300	1/1	6223.66	4666.82	4734.02
		1/4	3880.06	3744.23	2571.49
		4/1	1855.79	1543.12	1329.10
		40/1	636.13	565.68	229.06

rate 5 %. The number of rows was set to $n_\mathrm{r} = 1$ and the l-back parameter was maximal, enabling the greatest connectibility (there were no requirements on the propagation delay). The size of the grid was empirically chosen for each variable count n as a optimal choice with respect to the evolution time (about $10\times$ larger than the average individual found). No limitations on the number of generations were imposed, each run was successful. The spatially structured implementations exchanged the best individual over all populations every 100 generations.

The achieved results can be seen in Table 3, four different configurations of the algorithm were compared – basic single threaded variant, accelerated 4-thread parallel version, 4-island and 40-island spatially structured variants. For each configuration, 100 independent runs were performed and common statistical metrics were calculated – the mean time, the median value and the standard deviation. The evolution times for functions of less than 12 variables are negligible and cannot be improved by means of thread or process level parallelism, because there is not enough work to distribute. For higher numbers of variables, the computational effort grows rapidly and the parallel implementations help significantly to reduce the evolution time. For example, the design of 16-variable bent functions can be sped up $10\times$ on the computer cluster in comparison with the basic single threaded implementation. It shows that even a small number of isolated populations can more efficiently utilize the power of a multicore processor than the multithreaded single population approach. Not only the mean and median times, but also the standard deviations of the evolution times are lower, increasing the probability of finding a bent function in a limited time.

An example of a bent Boolean function of 16 variables synthesized by means of CGP can be seen in Figure 3. Its nonlinearity is 32,640 and the CGP representation has 24 active nodes with the maximum delay of 7.

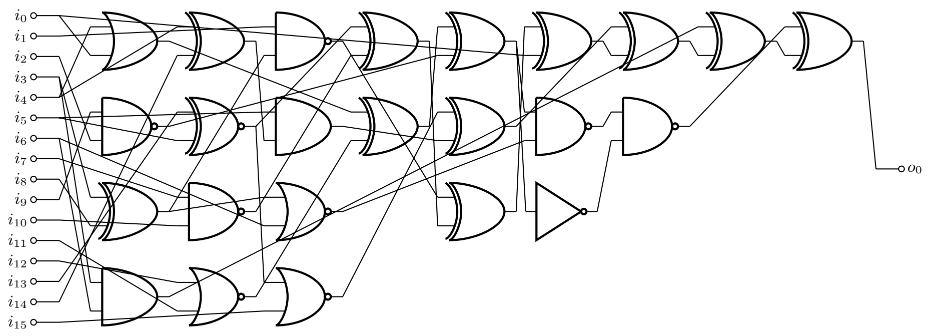

Fig. 3. CGP representation of a 16-variable bent Boolean function.

6 Conclusions

In this paper, a new approach to synthesize bent Boolean functions based on CGP has been proposed. Bent functions have applications in cryptography due to their significant properties – when used in a cipher, their nonlinearity makes cryptanalysis harder. The relative frequency of bent functions among all Boolean functions of the same arity is rapidly decreasing with the number of variables and designing such functions is harder and harder.

It was shown, that by using CGP, we are able to routinely design bent Boolean functions of up to 16 variables. The evolutionary process was sped up by employing various levels of parallelism in both fitness calculation and the search algorithm, which gives a great scalability to the proposed approach. Several algorithm configurations were experimentally compared and it was shown, that by using a simple isolated island model, one can significantly reduce the evolution time. Additional effort has to be made in order to investigate potential common features shared by bent functions found using independent CGP runs.

Even though bent Boolean functions themselves have great properties, in order to achieve maximum confusion in real cryptographic systems, there should be a balance between bits that are changed and that are not. This can be achieved by using *balanced* functions; however, no bent function is balanced and thus a trade-off between nonlinearity and balance has to be sought [17, 14]. In our future research, we want to focus on designing such functions by means of evolutionary algorithms. Further work will be also devoted to the optimization of the synthesized functions in terms of area and delay inspired by fast SAT-based optimization methods for complex combinational circuits [4].

7 Acknowledgements

This work was supported by the Czech Science Foundation project 14-04197S. The access to the CERIT-SC computing and storage facilities provided under the programme Center CERIT Scientific Cloud, part of the Operational Program Research and Development for Innovations, reg. no. CZ. 1.05/3.2.00/08.0144 is appreciated.

References

1. Koza, J.R.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers, Norwell, MA, USA (2003)
2. Miller, J., Thomson, P.: Cartesian genetic programming. In: Genetic Programming. Volume 1802 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2000) 121–132
3. Hrbacek, R., Sekanina, L.: Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation. In: Proceeding of Genetic and Evolutionary Computation Conference, GECCO 2014, Association for Computing Machinery (2014) (to appear).

4. Vasicek, Z., Sekanina, L.: On area minimization of complex combinational circuits using cartesian genetic programming. In: 2012 IEEE World Congress on Computational Intelligence, Institute of Electrical and Electronics Engineers (2012) 2379–2386
5. Vasicek, Z., Bidlo, M.: Evolutionary design of robust noise-specific image filters. In: 2011 IEEE Congress on Evolutionary Computation, IEEE Computer Society (2011) 269–276
6. Hrbacek, R., Sikulova, M.: Coevolutionary cartesian genetic programming in fpga. In: Advances in Artificial Life, ECAL 2013, Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems, MIT Press (2013) 431–438
7. Khan, G., Miller, J.: The cgp developmental network. In Miller, J.F., ed.: Cartesian Genetic Programming. Natural Computing Series. Springer Berlin Heidelberg (2011) 255–291
8. Vasicek, Z., Sekanina, L.: Hardware accelerator of cartesian genetic programming with multiple fitness units. Computing and Informatics **29**(6) (2010) 1359–1371
9. Harding, S., Banzhaf, W.: Hardware acceleration for cgp: Graphics processing units. In Miller, J.F., ed.: Cartesian Genetic Programming. Natural Computing Series. Springer Berlin Heidelberg (2011) 231–253
10. Cantu-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA, USA (2000)
11. Tomassini, M.: Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
12. Jaros, J.: Multi-gpu island-based genetic algorithm solving the knapsack problem. In: 2012 IEEE World Congress on Computational Intelligence, Institute of Electrical and Electronics Engineers (2012) 217–224
13. Shannon, C.: Communication theory of secrecy systems. Bell System Technical Journal, Vol 28, pp. 656715 (Oktober 1949)
14. Butler, J.T., Sasao, T.: Logic functions for cryptography - a tutorial. In: Proceedings of the Reed-Muller Workshop. (2009)
15. Shafer, J.L., Schneider, S.W., Butler, J.T., Stanica, P.: Enumeration of bent boolean functions by reconfigurable computer. In Sass, R., Tessier, R., eds.: FCCM, IEEE Computer Society (2010) 265–272
16. Schneider, S.W.: Finding bent functions using genetic algorithms. Master's thesis, Naval Postgraduate School, Monterey (2009)
17. Dobbertin, H.: Construction of bent functions and balanced boolean functions with high nonlinearity. In Preneel, B., ed.: Fast Software Encryption. Volume 1008 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1995) 61–74
18. Rothaus, O.: On "bent" functions. Journal of Combinatorial Theory, Series A **20**(3) (1976) 300 – 305
19. Miller, J.F., ed.: Cartesian Genetic Programming. Natural Computing Series. Springer Verlag (2011)
20. Miller, J., Smith, S.: Redundancy and computational efficiency in cartesian genetic programming. Evolutionary Computation, IEEE Transactions on **10**(2) (2006) 167–174
21. Clegg, J., Walker, J.A., Miller, J.F.: A new crossover technique for cartesian genetic programming. In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. Volume 2., London, ACM Press (7-11 July 2007) 1580–1587