Cartesian Genetic Programming Using Bit Representation

Petr Burian

***Abstract* — This paper deals with the evolutionary design of digital circuits. Concretely, it explores particular aspects of Cartesian Genetic Programming (CGP). The paper deals with the representation of a chromosome, especially the usage of a bit representation. An evolutionary algorithm using only bit operations is applied to CGP. Two circuits – 3 bit x 2 bit multiplier and 5 bit majority circuit – are used for algorithm power testing.**

***Keywords* — Cartesian Genetic Programming, Evolutionary Circuit Design.**

I. Introduction

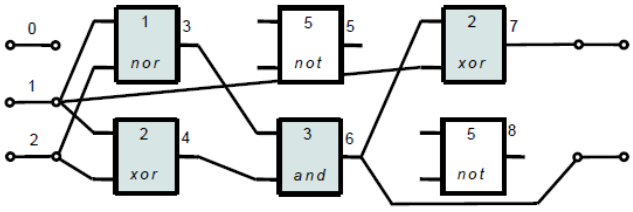
The use of the evolutionary computational techniques is a relatively new and modern method of designing circuits – digital or analogue. This branch is based on reconfigurable structures and on search algorithms. Most often, some kind of evolutionary algorithm is exploited to search for a suitable configuration of a reconfigurable structure. The interconnection between evolutionary algorithms and reconfigurable structures can be used in two similar domains - Evolvable Circuits (Hardware) and Evolutionary Circuit Design.

The evolutionary design deals just with the design of circuit by means of evolutionary techniques. The evolution is used only for development. This method of circuit design makes use of the mathematical model either of the reconfigurable structure or of the system with variable parameters. This model is used for the fitness function. A new and innovative solution to the circuits can be found by these techniques. The resulting circuit can be implemented as real hardware.

However, in case of evolvable hardware, both parts – the reconfigurable structure and the evolutionary algorithm – are really implemented. The evolution can run whenever. It means that the circuit can change its function dynamically in time. By this approach, we can get a circuit which performs various functions. To change its function, we need to define the required new function at first and then the evolution can be started. If the evolution finds a suitable configuration, the system will perform the required function.

II. Cartesian Genetic Programming

There are many methods of designing evolvable system or system using evolutionary design. The choice of a suitable method is mainly a question for developers. Every system is designed "ad hoc" for the given application. The designer has to propose a suitable representation of reconfigurable structure, the calculation of appropriate fitness function, search algorithm, etc. However, many evolutionary systems are based on Cartesian Genetic Programming (CGP). [1]



1,2,1; 1,2,2; 4,2,5; 3,4,3; 6,1,2; 0,5,5; 7,6

Fig. 1. Example of CGP [2]

In CGP, a reconfigurable structure is modelled as set of reconfigurable function cells in matrix organization (see Fig. 1). The size of this structure is defined as n_c (columns) x n_r (rows). Each cell has n_n inputs and one output; it can implement one function out of the group of defined functions. Other parameters of CGP are the number of primary inputs (n_i) and outputs (n_o). Configuration information of the structure determines the function of particular cells, interconnection among them, interconnection between primary inputs and cells, and interconnection between primary output and cells. For the interconnection in structure, there are several rules. Feedback is not allowed; the implementation of it results in difficult valuation of the structure. The primary input can be connected as input to all cells. Connectivity among cells is limited by the so-called level back parameters (l-back parameter), which determines columns, whose outputs can be connected to the current cell. For example, if l-back = 1, only cell outputs of the immediately previous column may be used as input for the current cell. If l-back parameter is set to maximum (n_c), there are no limits for the connectivity among cells. A higher value of the l-back parameter enables a greater possibility of connectivity; thereby we may generate more complicated circuit. However, we must take into account that high value of l-back increases the space where the algorithm searches for suitable solutions. The CGP uses integer representation (one gen of chromosome is represented by one integer), each cell is defined by n_n+1 integer values; first n_n values determine input nodes of the cell, and the last integer determines the function of the cell. The end of a chromosome is constructed by integers which define the

Petr Burian is with the University of West Bohemia in Pilsen, Department of Applied Electronics and Telecommunications, Univerzitni 26, 30614 Pilsen, Czech Republic
e-mail: burianp@kae.zcu.cz

output cells. The total length of the chromosome is defined as [1]:

$$\Lambda = n_r n_c (n_n + 1) + n_o$$

The algorithm which is used for the search for solutions is very similar to the Evolutionary Strategy – (1 + λ)-ES, where λ roughly equals 4. It is necessary to ensure it so that chromosome genes only take correct values. In other part we may use a classical version of the algorithm. At first, the initial population is randomly generated. The process of mutation changes a few genes randomly. After the calculation of the fitness function, a selection is performed. It selects the best individual; if there are more than one individual with the best value of fitness, the algorithm prefers the products of mutation to its parents. This step ensures population diversity.

The key element of every evolutional design is the fitness (valuation) function. This function tells us how good the quality of the solution produced by the individual is. The fitness expresses the number of the lines in the truth table, where the output matches the required output. For example, assume a structure with 4 inputs and 2 outputs. The maximum value of the fitness function is 32 (2 x 2^4). This value expresses that the algorithm has found a circuit which is fully functional. However, the CGP may optimise the circuit otherwise. If we define fitness function as:

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

where b represents circuit functionality (according to the required truth table), and z denotes the number of active (used) cells, we optimise the circuit to increase the number of used cells (gates). The algorithm takes into account the number of used cells only in cases when full functionality is found. It is very important. [2]

III. Integer and Bit Representation

As mentioned above, common representation (chromosome) of the Cartesian Genetic Programming (CGP) uses a group of integers. It means that appropriate algorithm performing the search of a suitable configuration has to be capable to work with integers. In the case of evolution design which is executed for example on PC, this one doesn't present a significant issue. Another situation arises if we implement algorithm by logic, it means perhaps by an FPGA device. It would be more suitable to approach the configuration information (chromosome) as a bit stream, because bit operations are less difficult than operations with integers. This part of the paper deals with comparing the efficiency of the CGP while using bit and integer representations.

The use of representation yields considerable benefits. Integer directly determines the used function of function cells or interconnections among them. On the other hand, we have to keep information on particular parts of chromosome, because every integer of chromosome can gain limited values. Another disadvantage is the process of mutation – the generation of a new part of chromosome is surely more difficult than mutation in bit representation – it means bit negation.

Assume the following instance: structure 1 x 16 (rows x columns); all cells can be connected to primary inputs; l-back parameter = maximum; each cell can perform one of these 2-input functions – AND, OR, XOR, interconnection. Further, 5 primary inputs are taken into account. In case of integer representation, 3 integers are needed for each cell. Only acceptable values determining interconnection are variable. Note that particular integer represents certain nodes directly. For example, the first cell can be connected only to primary inputs so that the interconnection integer takes values between 0 and 4. For the last cell, the interconnection integers are between 0 and 19. The integer determining the output cell is in the range between 5 (the first cell) and 20 (the last cell). In bit representation, we obviously want the chromosome to be as short as possible. For that reason, each cell is defined by a different number of bits. In our instance, the first cell needs 2 x 3 bits for interconnection (5 primary inputs) and 2 bits to determine the function; it means 8 bits as a whole. Indeed, the last cell needs 2 x 5 bits for interconnection, as a whole 12 bits. The output cell is defined by 4 bits (16 cells). If we approach the chromosome only as a bit stream, it is obvious that it will contain incorrect values. For example, the last cell can be connected only to 20 nodes (5 primary inputs and 15 previous cells), however, 5 bits code to up to 32 nodes. If we want to use the search algorithm working with a bit chromosome, it is necessary to ensure the correct calculation of a fitness function even with an incorrect chromosome. The easiest way how to solve it is the assignment of a maximum value for a certain group of bits. If the group of bits represents a higher than allowable value, the fitness function calculation uses this maximum value. It is necessary to note that we discuss only change in the implementation of a fitness function; the algorithm bit approach to the chromosome stays unchanged. It is clear that these steps increase the probability of neutral mutations.

These two approaches, bit and integer, were tested on two examples – the evolutionary design of a 3b x 2b multiplier and a 5 bit majority circuit. These ones are relatively the favourite for testing the effectiveness of the CGP. [2], [3] For the evolutionary design of a multiplier the cell array 1 x 16 was used and for a majority circuit the 5 x 5 array was used. The l-back parameter is set to its maximum value. For the instances, the algorithm (1 + λ) was used, where λ = 4. For the integer representation the common version of an algorithm is used. For a bit representation, an algorithm whose mutation part and initialization part work with a bit stream without extra information on the meaning of particular bits, their limits, etc., is used. The mutation rate expresses the number of mutation (generating new integer value or bit negation) within every mutation cycle of the algorithm. The value of a fitness function describes the correspondence between the required and the achieved truth tables. In the case of a 3b x 2b multiplier, it means a circuit with 5-bit inputs and 5-bit outputs, the maximum fitness function is 160 (2^5 x 5); in the case of a majority circuit (5 inputs; 1 output), the fitness maximum is 32.

After reaching this value, the algorithm starts to reduce the number of cells used in the circuit so that the number of unused cells is added to the fitness value. Thus, the aim of the evolution is a circuit with full functionality and with as few active (used) cells (gates in this case) as possible. For the data relevancy, each run of evolutionary design was repeated 20 times; the run of the evolution is terminated if the fitness achieves the value of 163 (multiplier; full functionality and 13 used cells), respectively 45 (majority circuit; full functionality and 12 used cells) or if the number of generations exceeds the value of 2,000,000. The results for different mutation rates are shown in the following tables.

TABLE 1: RESULTS OF EVOLUTIONARY DESIGN (INTEGER REPRESENTATION)

Circuit	*Mut. rate*	*Mean # generations*	*Full funct.*	*Mean # gates*
Multiplier 3b x 2b	1	208,666	100%	13.00
Multiplier 3b x 2b	2	264,281	100%	13.00
Multiplier 3b x 2b	3	261,808	100%	13.00
Multiplier 3b x 2b	4	733,898	95%	14.50
Majority 5 bit	1	67,819	100%	11.70
Majority 5 bit	2	49,423	100%	11.80
Majority 5 bit	3	54,170	100%	11.75
Majority 5 bit	4	43,020	100%	11.40
Majority 5 bit	5	75,337	100%	11.70

Table 1 summarises the results of algorithm with an integer representation. It shows the average number of generations needed for termination (finding required solution or achieving the allowable number of generations) of the evolutionary run. It also shows the average number of gates (only fully functional circuits are taken into account) which are used in the resulting circuit. The "Full funct." item expresses the percentage of evolutionary runs in which the fully functional circuit was found. The design of a multiplier is a more difficult task than the design of a majority circuit; the algorithm needs more generations to find it. The results tell us that almost all algorithm runs found the required solution; similar results were obtained also by other authors [2].

Table 2 is equivalent to the previous table. However, it presents results of an algorithm which uses purely bit operations (initialization and mutation) and its fitness function respects incorrect chromosomes. If the incorrect part of the chromosome rises, the calculation uses the maximum allowable value (saturation) of this chromosome part – however, the chromosome stays unchanged. From the Table 2 results, it can be explicitly read that this type of algorithm is less effective than the algorithm with an integer chromosome representation. In both cases – multiplier and majority circuit – the algorithm needs roughly 3 times more generations than the previous integer representation. Bit representation makes the search possibilities of the algorithm worse. Also the optimal value of a mutation rate is higher than in the case of an integer representation. This value is equal to 3 (multiplier) and 6 (majority circuit), in comparison with 1, respectively 4, in an integer representation. In the case of a majority circuit, not all runs of the evolution provide a fully functional circuit for low values of a mutation rate. As shown in Table 1 and Table 2, from the viewpoint of the used cells (gates), the results of both versions of CGP are similar.

TABLE 2: RESULTS OF EVOLUTIONARY DESIGN (BIT REPRESENTATION; SATURATION OF INCORRECT VALUES)

Circuit	*Mut. rate*	*Mean # generations*	*Full funct.*	*Mean # gates*
Multiplier 3b x 2b	1	1,394,053	100%	13.70
Multiplier 3b x 2b	2	786,083	100%	13.25
Multiplier 3b x 2b	3	622,311	100%	13.15
Multiplier 3b x 2b	4	977,957	100%	13.20
Multiplier 3b x 2b	5	1,532,338	55%	13.00
Majority 5 bit	1	653,650	80%	11.94
Majority 5 bit	2	422,672	95%	11.79
Majority 5 bit	3	208,896	100%	11.95
Majority 5 bit	4	169,178	100%	11.95
Majority 5 bit	5	120,153	100%	11.70
Majority 5 bit	6	116,610	100%	11.85
Majority 5 bit	7	218,577	100%	11.55

Nevertheless, also this modification (with bit representation) of the CGP makes it possible to solve the design of a logic circuit. Remind that the main advantage of this is the possibility to exploit simple evolutionary algorithms with bit operations; the algorithms don't need extra information on the meaning of particular parts of a chromosome. However, the possibility of an effective chromosome mutation is limited; for that reason another version of representation was implemented and tested.

TABLE 3: "MIRROR ON MAXIMAL VALUE" MODIFICATION

Binary digit	*Decimal value*	*Modified value*
00000	0	0
00001	1	1
…	…	…
…	…	…
11000	24	24
11001	25	23
11010	26	22
…	…	…
…	…	…
11110	30	18
11111	31	17

Assume the instance of a majority circuit with a 5x5 cell array as mentioned above. The last part (integer or group of bits) of a chromosome determines which signal (output

of cell) in the array is regarded as the output of the whole proposed circuit. The total number of cells is 25 (5 x 5), hence the last part of the chromosome has to allow to code 25 (0 to 24) states. In the case of a bit representation, 5 bits are needed. However, 5 bits can code 32 states; therefore this part of the chromosome may acquire higher value than 24. If this case arises, the fitness function modifies the value in compliance with the "mirror on maximal value". The principle of this modification is described in Table 3. Eventually, we may express this modification by means of the following formula:

$$modified\ value = \begin{cases} d & when\ d < (max + 1), \\ 2\,max - d & otherwise, \end{cases}$$

where d is the original value of the chromosome part determining the output cell, and max denotes the maximum value acceptable value pointing output cell. This modification should improve the mutation process, thereby also the whole process of an evolutionary design. In the first place, we ensure the decrease in the probability of neutral mutations. Evidently, it doesn't mean that the mutation of the chromosome part determining the output cell will cause a real change in the output cell every time. However, even an integer mutation doesn't ensure this. In comparison with the previous modification of incorrect values, by this modification we avoid situations in which many binary digits represent only one (determined by max. allowable value) output cell. The modification was implemented and tested on the same benchmark. The results are shown in Table 4.

TABLE 4: RESULTS OF EVOLUTIONARY DESIGN (BIT REPRESENTATION; "MIRROR ON MAXIMAL VALUE" MODIFICATION)

Circuit	*Mut. rate*	*Mean # generations*	*Full funct.*	*Mean # gates*
Multiplier 3b x 2b	1	845,131	100%	13.35
Multiplier 3b x 2b	2	414,883	100%	13.15
Multiplier 3b x 2b	3	556,917	100%	13.20
Multiplier 3b x 2b	4	865,964	90%	13.06
Multiplier 3b x 2b	5	1,091,678	80%	13.19
Majority 5 bit	1	146,600	100%	11.75
Majority 5 bit	2	62,653	100%	11.70
Majority 5 bit	3	45,009	100%	11.55
Majority 5 bit	4	51,120	100%	11.90
Majority 5 bit	5	162,551	100%	12.00

The "mirror on maximal value" modification brought expected improved behavior of an evolutional process. As well as in the previous case, some runs don't find required solution but the number of generations needed for successful evolution is generally lower. The design of a multiplier needs c. 400,000 generations, which presents an improvement of approximately 33%. The situation is yet more positive in the case of a majority circuit; we get values comparable to an integer representation. Note that both the circuits used as a benchmark belong to less difficult tasks. Indeed, we may expect similar behavior within the design of a more complex circuit.

Exhibits of the final circuits are shown in Fig.2 and Fig.3. The function labels: 0 – XOR, 1 – OR, 2 – AND.

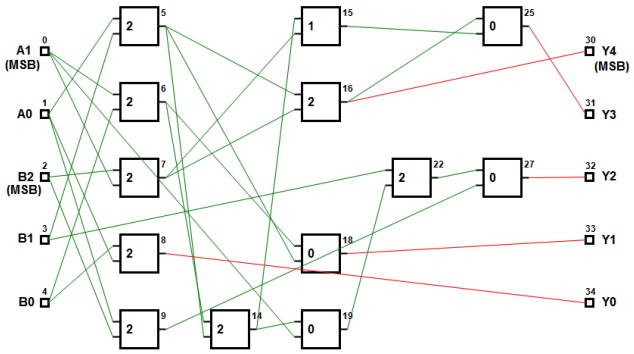


Fig. 2: Designed 3 bit x 2 bit multiplier circuit

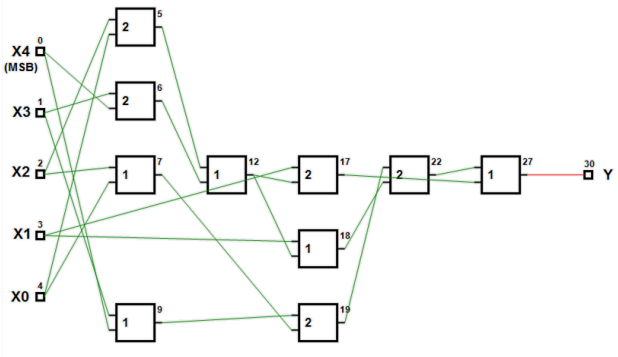


Fig. 3: Designed 5-bit majority circuit

IV. CONCLUSION

The use of the algorithm using only a bit operation was tested. The algorithm works with a bit chromosome without information on the meaning of particular bits. The author presented two basic methods of how to deal with the problem of an incorrect value in the chromosome – the replacement of the incorrect value by a maximal allowable value and the "mirror on maximal value" modification. The latter method provides relatively good results in comparison to an integer representation. Two tasks were exploited as benchmarks – the evolutionary design of a 3 bit x 2 bit multiplier and a 5 bit majority circuit. Both circuits were successfully designed and further the optimization of the number of needed gates was performed. The best solution of a 3 bit x 2 bit multiplier needs 13 gates (4 x XOR, 1 x OR, 8 x AND); the 5-bit majority circuit needs 10 gates (gates; 5 x OR, 5 x AND).

REFERENCES

[1] Sekanina L.: Evolvable components. Springer, Natural Computing series, 2004. ISBN 3540403779.

[2] Gajda Z., Sekanina L.: An Efficient Selection Strategy for Digital Circuit Evolution, In: *Evolvable Systems: From Biology to Hardware*, Berlin, DE, Springer, 2010, pp. 13-24, ISBN 978-3-642-15322-8.

[3] 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), 8-35 (2000)