Cluster-based Evolutionary Design of Digital Circuits using Improved Multi-Expression Programming

Fatima Zohra Hadjam
Dept. Comp. Science,
University of Djillali Liabes
BP 89, 22000 Sidi Bel Abbes Algeria
00213 48577750
fatima.hadjam@uni-dortmund.de

Claudio Moraga
European Centre for Soft Computing
33600 Mieres, Spain
University of Dortmund, Germany
0034 985456545
mail@claudio-moraga.eu

Mohamed Benmohamed
Dept. Comp. Science,
University of Mentouri
BP 325, Constantine 25017. Algeria
00213 31614348
ibnm@yahoo.fr

ABSTRACT
Evolutionary Electronics (EE) is a research area which involves application of Evolutionary Computation in the domain of electronics. EE algorithms are generally able to find good solutions to rather small problems in a reasonable amount of time, but the need for solving more and more complex problems increases the time required to find adequate solutions. This is due to the large number of individuals to be evaluated and to the large number of generations required until the convergence process leads to the solution. As a consequence, there have been multiple efforts to make EE faster, and one of the most promising choices is to use distributed implementations. In this paper, we propose a cluster-based evolutionary design of digital circuits using a distributed improved multi expression programming method (DIMEP). DIMEP keeps, in parallel, several sub-populations that are processed by Impoved Multi-Expression Programming algorithms, with each one being independent from the others. A migration mechanism produces a chromosome exchange between the subpopulations using MPI (Message Passing Interface) on a dedicated cluster of workstations (Lido Cluster, Dortmund University). This paper presents the main ideas and shows preliminary experimental results.

Categories and Subject Descriptors
B.6.1 [**Design Styles**]: Hardware, Logic Design – Combinational logic.

General Terms
Algorithms, Design, Experimentation, Performance.

Keywords
Genetic Programming, Improved Multi-Expression Programming, Islands Model, Combinational Circuits, Computational Effort.

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

1. INTRODUCTION
Since the introduction of standard GP by Koza (1992) in the early 90's, numerous GP paradigms have been proposed by the community, which include linear structured GP (linear GP) (Banzhaf et al., 1998), graph-based GP (Poli, 1999; Teller and Veloso, 1996), stack-based GP (Perkis, 1994; Stoffel and Spector, 1996), Cartesian GP (Miller and Thomson, 2000), concurrent GP (Trenaman, 1999), grammar-based GP (Wong and Leung, 2000), etc.

There are two main streams in GP, standard GP [14] and Linear structure GP (LGP) [2]. In standard GP, a genetic program is represented in a tree structure, similar to the S-expression structure used in LISP. Running a program tree on a classical von Neumann machine involves stack operations, recursive function calls, and program tree interpretation. In LGP, a genetic program is represented in a linear list of machine code instructions or high-level language statements.

Several linear variants of LGP have recently been proposed. Some of them are: multi-expression programming (MEP) [18], grammatical evolution (GE) [23], gene expression programming (GEP) [10], cartesian genetic programming (CGP) [15], genetic algorithm for deriving software (GADS) [22] and infix form genetic programming (IFGP) [19]. In [20] the authors provide a comparison of several Linear Genetic Programming Techniques: MEP, GEP, GE, and LGP. Several numerical experiments using five benchmarking problems are carried out. The results reveal that multi-expression programming has the best overall behavior for some well-known problems such as symbolic regression and even-parity.

The idea of using the Improved Multi Expression Programming (IMEP) in designing Digital Circuit was proposed in a previous study [11]. Despite the success of this technique, the obvious drawback is the substantial amount of processing time. However, the IMEP process can be easily implemented as a distributed algorithm since the fitness evaluation of a population of candidate solutions can be performed independently. The population-based nature of IMEP (GP) makes it easy to be distributed with a suitable architecture, because the genetic programs in a population are independent. Multiple genetic programs can be evolved in parallel in multiple engines (processors). These observations give rise to the most commonly used approach to "parallelization" –(actually, distribution)– of genetic programming, namely the asynchronous islands.

To this end, different approaches to distribute genetic programming have been studied. Extensive surveys on the subject can be found in [1], [3],[5], [6], [7], [8], [9], [13] and [24]. It should be mentioned that the use of "parallel" in the EA community is not sharp as in the Computer architecture community, where parallelism leads to vector machines like the Cray computer.

2. PROBLEM STATEMENT

The problem that we are trying to solve may be briefly stated as follows: Designing Digital Circuits using Distributed IMEP (Improved Multi-Expression Programming) under the structural model of "Islands". Multi-Expression Programming (MEP) [20], [21] is a Genetic Programming (GP) variant that uses linear chromosomes for solution encoding. A unique MEP feature is its ability of encoding multiple solutions of a problem in a single chromosome. These solutions are handled in the same time complexity as other techniques that encode a single solution in a chromosome.

IMEP is a modified MEP which outperforms MEP in different Benchmarks problems studied in [11]. The main introduced changes are on the way the individuals are encoded and also in the mutation process. To parallelize IMEP, a coarse-grained model called also Islands model was adopted. It keeps, in parallel, several subpopulations that are processed by multi-expression programming algorithms, with each one being independent from the others. A migration mechanism produces a chromosome exchange between the subpopulations using MPI (Message Passing Interface) on a dedicated cluster of workstations (The Lido Cluster at the University of Dortmund, Germany).

3. IMPROVED MULTI -EXPRESSION PROGRAMMING (MEP)

In this section, we give a summary on the main features of the improved MEP. All terminals are kept in the first positions (genes) of a chromosome and no other genes containing terminals are allowed in the rest of the chromosome. This improves efficiency, particularly since the generation of redundant individuals through mutation has a lower probability as compared with the original MEP.

3.1. IMEP Algorithm

S1. Randomly create the initial population P(0); // keeping all terminals in the first positions.

S2. for t = 1 to Max Generations do

S3. for k = 1 to |P(t)| / 2 do

S4. p_1 = Select(P(t)); // select one individual from the current population

S5. p_2 = Select(P(t)); // select the second individual

S6. Crossover (p_1, p_2, o_1, o_2); // crossover the parents p_1 and p_2 // the offsprings o_1 and o_2 are obtained

S7. Mutation (o_1); // mutate the offspring o_1

S8. Mutation (o_2); // mutate the offspring o_2

S9. Select best 2 individuals from {p_1,p_2, o_1,o_2} based on the fitness

S10. endfor

S11. Mutate a copy of the Best Individual

S12. Replace randomly an individual, except for the best one, with the mutated copy // avoid to lose the fittest

S13. Mutation (Worst Individual)

S14. Replace the worst individual with the worst mutated

S15. endfor

3.2. Fitness computation

Each circuit has one or more inputs (denoted by NI) and one or more outputs (denoted by NO). When multiple genes are required as outputs we have to select those output genes which minimize the difference between the obtained results and the expected output. Each of the IMEP chromosome expressions is considered as being a potential solution of the problem. Partial results are computed by Dynamic Programming [3]. A terminal symbol specifies a simple expression (a variable: circuit input). A function symbol specifies a complex expression obtained by connecting the operands specified by the argument positions with the current function symbol. The fitness of each sub-expression (gene) is calculated by computing this sub-expression for each fitness case (truth table input combinations) and then compared to the corresponding target value (truth table outputs): the fitness value is given by the number of not matching values. The chromosome fitness is defined as the fitness of the best expression(s) encoded by that chromosome. Fitness = 0 means that 100% of target values match with the values given by this (these) sub-expression(s). For more details, see [11];

3.3. The evolution operators

The evolution operators used within IMEP algorithm are Selection, Crossover and Mutation. They preserve the chromosome structure (see [11]) thus, all offspring are syntactically correct expressions.

- Selection: we use the Tournament with variable size. The size is dependent on the population size.
- Crossover: two parents are selected and recombined to produce offsprings. In our experiments, we have considered the multi-cut crossover, where several cuts are chosen randomly.
- Mutation: The first genes representing the problem variables are immune against probable changes. Function arguments always have indices of lower values than the position of that function in the chromosome. The function symbols can be mutated in only other function symbols and the links (function arguments) can also be mutated into only other links respectively. This reduces the probability of generating redundant individuals.

4. DISTRIBUTED IMEP FOR EVOLVING DIGITAL CIRCUITS

Biological evolution works in parallel. If we want to imitate the way nature works, evolutionary algorithms should be parallel and distributed. A more practical reason for using parallel EAs is the need for a large amount of computing power (large number of individuals to be evaluated and large number of generations required until the convergence process leads to a solution) when hard problems are tackled. Our objective is to reduce the

computational effort (see below), by distributing the task of the serial IMEPs to different processors on a dedicated cluster of workstations (Lido Cluster, University of Dortmund, Germany).

Parallel evolutionary algorithms may be classified into three main models: Global, Cellular and Islands models:

- Global model: uses parallelism to speed up the sequential GA. It uses a global shared population and the fitness evaluation is done on different processors.
- Cellular model: the population is separated into a large number of very small subpopulations which are maintained by different processors.
- Islands model: In general, the population is divided into a few large subpopulations called *demes*. The subpopulations are maintained by different processors. When the algorithm starts, all processors create their own random subpopulations with different random seeds. Each processor is responsible for selecting and mating in its own subpopulation. Every predetermined interval, some selected individuals are exchanged via a migration operator. Besides using similar parameters on each node, an island model allows the subpopulations to use different genetic parameters, coding, operators and objective functions to eventually increase diversity since each subpopulation evolves independently from the others. This model is called a heterogeneous islands model.

As mentioned in section 2, the third Model was adopted and the connection between the different nodes (processors) is a ring topology (see figure1).

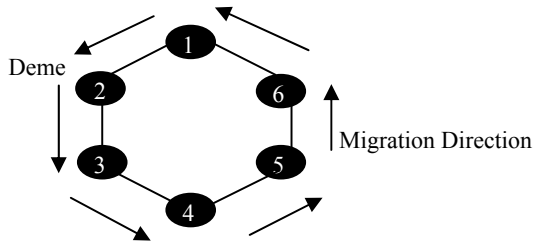

Figure 1 : Ring Topology

An Islands Model requires an identification of the migration policy:

1. What is the neighborhood? In the present case the topology as mentioned above, is a ring.
2. How and when do migrations occur? The period of migration is proportional to the number of generations. If one waits enough before migrating, the global diversity remains large. This means that different islands converge to different peaks. Exchange of individuals at this point results in a better average fitness.
3. How many solutions (Individuals) have to be sent? This parameter is very sensitive since a big number of migrated individuals may cause one island to dominate others and lose global diversity. On the other hand, a small number of migrated individuals causes slow convergence of the evolution process.
4. How to select migrating solutions? The best individual is selected (the best fitness).
5. Which solutions have to be replaced by the received solutions? In the present case the worst individuals are replaced by those received.

5. NUMERICAL EXPERIMENTS

In this section, numerical experiments with distributed IMEP for evolving digital circuits, are performed. For this purpose several well-known test problems [16], [17] are used.

A comparative study between sequential and distributed IMEP in the design of digital circuits and particularly in the considered test problems is done..

To assess the performance of the both algorithms, three statistics are considered: the maximum number of generations, the success rate and the computational effort:

Success Rate = Number of successful runs / the total number of runs.

Computational Effort: in [14] Koza describes a method to compare the results of different evolutionary methods. The so called Computational Effort is calculated as the number of fitness evaluations needed to find a solution of a problem with a probability of success z of at least z = 99%. One first calculates $P(M,i)$, the probability of success by the i-th generation using a population of size M. For each generation i this is simply the total number of runs that succeeded on or before the i-th generation, divided by the total number of runs conducted. One then calculates $I(M,i,z)$, the number of individuals that must be processed to produce a solution by generation i with probability greater than z (where z is usually 99%). The minimum of $I(M,i,z)$ over the range of i is defined as the "computational effort" required to solve the problem.

Koza defined the following equation:

$$I(M,z) = \min_i \; M(i) \; ceil\left[ln(1-z) / ln(1-P(M,i)) \right] \qquad (1)$$

P(M,i) = Ns(i) / N_{total}, where Ns(i) represents the number of successful runs at generation i and N_{total} represents the total number of runs.

There is not known definition of computational effort in the case of a distributed evolutionary algorithm. Different points of view may be possible. In the present paper, the computational effort of an islands model will be calculated in analogy to the computing time of a parallel system, to illustrate the obtained speed-up. The computational effort of every single process will be calculated independently and, for the whole process, the computational effort of the process obtaining the most successful runs will be considered as the computational effort of the distributed system. (Should several processes reach the same number of successful runs, then the lowest computational effort will be taken.)

5.1. Experiment Details

The performance of the DIMEP was tested on two different classes of problems shown in Table 1: multipliers and even parity problems.

Table 1. The experimental problems used to test the performance of DIMEP

Problem	Inputs	Outputs	Description
3x3-bit Multiplier	6	6	The Product of two 3–bit number to produce a 6–bit number
3x2-bit Multiplier	5	5	The Product of a 3-bit number and a 2-bit number to produce a 5-bit number
N-bit Even Parity problem	N=6,7	1	The function returns True if an even number or none of its arguments are True.

5.2. Results

The study aims to analyze the performance of DIMEP in term of

- Improvement of the problem solving process.
- Improvement of the speedup of the evolutionary process.
- Reduction of the computational effort.

The parameters setting used in all of the experiments, is given in Table 2.

Table 2. The parameters setting used in all experiments

Parameters	Range of Values
Number of Runs	50
Number of Islands	1 - 10
Migration Period	20 - 500
Number of Migrated Individuals	One Individual (Fittest)
Maximum Number of Generations	5,000 -100,000
Chromosome Length	100 - 300 (Gates + Problem inputs)
Population Size	5 - 100
Selection	Tournament: Size 2 - 6
Crossover	multi-cut crossover
Crossover Probability	0.9
Mutation Probability	0.02 - 0.03
Functions Set 1	A AND B, A AND NOT B, A XOR B
Functions Set 2	A AND B, A OR B, A NOR B

5.2.1. Multipliers

- 3x2–bit multiplier :

Table 3. 3x2–bit Multiplier : Chromosome length 100, Population size 5, Tournament size 2, Maximum runs 50 using Function Set 1.

Parameters	Values		
Number of Islands	1	5	10
Maximum Number of Generations	60,000	10,000	5,000
Migration Period	-	50	20
Mutation probability	0.03	0.03	0.03
Successful Runs	**50**	**50**	**50**
Computational Effort	**69,000**	**12,900**	**6,080**

- 3x3–bit multiplier :

Table 4. 3x3–bit Multiplier : Chromosome length 300, Population size 50, Tournament size 2, Maximum runs 50 using Function Set 1.

Parameters	Values		
Number of Islands	1	5	10
Maximum Nr. of Generations	500,000	50,000	20,000
Migration Period	-	100	100
Mutation probability	0.01	0.02	0.02
Successful Runs	**50**	**50**	**50**
Computational Effort	**1,864,450**	**400,950**	**311,850**

5.2.2. Parity problem

- 6–bit parity problem

Table 5. 6–bit Parity problem : Chromosome length 150, Population size 100, Tournament size 4, Maximum runs 50 using Function Set 2.

Parameters	Values		
Number of Islands	1	5	10
Maximum Nr. of Generations	100,000	100,000	100,000
Migration Period	-	500	500
Mutation probability	0.03	0.03	0.03
Successful Runs	**27**	**45**	**50**
Computational Effort	**18,734,200**	**2,622,000**	**1,796,400**

- 7–bit parity problem

Table 6. 7–bit Parity problem : Chromosome length 200, Population size 100, Tournament size 6, Maximum runs 50 using Function Set 2.

Parameters	Values		
Number of Islands	1	5	10
Maximum Nr. of Generations	100,000	100,000	100,000
Migration Period	-	500	200
Mutation probability	0.03	0.03	0.03
Successful Runs	**15**	**30**	**49**
Computational Effort	**45,696,400**	**15,522,000**	**5,321,400**

5.2.3. Interpretation of the results

In the case of the multiplier, we started with performing the sequential algorithm with different generation numbers, with/without varying the chromosome length and population size in order to reach 100% successful runs and also to optimize the parameters to be used when working with islands. Then, the number of islands was increased progressively.

In the case of the parity problem, the approach was different: the maximum number of generations was fixed for both sequential and parallel programs.

The motivation for using the islands model is twofold. First, one wants to improve the speedup of the evolutionary process by creating multiple independent populations of smaller size. Second, one intends to improve the problem solving process by increasing diversity. In both cases, one can easily deduce that applying the islands model to the IMEP method, decreases considerably the computational effort.

In the first case, our aim was to prove that even though a small population size is considered, the speedup is reached by decreasing the number of generations and by increasing the number of islands.

In the second case, we show that the diversity created by the migration phenomenon, increases the number of successful runs when the number of islands is increasing.

Some of the evolved circuits relative to the examples given in this paper are given by figures 2 through 5 (see appendix). Input or output terminals marked with a black dot are meant to be holding a logical 1; otherwise, a logical 0.

6. CONCLUSION AND FUTURE WORKS

In this paper, a distributed implementation of the Improved Multi Expression Programming (DIMEP) was used to evolve well known benchmark problems such as multipliers and parity problems.

Different experiments were performed to compare the Distributed IMEP and the Sequential one. The Islands model was considered using a ring topology connection. The performance analysis of the distributed algorithm was measured in terms of maximum number of generations, successful runs, and computational effort.

Three aspects were considered :

- Improving the problem solving process by increasing diversity in the population; the case of the parity problem.
- Improving the speedup of the evolutionary process by decreasing the number of generations; the case of multiplier.
- Decreasing the computational effort; both cases.

As future work, the Influence of the migration policy on the behavior of the island model applied to the evolution process as well as the neighborhood structure and the number of migrated individuals, will be studied. Furthermore other topologies, like the hypercube or CCC [12] will be considered.

7. REFERENCES

[1] Alba, E. and Tomassini, M., Parallelism and evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 6, 5, (2002), 443-462.

[2] Banzhaf, W., Nordin, P., Keller, R.E. and Francone, F.D., *Generic Programming: An Introduction on the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann ,1998

[3] Bellman, R., Dynamic Programming, *Princeton University Press*, New Jersey, 1957.

[4] Cantú-Paz, E., A survey of parallel genetic algorithms, *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10, 2, (1998), 141–171

[5] Cheang, S. M., Leung, K. S., and Lee, K. H., Genetic Parallel Programming: Design and Implementation, *Evolutionary Computation*, 14, 2, (2006), 129-156

[6] Coello, C.A.C., Alba, E. and Luque, G., Comparing Different Serial and Parallel Heuristics to Design Combinational Logic Circuits. In *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware*, 2003, 3-12.

[7] Corcoran, A. L. and Wainwright, R. L., A parallel island

model genetic algorithm for the multiprocessor scheduling problem. In *Proceedings of the 1994 ACM/SIGAPP Symposium on Applied Computing*, 1994, 483-487.

[8] Fernandez, F. and Tomassini, M., Improving Parallel Genetic Algorithm GA Performance by Means of Plagues, *Advances in Soft Computing* 2, Springer Verlag, Berlin, Heidelberg, (2005), 515-523,.

[9] Fernández, F., Tomassini, M. and Vanneschi, L., Studying the influence of Communication Topology and Migration on Distributed Genetic Programming, In J. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A. G.B. Tettamanzi, W. Landdon, *LNCS* 2038 *Genetic Programming, 4th European Conference, EuroGP 2001*, Springer Verlag, Berlin, (2001), 51-63.

[10] Ferreira, C., Gene Expression Programing: A New Adaptive Algorithm for Solving Problems, *Complex Systems*, 13, (2001), 87-129

[11] Hadjam F. Z., Moraga C., and Hildebrand L.: Evolutionary design of digital circuits using Improved Multi-Expression Programming. Research Report 812, Faculty of Informatics, University of Dortmund, Germany, 2007. (Copy may be obtained from the authors)

[12] Herrera F, Lozano M, Moraga C.: Hybrid distributed real-coded genetic algorithms. *Lecture Notes in Computer Science* 1498, Springer Verlag, Berlin (1998), 603-612

[13] Koza, J.R. and Andre, D., Parallel genetic programming on a network of transputers, *Proc. of the Workshop on Genetic Programming: From Theory to Real- World Applications*. University of Rochester. National Resource Laboratory for the Study of Brain and Behavior. Technical Report 95-2, 111–120, 1995

[14] Koza, J.R., *Genetic Programming. On the Programming of Computers by means of Natural Selection*, MIT Press, 1992.

[15] Miller, J.F., and Thomson, P., Cartesian Genetic Programming. In *Proc. of the 3rd International Conference on Genetic Programming (EuroGP2000), LNCS* 1082, SpringerVerlag, Berlin, (2000), 15-17

[16] Miller, J.F., Job. D. and Vassilev, V.K., Principles in the Evolutionary Design of digitals circuits- Part I, *Genetic Programming and Evolvable Machines*, 1, 1, (2000), 7-35.

[17] Oltean M., Solving Even-Parity Problems using Multi Expression Programming, *Proceedings of the 5th International Workshop on Frontiers in Evolutionary Algorithms, The 7th Joint Conference on Information Sciences*, (September 26-30, 2003, Research Triangle Park, North Carolina), Edited by Ken Chen (et. al), 2003, 315-318.

[18] Oltean M., Multi-expression Programming, Technical Report, Babes-Bolyai Univ, Romania, 2006.

[19] Oltean, M. and Grosan, C., Solving Classification Problems using Infix Form Genetic Programming, *The Fifth International Symposium on Intelligent Data Analysis*, edited by M. Berthold (et. al), *LNCS* 2810, Springer Verlag, Berlin, (2003), 242–252

[20] Oltean, M. and Grosan, C., A Comparison of Several Linear Genetic Programming Techniques, *Complex-Systems*, 14, 4, (2003), 282-311.

[21] Oltean, M. and Grosan, C., Evolving Digital Circuits using Multi Expression Programming. *NASA/DoD Conference on Evolvable Hardware*, (24-26 June, Seattle), Edited by R. Zebulum (et. al), IEEE Press, NJ, 2004, 87-90

[22] Patterson, N., Genetic Programming with Context-Sensitive Grammars, Ph.D. Thesis, School of Computer Science, University of Scotland, 2002

[23] Ryan, C., O'Neill, M., Grammatical Evolution : A Steady State Approach, *Late Breaking Paper*, *Genetic Programing* Edited by J.R. Koza (University of Wisconsin, Madison, Wisconsin, 1998).

[24] Skolicki, Z., An Analysis of Island Models in Evolutionary Computation. In *Proceeding of the 2005 workshops on Genetic and evolutionary computation*, (Washington, D.C. June 2005), 2005, 25 - 26.

Appendix: Circuits

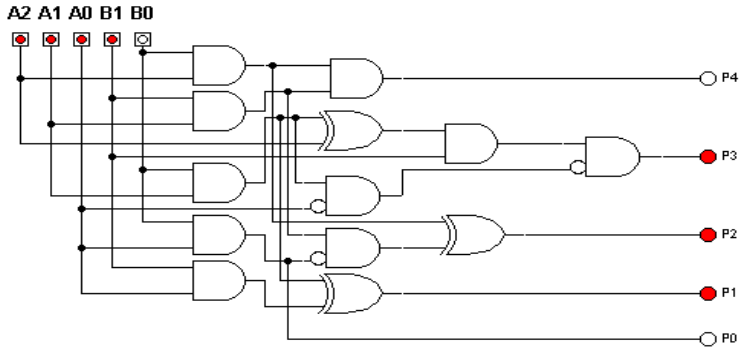

Figure 2 : Evolved 3x2–bit multiplier : 13 gates with 4 levels, using {and, and with one input inverted, xor}

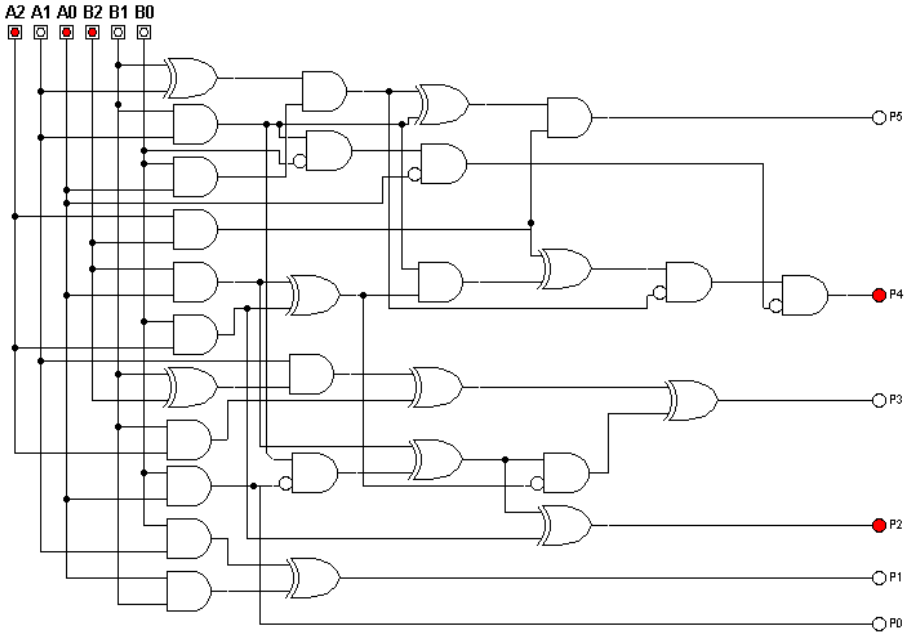

Figure 3 : Evolved 3x3–bit multiplier : 29 gates with 6 levels, using {and, and with one input inverted, xor}

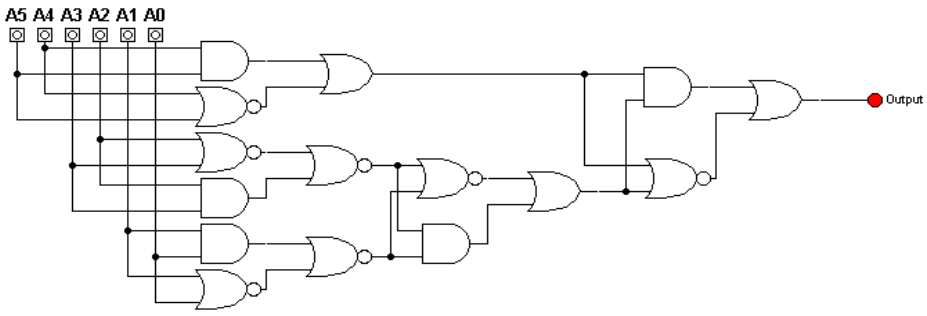

Figure 4 : Evolved 6–bit parity problem : 15 gates using {and, or, nor}

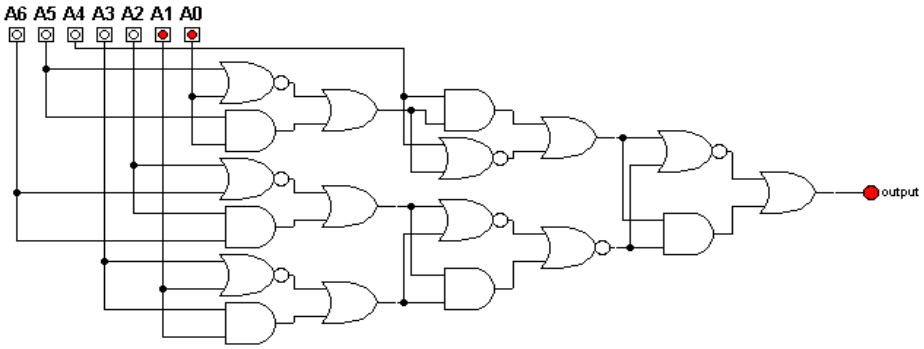

Figure 5 : Evolved 7–bit parity problem : 18 gates using {and, or, nor}