A Constructive Genetic Algorithm for the Linear Gate Assignment Problem

Alexandre César Muniz de Oliveira	Luiz Antonio Nogueira Lorena
DEINF - Universidade Federal do Maranhão Campus Bacanga, São Luís, MA, Brasil acmo@deinf.ufma.br	LAC - Instituto Nacional de Pesquisas Espaciais Jd da Granja, 12201-970, S. J. dos Campos, SP, Brasil lorena@lac.inpe.br

***Abstract* - We present in this paper an application of the Constructive Genetic Algorithm (CGA) to the Linear Gate Assignment Problem (LGAP). The LGAP happen in very large scaling integration (VLSI) design, and can be described as a problem of assigning a set of circuit nodes (gates) in an optimal sequence, such that the layout area is minimized, as a consequence of optimizing the number of tracks necessary to cover the gates interconnection. The CGA has a number of new features compared to a traditional genetic algorithm. These include a dynamic population size composed of schemata and structures, and the possibility of using heuristics in structure representation and in the fitness function definitions. In our application of CGA to LGAP we use a 2-Opt like heuristic to define the fg-fitness and local search mutation. Computational tests are presented using available instances taken from the literature.**

Key words: Constructive genetic algorithms, linear gate assignment, VLSI layout design.

I. INTRODUCTION

The *Constructive Genetic Algorithm (CGA)* was proposed recently as an alternative to a traditional *GA* approach [1], particularly, for evaluating schemata directly. The population, initially formed only by schemata, evolves controlled by recombination to a population of well adapted structures (schemata instantiation) and schemata.

The *CGA* application can be divided in two phases, the constructive and the optimal:

- The *constructive phase* is used to build a population of quality solutions, composed of well adapted schemata and structures.

- The *optimal phase* is conducted simultaneously and transform the optimization objectives of the original problem on an interval minimization problem, that evaluates schemata and structures in a common way.

In the same lines, Goldberg and collaborators [2,3] have introduced an alternative GA, the messy-GA, that allows variable length strings that look for the construction and preservation of good schemata.

Linear gate assignment problems (*LGAP*) are related to gate matrix layout and programmable logic arrays folding [4,5]. In very large scaling integration design (VLSI design), the goal is to arrange a set of circuit nodes (gates) in an optimal sequence, such that the layout area is minimized, e.g., it minimizes the number of tracks necessary to cover the gates interconnection.

Fig.1 shows an example of gate matrix, where the gates are numbered from 1 to 9 and the dots are the connection requests (*Fig.1a*). A group of gates at the same connection is called *net*. There are 7 nets in circuit of *Fig.1*. To connect gates 1,3,4 and 7 of the net 1, it is necessary to cross gates that are not part of this net. Moreover, non-overlapping nets can be placed at the same connection track. To compute the number of tracks to cover all nets, it is enough to verify the maximum of overlapping nets. The number of tracks is an important factor of the cost of VLSI circuits. The *Fig.1b* shows, at the bottom, the overlaps and the maximum of them. One can see that the number of tracks in that gate matrix is 7. Besides, there is also the cost of connection of the gates that symbolizes the amount of necessary metal to cover the nets. In this example, the total metal length is 3+3+3+5+6+7+7+5+3=42.

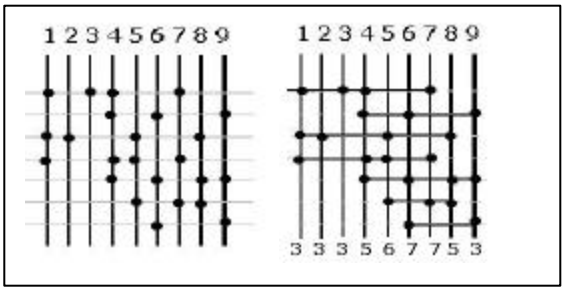

Fig.1 - Gate matrix. a) Original matrix; b) Gate matrix derived by interconnection of gates at same net

This paper is organized as follows. In section 2 we present aspects of modeling that involves definitions of the schema and structure representations and the consideration of the problems at issue as *bi-objective* optimization problems. Section 3 describes the *CGA* operators, namely, selection, recombination and

mutation, as well as a *CGA* pseudo-code. Section 4 shows computational results using instances taken from the literature.

II. CGA MODELING

In this section is described the modeling phase of the *CGA*. The *LGAP* is formulated as a bi-objective optimization problem. Two fitness functions are defined on the space of all schemata and structures that can be obtained using a specific representation. The evolution process considers the two objectives on an adaptive rejection threshold, which gives ranks to individuals in population and yields a dynamic population.

A typical instance of the *LGAP* is composed of an *I x J* binary matrix, where *I* and J are the numbers of *nets* and *gates*, respectively. The initial sequence of gates is {1,2,3,4,...,J}. Very simple structure and schema representations are adopted to the *LGAP*. They use a direct alphabet of symbols representing the gate sequence (each gate representing their connection requirements). The 1 to J values are reserved to gate number and the symbol # is used to indetermination (# - *do not care*) on schemata. *Fig.2* shows a *LGAP* instance and a permutation over it, representing structures and an example of schema.

1 0 1 1 0 0 1 0 0	1 0 0 0 1 1 0 1 0	1 ? 0 ? ? ? 0 1 ?
0 0 0 1 0 1 0 0 1	0 0 0 1 0 1 1 0 0	0 ? 0 ? ? ? 1 0 ?
1 1 0 0 1 0 0 1 0	0 1 1 0 0 0 0 1 1	0 ? 1 ? ? ? 0 1 ?
1 0 0 1 1 0 1 0 0	1 0 1 0 0 1 0 1 0	1 ? 1 ? ? ? 0 1 ?
0 0 0 1 0 1 0 1 1	0 0 0 1 0 1 1 0 1	0 ? 0 ? ? ? 1 0 ?
0 0 0 0 1 0 1 1 0	1 0 1 0 0 0 0 0 1	1 ? 1 ? ? ? 0 0 ?
0 0 0 0 0 1 0 0 1	0 0 0 1 0 0 1 0 0	0 ? 0 ? ? ? 1 0 ?
s_i=(1 2 3 4 5 6 7 8 9)	s_j=(7 2 5 9 3 4 6 1 8)	s_k=(7 # 5 # # # 6 1 #)

Fig.2 - a) Net-gate requirements matrix and structure s_i(derived from Fig.1a); b) Example of net-gate permutated matrix and corresponding structure s_j; c) Example of schema.

Let X be the space of all schemata and structures that can be formed by this representation. The *CGA* is modeled as the following Bi-objective Optimization *Problem (BOP)*:

$$\begin{aligned} &Min && \{g(s_k) - f(s_k)\} \\ &Max && g(s_k) \\ &\text{subject to} && g(s_k) \geq f(s_k) \\ &&& \forall\, s_k \in C \end{aligned} \qquad (2.1)$$

Function *g* is the fitness function that reflects the total cost of a given permutation of gates. Commonly, it is used the formulation that considers track minimization as primary objective and wire length minimization as a secondary one. Therefore, it is used the following cost function [6] as the fitness function *g*:

- $$g(s_k) = I.J \cdot \max_{j \in \{1,\ldots,J\}} \sum_{i=1}^{I} b_{ij} + \sum_{j=1}^{J} \sum_{i=1}^{I} b_{ij} \qquad (2.2)$$

where b is the matrix generated by transforming in *ones* all zeros between most left and most right ones in nets, for all gates on the sequence corresponding to s_k (structure or schema), and the $I.J$ product is a weight to reinforce the part of the objective considering the maximum number of tracks and to make it proportional to the second part of the objective concerning the wire length. If s_k is schema, the non-defined columns (# label) are bypassed. The b matrix used to compute $g(s_k)$ contains only columns with information.

The other fitness function f is defined to drive the evolutionary process to a population trained by a heuristic. The chosen heuristic is the *2-Opt* neighborhood. Thus, function f is defined by:

- $$f(s_k) = g(s_v), s_v \in \{s_1, s_2, \ldots, s_V\} \subseteq j^{2-Opt}, g(s_v) \leq g(s_k) \qquad (2.3)$$

where j^{2-Opt} is a *2-Opt* neighborhood of structure or schema s_k.

As the *2-Opt* neighborhood can grow exponentially, in our implementation, only a fixed part, randomly chosen, of this space is used on each function call. Implementation details can be found on section 3.3.

By definition, f and g can be applied to structures and schemata, just differing in the amount of information and consequently in the costs associated to them. More information means greater cost. In this way, the g maximization objective in *BOP* drives the constructive phase of the *CGA*. The optimal phase is conducted by the interval minimization $g - f$, which happen on structures with near local *2-Opt* minimum.

III. THE EVOLUTION PROCESS

The *BOP* defined above is not directly considered as the set X is not completely known. Instead we consider an evolution process to attain the objectives (*interval minimization* and *g maximization*) of the *BOP*. At the beginning of the process, two *expected values* are given to these objectives:

- *g maximization* → we use a value $g_{max} > \max_{s \in X} g(s)$ that is an upper bound on the objective value
- *interval minimization* → we use a value dg_{max}, obtained from g_{max} using a real number $0 < d \leq 1$.

The evolution process is then conducted considering an adaptive rejection threshold, which contemplates both objectives in *BOP*. Given a parameter $a \geq 0$, the expression

$$g(s_k) - f(s_k) \geq d\,g_{\max} - a \cdot d[g_{\max} - g(s_k)] \qquad (2.4)$$

presents a condition for rejection from the current population of a schema or structure s_k.

The right hand side of *(2.4)* is the threshold, composed of the expected value to the interval minimization $d\,g_{\max}$, and the measure $[g_{\max} - g(s_k)]$, that shows the difference of $g(s_k)$ and $g_{\max}$ evaluations.

Parameter a is related to time in the evolution process. Considering that the good schemata need to be preserved for recombination, the *evolution parameter* a starts from 0, and then increases slowly, in small time intervals, from generation to generation. The population at the evolution time a, denoted by P_a, is dynamic in size accordingly the value of the adaptive parameter a, and can be emptied during the process. The parameter a is now isolated in expression *(2.4)*, thus yielding the following expression and corresponding rank to s_k:

$$a \geq \frac{dg_{\max} - [g(s_k) - f(s_k)]}{d[g_{\max} - g(s_k)]}. \qquad (2.5)$$

The right hand side of expression *(2.5)* gives a *rank* value to s_k:

$$d(s_k) = \frac{dg_{\max} - [g(s_k) - f(s_k)]}{d[g_{\max} - g(s_k)]}. \qquad (2.6)$$

At the time they are created, structures and/or schemata receive their corresponding rank value $d(s_k)$. The *rank* of each schema or structure is compared with the current evolution parameter a. At the moment a structure or schema is created, it is then possible to have some figure of its survivability. The higher the value of $d(s_k)$, and better is the structure or schema to the *BOP*, and they also have more surviving and recombination time.

For the *LGAP*, the overall bound $g_{\max}$ is obtained at the beginning of the *CGA* application, by generating a random structure and making $g_{\max}$ receive the g evaluation for that structure. In order to ensure that $g_{\max}$ is always an upper bound, after recombination, each new structure generated s_{new} is rejected if $g_{\max} \leq g(s_{new})$.

A. Initial population

The initial population is composed exclusively of schemata, considering for each schema, a proportion of random positions receiving a unique gate number. The remaining positions receive label #. Along the generations, the population can increase by addition of new offspring generated out of the combination of two schemata.

B. Recombination

There are two purposes on the evolution process: to obtain structures (good solutions to the *g maximization* objective on the *BOP*), and that these structures be good ones (best solutions to the *interval minimization* problem on the *BOP*). The selection of structures and/or schemata for recombination will be conducted to attain these two objectives. The first one is attained by selecting for recombination schemata with small number of labels #, and the second considering structures or schemata with small $d_k = \frac{g(s_k) - f(s_k)}{g(s_k)}$.

The structures and schemata in population P_a are maintained on ascending order, according to the key $\Delta(s_k) = (1 + d_k)/h$, where η is the number of genes containing information (not #). Thus, individuals with more genetic information (structures or semi-complete schemata) appear in first order places on the population.

Two structures and/or schemata are selected for recombination. The first is called the *base* (s_{base}) and is randomly selected out of the first positions in P_a, and in general it is a good structure or a good schema. The second structure or schema is called the *guide* (s_{guide}) and is randomly selected out of the total population. The objective of the s_{guide} selection is the conduction of a guided modification on s_{base}. *Fig.3* represents the base-guide selection process.

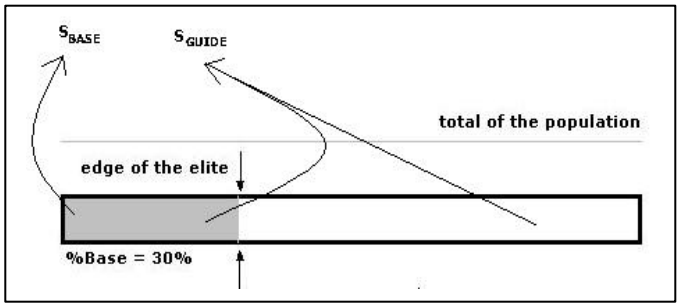

Fig.3 - Base-guide selection

The current labels in corresponding positions are compared. Let s_{new} be the new structure or schema (offspring) after recombination. Structure or schema s_{new} is obtained by applying the following operations:

{ Recombination }
For i **from** 1 **to** *individual length*

i	**If** s_{base} (i) = #	**and** s_{guide} (i) = #	**then set** s_{new}(i) = #	
ii	**If** s_{base} (i) <> #	**and** s_{guide} (i) = #	**then set** s_{new}(i) = s_{base} (i)	**if** s_{base} (i) it is not in s_{new}
			else set s_{new}(i) = #	
iii	**if** s_{base} (i) = #	**and** s_{guide} (i) <> #	**then set** s_{new}(i) = s_{guide} (i)	**if** s_{guide} (i) it is not in s_{new}
			else set s_{new}(i) = #	
iv	**if** s_{base} (i) <> #	**and** s_{guide} (i) <> #	**then set** s_{new}(i) = s_{base} (i)	**if** s_{base} (i) it is not in s_{new}
			else set s_{new}(i) = s_{guide}	**if** s_{guide} (i) it is not in s_{new}
			else set s_{new}(i) = #	

Observe that s_{base} is a privileged individual to compose s_{new}, but it is not totally predominant. There is a small probability of s_{guia} gene information is used instead of.

C. The algorithm

The *Constructive Genetic Algorithm* can be summed up by the pseudo-code:

CGA { Constructive Genetic Algorithm }

```
Given g_max and d ; α := 0 ; ε := 0.05;          { initialization of parameters }
Initialize P_α ;                                  { initial population }
A. For all s_k ∈ P_α do compute g(s_k), f(s_k), d(s_k)      {Evaluate P_a }
While (not stop condition) do
        While (number of recombination) do
                Select Base and Guide from P_α ;          {selection operator }
                Recombine Base and Guide                  { recombination operator }
                Evaluate Offspring;                       { fg-fitness and ranking }
                Update Offspring in P_α;                  { update P_a }
        end_while
        α := α + ε ;
        For all s_k ∈ P_α satisfying α > d(s_k) do
                Eliminate s_k from P_α
        end_for
end_while
```

The ε increment is a linear step that increases the adaptive rejection threshold. The stop conditions occur with an emptied population (assured by a sufficiently higher a) or at a predefined number of generations. The population increases, after the initial generations, reaching an upper limit (in general controlled by storage conditions), and decreases for higher values of the evolution parameter a.

The *CGA* algorithm begins with the recombination procedures (over schemata only) and the constructive process builds structures (full individuals) progressively, on each generation. By this way, the constructive process, repeatedly, uses genetic information contained in two individuals to generate another one. However, the constructive process can be complemented using especially designed *mutation* and *filling* heuristics, searching for a better overall performance. This filling process is called *unary filling operator* and is applied before the recombination. It consists to fill the s_{base} substituting the # labels for gate numbers. For *LGAP*, this feature was implemented as following:

{Unary filling operator}

Select *Base and Guide* from P_α ; *{selection operator }*
If *Base* **is a** schemata **then**
Fill *Base* **giving** *filled_Base;* *{heuristic for filling}*
Recombine *Base and Guide* *{ recombination operator }*
end_if
Mutation on *full Base (or filled_Base);* *{local search on structures only}*
Evaluate *Offspring*; *{ fg-fitness and ranking }*
Update *Offspring* in $P_{\alpha;}$ *{ update P_a }*

The code fragment before replaces the more internal while loop of *CGA*. If the selected base is a schema, it is combined with the guide individual (schema or structure) giving a new individual. In any way, local search mutation is applied to a structure (full or filled base) and the offspring are updated depending on its rank. If a best structure is found, it is kept for the end. For filling schemata it is used an algorithm based on neighborhood minimization. Given a schema s_k to be filled in a good sequence with gate columns that still not belong to it at # possible positions.

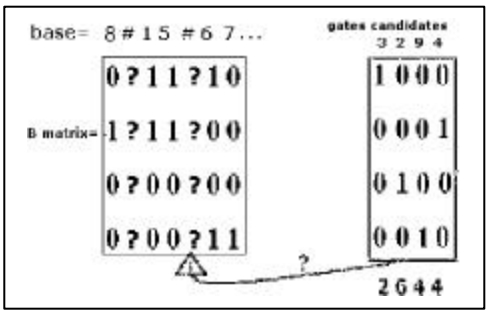

Fig.4 - Rule for filling schemata

In the example on *Fig.4*, there are 4 candidate gate columns (on right) to include in a partially showed schema at the marked position (# position). For each # position will be tested all candidate gate columns. It will be chosen the gate column that minimizes the bit-to-bit xor relation considering the neighbors gate columns (left and right of the # point, if exists). The bit-to-bit *xor* sum of first candidate column is 2, and is the smaller sum. This mechanism intends to decrease the possibility of zeros between ones at the included gate columns.

The local search mutation is always applied to structures, no matter how they are created (after recombination or after the filling process). The search at *2-Opt* neighborhood of the structure was used (like the f definition on *expression 2.3*). To avoid the computational effort growing proportional to problem length caused by *2-Opt* procedure, a constant pre-defined number of neighbors is inspected until the best is found. The neighbors are generated by all the 2-move changes in a constant length part of the structure. It is chosen a position at random and starting from there an interactive process that inspect all possible 2-move changes in the structure.

The example of 2-move change is showed in *Fig.5*. The marks in positions of the structures mean reference points to be change. Non consecutive references cause the first change type, as showed in *Fig.5a*. Consecutive points cause the second change type in *Fig.6b*. Inspecting all or part of *2-Opt* neighborhood needs several moves (catching 2-to-2 points).

The amount of 2-move changes (neighborhood width) on each local search mutation is a *CGA* parameter of execution to be tuning. This and all the others settings will be describes in next section.

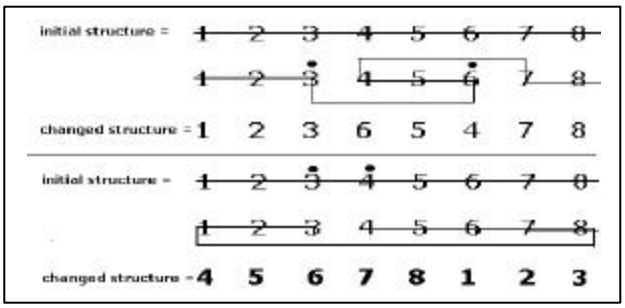

Fig.5 - Examples of one-move in 2-Opt neighborhood; a) non-consecutive reference points change; b) consecutive reference points change.

IV. COMPUTATIONAL RESULTS

The *CGA* for *LGAP* was coded in *ANSI C* and it was run on Intel Pentium II (266Mhz) hardware. For the computational tests, some *CGA* parameters were set as follows: *e* and *d* parameters were set to 0.005 and 0.15, respectively; each schema of initial population had received 50% of # genes (indetermination percentage), and 10% up to 20% of population were considered base individuals for base-guide selection. Local search mutation rate was fixed in 100%. The number of individuals initially generated was proportional to problem length (gate number equal to initial population at least). Other important parameter to tuning is neighborhood width (*nw*) to each local search mutation. After several simulations, we found good results using *nw* = 20. The ideal would be to use great values for *nw*, but this would turn the mutation very slow too.

Table I presents the problem circuits considered in this work, along with the corresponding number of nets and gates, a lower bound to the track number, the previous best known number of tracks (solution) and its reference. Chosen problems were the largest ones found in the literature and others for which authors have obtained recent improvements in results. The lower bound is computed by maximum column sum of the original matrix. Most of the best previous results comes of microcanonical optimization approach (MCO)[6], but other ones with same best results were included as in references: *GM_Learn* [7] and *GM_Plan* [8].

problem	gate	net	LB	best tracks	Refs
wli	10	11	4	4	[6,8]
wsn	25	17	4	8	[6,8]
v4000	17	10	5	5	[6]
v4050	16	13	5	5	[6,7]
v4090	27	23	9	10	[6,7]
v4470	47	37	5	9	[6,7]
x0	48	40	6	11	[6,7]
w1	21	18	4	4	[6,8]
w2	33	48	14	14	[6,8]
w3	70	84	11	18	[6]
w4	141	202	18	27	[6]

Table I - Chosen problems to compare

Microcanical Optimization (MCO) approach overcame (or match) all the other approaches and it was chosen for comparison with the results of *CGA. Table II* presents the comparison among the best results of *MCO.*

	MCO				CGA				
Prob	**tracks:10 replications**	**tracks:1000 replications**	**one trial time**	**1000 trial time**	**Tracks:10 replications**	**wire lenght**	**Gen.**	**one trial time**	**10 trial time**
wli	5.00	4.00	0.01	10.00	4.00	35.00	5.00	0.50	5.00
wsn	8.00	-	0.01	10.00	8.00	115.00	7.00	1.50	15.00
v4050	5.00	-	0.01	10.00	5.00	51.00	5.00	0.50	5.00
v4000	6.00	5.00	0.01	10.00	5.00	66.00	5.00	0.50	5.00
v4470	10.00	9.00	0.70	700.00	9.00	269.00	33.00	66.50	665.00
v4090	10.00	-	0.10	100.00	10.00	132.00	13.50	2.03	20.33
x0	11.00	11.00	0.70	700.00	11.00	343.00	92.57	75.56	755.60
w1	4.00	-	0.01	10.00	4.00	57.00	5.00	1.00	10.00
w2	14.00	-	0.40	400.00	14.00	283.00	19.50	18.50	185.00
w3	21.00	18.00	3.90	3900.00	18.00	761.00	186.00	306.25	3062.50
w4	32.00	27.00	61.70	61700.00	27.00	1932.00	225.00	5224.67	52246.67

Table II - CGA comparison to MCO

The *MCO* was run initially for 10 replications, but only after 1000 replications the authors reached their best results for problems *wli, v4000, v4470, x0, w3 and w4* . The *CGA* best result were obtained after 10 replications only at same hardware environment. To compare different number of replications with respect to execution time, the total time of all the replications was computed based on one trial execution time. Thus, the total time of *CGA* experiments is 10 times one trial execution. By its turn, the total time of MCO experiments is 1000 times one trial execution. The "-" execution time, reported in [6] as close to zero, to our computations, was turned in 0.01 seconds, that is the smallest fraction of one second with two decimal digits.

The number of tracks columns refer to the best result found for each problem. The *CGA* columns present the wire length value, not reported in other references. Observing *Table II*, *CGA* found the same results of MCO, but the total time of all replications to find the best results is smaller.

In [6], the author remarks the frequency of 36.3% to find the best result for *wli* problem (relatively small) in 1000 replications. In our experiments, this percentage is 100% in 10 replications. In *CGA*, smaller percentages were found only in larger problems (w3 and w4).

	CGA 5 best solutions										% best tracks
wsn	115	8	115	8	116	8	118	8	124	8	100.00
wli	35	4	35	4	35	4	35	4	35	4	100.00
v4050	51	5	51	5	51	5	52	5	52	5	100.00
v4000	66	5	66	5	66	5	67	5	68	5	100.00
v4470	269	9	276	9	288	9	289	9	305	9	100.00
v4090	132	10	133	10	133	10	135	10	136	10	90.00
x0	343	11	344	11	345	11	346	11	348	11	80.00
w1	57	4	57	4	57	4	57	4	61	4	100.00
w2	283	14	284	14	315	14	321	14	323	14	100.00
w3	761	18	771	18	918	18	927	18	801	18	50.00
w4	1932	27	2038	27	2051	27	1947	28	1947	28	30.00

Table III - Five best results and frequencies found for best tracks

The *Table III* shows the *CGA* 5 best results (wire length and tracks) obtained in 10 replications. Problems *w3, w4, x0 v4470* have less than 100% of frequency in reaching their best solution. The w4 problem appears to be the most difficult one, but these results show that no more than 10 trials were necessary to *CGA* to find the best known solutions. *CGA* seems to be more robust than MCO.

V. CONCLUSION

This work describes a constructive approach to genetic algorithms and an application to *linear gate assignment problems (LGAP)*. The *CGA* adapted to work with *LGAP* presents some new specific features, like the filling operator and *2-Opt* heuristic used at local search mutation, besides the two fitness functions (*f* and *g*). On computational tests, the *CGA* reached all the best results (number of tracks) for instances taken from the literature, but it appears to be more robust than other approaches. There is not enough information

about wire length in the found literature to comparison of all the approaches completely. Applications of *CGA* to other classes of problems are foreseen for future works.

References

1 L.A.N Lorena and J.C. Furtado. *Constructive Genetic Algorithm for Clustering Problems*. Evolutionary Computation - to appear (2000). Available from http://www.lac.inpe.br/~lorena/cga/cga_clus.PDF

2 Goldberg, D.E.; Korb, B.; Deb, K. Messy genetic algorithms: Motivation, analysis, and first results. Complex Systems v. 3: p. 493-530, 1989

3 Goldberg, D.E.; Deb, K.; Kargupta, H.; Harik, G. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. IlliGAL Report No. 93004, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, 1993.

4 R. Möhring. *Graph problems related to gate matrix layout and PLA folding*. Computing, Vol 7, pp. 17-51, 1990.

5 Kashiwabara, T. and Fujisawa T., NP-Completeness of the problem of finding a minimum clique number interval graph containing a given graph as a subgraph in Proc Symp. Circuits and Systems. 1979.

6 A. Linhares, H. Yanasse and J.R.A. Torreão. *Linear Gate Assignment: a Fast Statistical Mechanics Approach*. IEEE Trans. on Computer-Aided Designed of Integrated Circuits and Systems. Vol. 18(12), pp. 1750-1758. 1999.

7 S.J. Chen and Y. H. Hu. *GM_Learn: an interactive learning algorithm for CMOS gate matrix layout*. IEE Proc E, vol 137, pp 301. 1990

8 Y. H. Hu and S. J. Chen. *GM_Plan: a gate matrix layout algorithm based on artificial intelligence planning techniques* . IEEE Trans. Computer-Aided Designed, Vol. 9, pp. 836-845, 1990.