A Constructive Genetic Algorithm for Gate Matrix Layout Problems

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* - This paper describes an application of a Constructive Genetic Algorithm (*CGA*) to the Gate Matrix Layout Problem (*GMLP*). The *GMLP* happens in very large scale 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 population of dynamic size composed of schemata and structures, and the possibility of using heuristics in structure representation and in the fitness function definitions. The application of *CGA* to *GMLP* uses a *2-Opt* like heuristic to define the fitness functions and the mutation operator. Computational tests are presented using available instances taken from the literature.**

Key words: Constructive genetic algorithms, Gate Matrix Layout, *VLSI* layout design.

I. INTRODUCTION

Gate Matrix Layout problems (*GMLP*) are related to one-dimensional logic arrays and programmable logic arrays folding [1,2]. In very large scale 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 cost factor of *VLSI* circuits.

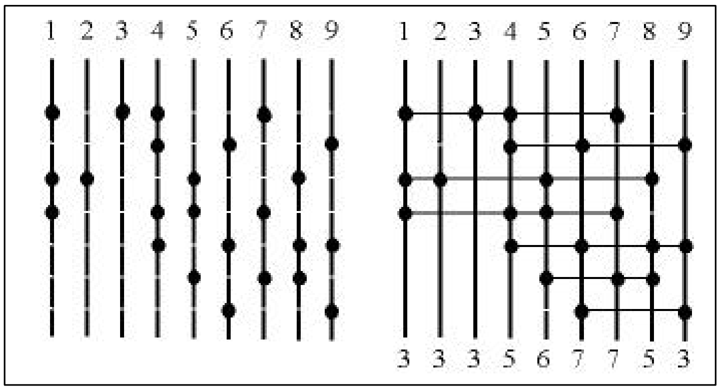

Fig.1 - Gate matrix. a) Original matrix; b) Gate matrix derived by interconnection of gates at same net

Fig.1b shows, at the bottom, the number of overlaps at each gate. A typical instance of the *GMLP* is composed of an $I \times J$ binary matrix, where I and J are the numbers of *nets* and *gates*, respectively. The original matrix is represented in the following by a matrix a where all dots are transformed in *ones* and the other positions receive *zeros*, and the derived interconnected matrix will be denoted by b, which is generated by transforming in ones all zeros between leftmost and rightmost ones in matrix a. *Fig. 2* shows the matrices a and b related to the matrices in *Fig.1*.

1 0 1 1 0 0 1 0 0	1 1 1 1 1 1 1 0 0
0 0 0 1 0 1 0 0 1	0 0 0 1 1 1 1 1 1
1 1 0 0 1 0 0 1 0	1 1 1 1 1 1 1 1 0
1 0 0 1 1 0 1 0 0	1 1 1 1 1 1 1 0 0
0 0 0 1 0 1 0 1 1	0 0 0 1 1 1 1 1 1
0 0 0 0 1 0 1 1 0	0 0 0 0 1 1 1 1 0
0 0 0 0 0 1 0 0 1	0 0 0 0 0 1 1 1 1
Matrix a	*Matrix b*

Fig. 2 – Derived matrices a and b.

The number of tracks (*TR*) is easily calculated by:

$$TR = \max_{j \in \{1,\ldots,J\}} \sum_{i=1}^{I} b_{ij} = \max\{3,3,3,5,6,7,7,5,3\} = 7. \tag{1.1}$$

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 wire length (*WL*) is:

$$WL = \sum_{j=1}^{J} \sum_{i=1}^{I} b_{ij} - I = (3+3+3+5+6+7+7+5+3) - 7 = 35. \tag{1.2}$$

GMLP is also a permutation problem, and considering that the initial sequence of gates is {1,2,3,4,...,J}, both *TR* and *WL* can be improved by permutated matrices that represent particular sequences of gates.

Fig. 3 shows a permutated gate matrix corresponding to the sequence of gates {6,9,4,8,7,5,1,2,3}. The *TR* is reduced to *5* and the new *WL* is *22*. Nets 3 and 7 can be placed at the same connection track (the same for nets 2 and 6).

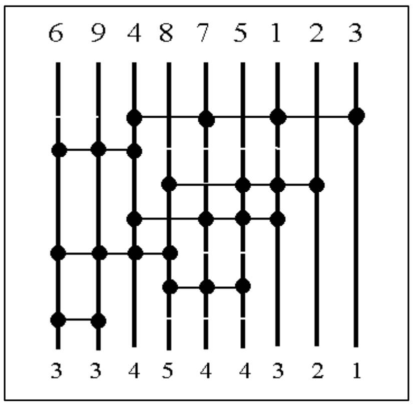

Fig. 3 - Permutated (optimal) solution

This paper describes the application of a *Genetic Algorithm (GA)* to problem *GMLP*. Traditional *GAs* work on a set of variables called structures. For applying them to optimization problems, the first step is the definition of a coding scheme that allows a one-to-one mapping between solutions and structures. A fitness function assigns a numeric value to each member of the current population (a collection of structures). The genetic operators used are selection (like tournament or biased roulette wheel) working together with a number of crossover and mutation operators. The best structure is kept after a predefined number of generations [3-5].

Holland put forward the "building block" hypothesis (schema formation and conservation) as a theoretical basis for the *GA* mechanism [4]. In his view, avoiding disruption of good schema is the basis for the good behavior of a *GA*. Schemata can be defined as parts of the structures, where some data is missing, and a major problem is that they are evaluated indirectly, via evaluation of their instances (structures). Goldberg and collaborators have addressed the problem of schemata evaluation and introduced the messy-GA [6-8]. In a different approach, Lorena and Furtado [9] proposed recently the *Constructive Genetic Algorithm (CGA)* as an alternative to a traditional *GA* approaches, particularly, for evaluating schemata directly [9].

The *CGA* evolves a population initially formed only by schemata, controlled by recombination, to a population of well-adapted structures (schemata instantiation) and schemata. It was applied to clustering problems [9], location problems and timetabling problems [10]. The *GMLP* is the first permutation like problem modeled and solved using the *CGA* approach.

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, through operators as selection, recombination and specific heuristics.

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

This paper is organized as follows. Section 2 presents the aspects of modeling for schema and structure representations and the consideration of the *GMLP* as a *bi-objective* optimization problem. 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 the modeling phase of the *CGA* is described. The *GMLP* is formulated as a bi-objective optimization problem. To attain the objective of evaluating schemata and structures in a common way, two fitness functions are defined on the space of all schemata and structures that can be obtained using a specific representation. An evolution process will be described in the next section. This process considers the two objectives on an adaptive rejection threshold, which gives ranks to individuals in population and yields a dynamic population.

Very simple structure and schema representations are implemented to the *GMLP*. A direct alphabet of symbols (natural numbers) represents the gate sequence and each gate is associated to a column of binary numbers, representing the connection request with each net. The 1 to J values are reserved to gate number and the symbol # is used to express indetermination (# - *do not care*) on schemata.

Fig.4 shows the representation for the *GMLP* instance of *Fig.1* 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.4 - a) Net-gate requirements matrix and structure s_i (Fig.2a); b) Example of permutated matrix and corresponding structure s_j ; c) Example of schema.

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

$$\begin{array}{ll} Min & \{g(s_k) - f(s_k)\} \\ Max & g(s_k) \\ \text{subject to} & g(s_k) \geq f(s_k) \\ & " \; s_k \in C \end{array} \qquad (2.1)$$

Function *g* is the fitness function that reflects the total cost of a given permutation of gates. Commonly, it is used in the formulation that considers track minimization as primary objective and wire length minimization as a secondary one. Therefore, it is defined as $g(s_k) = I.\ J.TR(s_k) + WL(s_k)$, or

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

where 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. It seems as these columns do not exist and the *b* matrix used to compute $g(s_k)$ contains only columns with information. In the example of *Fig. 3.c*, the track number is max {3,?,4,?,?,?,6,3,?} = 6 and the wire length is *sum*{3+0+4+0+0+0+6+3+0} - 7 = 9.

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:

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

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 in section III, subsection C.

Considering the definition of g , the maximization objective on *BOP* appears to be a contradiction, but it is needed at the constructive phase that gives distinct treatments to structures and schemata. By definition, f and g is applied to structures and schemata, just differing in the amount of information and consequently in the values associated to them. More information means larger values. In this way, the g maximization objective in *BOP* drives the constructive phase of the *CGA* aiming that schemata will be filled up to structures.

The optimal phase is conducted by the interval minimization $g - f$, which happens on structures near to a local *2-Opt* minimum. It is interesting to note that a direct *GMLP* objective of tracks and wire length minimization is indirectly reproduced at the *(g-f)* minimization.

III. THE EVOLUTION PROCESS

The *BOP* defined above is not directly considered as the set *X* is not completely known. Alternatively is considered 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*:

  a non-negative real number $g_{max} > max_{s \in X}\, g(s)$ that is an upper bound on the objective value;

- *interval minimization:*

  an interval length dg_{max}, obtained from g_{max} considering 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

$$g(s_k) - f(s_k) \geq d\, g_{\max} - a\,.\,d[g_{\max} - g(s_k)] \qquad (3.1)$$

presents a condition for rejection from the current population of a schema or structure s_k. The right hand side of *(3.1)* 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.

Expression *(3.1)* can be examined varying the value of a. For $a=0$, both schemata and structures are evaluated by the difference g-f (first objective of *BOP*). When a increases, schemata are most penalized than structures by the difference g_{max}-g (second objective of *BOP*).

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 *(3.1)*, thus yielding the following expression and corresponding rank to s_k:

$$a \geq \frac{dg_{\max} - [g(s_k) - f(s_k)]}{d[g_{\max} - g(s_k)]} = d(s_k). \qquad (3.2)$$

At the time they are created, structures and/or schemata receive their corresponding *rank value* $d(s_k)$. These ranks are compared with the current evolution parameter a. 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 *GMLP*, 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 that for each schema, a proportion of random positions receive a unique gate number. The remaining positions receive labels #. Along the generations, the population increases by addition of new offspring generated out of the combination of two schemata.

B. Selection

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* objective on the *BOP*). The selection of structures and/or schemata for recombination will be conducted to attain these two objectives, selecting for recombination schemata with small number of labels # and structures or schemata with small $d_k = \dfrac{g(s_k) - f(s_k)}{g(s_k)}$.

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, good 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.5* represents the base-guide selection process.

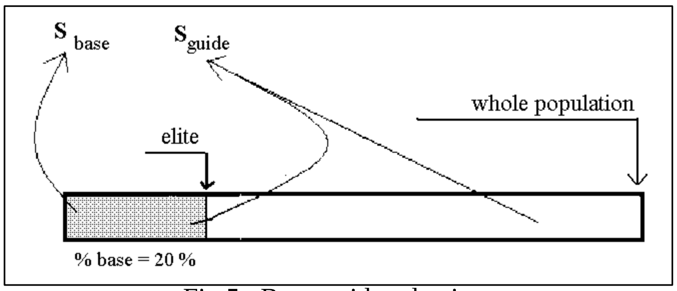

Fig.5 - Base-guide selection

C. Recombination

In the recombination operation, 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 the s_{guide} gene information to be used instead of s_{base} one.

D. The algorithm

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

```
CGA { Constructive Genetic Algorithm }
Given $g_{max}$ and $d$ ; $\alpha := 0$ ; $\varepsilon := 0.005$;          { initialization of parameters }
Initialize $P_\alpha$ ;                                                     { initial population }
For all $s_k \in P_\alpha$ 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 $P_\alpha$ ;                             { selection operator }
        Recombine Base and Guide                                            { recombination operator }
        Evaluate Offspring;                                                 { fg-fitness and ranking }
        Update Offspring in $P_\alpha$;                                     { update $P_a$ }
    end_while
    $\alpha := \alpha + \varepsilon$ ;
    For all $s_k \in P_\alpha$ satisfying $\alpha > d(s_k)$ do
        Eliminate $s_k$ from $P_\alpha$
    end_for
end_while
```

The ε increment is a linear step that increases the adaptive rejection threshold. Each distinct value of a corresponds to a generation. 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 at each generation. 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 *GMLP*, this feature was implemented as following:

```
{Unary-filling operator}
Select Base and Guide from $P_\alpha$ ;                  { selection operator }
If  Base is a schema 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 $P_a$  }
```

This code fragment replaces the innermost while loop of the *CGA* pseudo-code. 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. The principle is that a schema s_k must be filled with a good sequence of gate columns that do not belong to it yet.

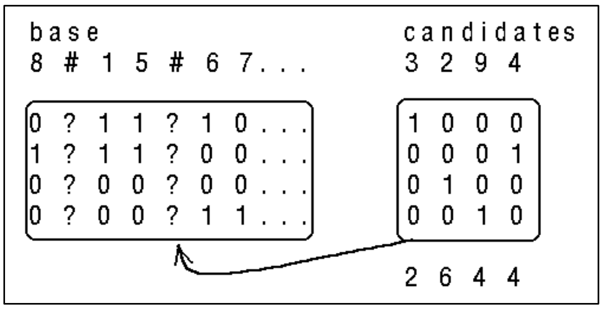

Fig.5 - Rule for filling schemata

In the example on *Fig.6*, 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, in matrix, the immediate neighbors gate columns to the left and to the right (if exists) of the # point. 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 (like the *f* definition on *expression 2.3*) avoids that the computational effort increases defining a constant number of neighbors to be inspected until the best is found. The neighbors are generated by all the 2-move changes in a constant length part of the structure. A position is chosen at random and starting from there an iterative process that inspects all possible 2-move changes in the structure.

The example of 2-move change is showed in *Fig.7*. The marks in positions of the structures mean reference points to be changed. Non-consecutive references cause the first change type, as showed in *Fig.7a*. Consecutive points cause the second change type in *Fig.7b*. Inspecting all or part of *2-Opt* neighborhood needs several moves (catching 2-to-2 points). The number of 2-move changes (neighborhood width) on each local search mutation is a *CGA* parameter of execution to be tuned. This and all the others settings will be described in next section.

The offspring is included in P_α if its rank value *d* is greater than the adaptive rejection threshold α and also it is not equal to any other individual in P_α.

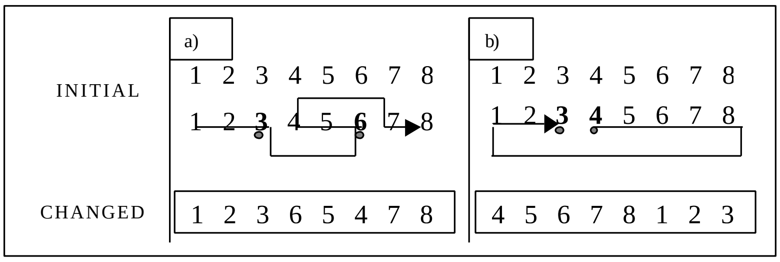

Fig.7 - Examples of one-move in 2-Opt neighborhood; a) non-consecutive reference points change; b) consecutive reference points change.

IV. COMPUTATIONAL RESULTS

The *CGA* for *GMLP* was coded in *ANSI C* and it was run on Intel Pentium II (266Mhz) hardware. For the computational tests, some *CGA* parameters were adjusted. The *d* parameter was set to 0.15 (usually between 0.10 and 0.20 values, for others applications [9]). This configures the interval $d.g_{max}$, establishing the survival time of each individual, once the expected *d* values are proportional to this interval. The *e* was set to 0.005 and also contributes to the survival time of each individual in P_a. In this case, it is set to slow the increase on a. These parameters avoid the premature termination with an empty population.

Each schema of the initial population received 50% of # genes (indetermination percentage), and 10% up to 20% of population were considered base individuals for base-guide selection, determining a small degree of diversification in selection process. In this case, 10% up to 20% of better individuals of the population are considered for the selection of individuals base.

Local search mutation rate was fixed in 100%, which means a constant improvement of individuals. The number of individuals initially generated was proportional to problem length (at least the number of gates). Other important parameter to be tuned is the neighborhood width (*nw*) to each local search mutation. After some simulations the better results set *nw* = 20. The ideal situation would be to use greater values for *nw*, but this would turn the mutation very slow.

The *CGA* was applied to several instances taken from the literature. Previous tests showed that *CGA* did not found any difficulties on small problems (smaller than v4090 problem), and the tests for some small problems are not reported in this work.

Table I presents the problem circuits considered, along with the corresponding number of nets and gates, a lower bound to the number of tracks, 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 the maximum column sum of the original matrix. Most of the best previous results comes of *Microcanonical Optimization* (*MCO*) approach [11], but other ones with same best results were included as in references: *GM_Learn* [12] and *GM_Plan* [13]. The *MCO* overcame (or match) all the other approaches and it was chosen for comparison with the results of *CGA*.

problem	gate	net	LB	best tracks	Refs
wli	10	11	4	4	[11,13]
wsn	25	17	4	8	[11,13]
v4000	17	10	5	5	[11]
v4050	16	13	5	5	[11,12]
v4090	27	23	9	10	[11,12]
v4470	47	37	5	9	[11,12]
x0	48	40	6	11	[11,12]
w1	21	18	4	4	[11,13]
w2	33	48	14	14	[11,13]
w3	70	84	11	18	[11]
w4	141	202	18	27	[11]

Table I - Chosen problems to compare

Table II presents the comparison among the best results of *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 results were obtained after 10 replications at same hardware environment. One replication means a complete execution over the same problem with the same parameters settings. For the *CGA*, the stop condition of one replication is when the best-known solution is reached or at a pre-defined number of generations with no improvement.

	MCO				CGA				
Prob	tracks:10 replications	tracks:1000 replications	one trial time (avg)	1000 trial time	Tracks:10 replications	wire lenght	Gen. (avg)	One trial time (avg)	10 trial time
wli	5.00	4.00	0.01	10.00	4.00	24	5.00	0.50	5.00
wsn	8.00	-	0.01	10.00	8.00	98	7.00	1.50	15.00
v4050	5.00	-	0.01	10.00	5.00	41	5.00	0.50	5.00
v4000	6.00	5.00	0.01	10.00	5.00	53	5.00	0.50	5.00
v4470	10.00	9.00	0.70	700.00	9.00	246	33.00	66.50	665.00
v4090	10.00	-	0.10	100.00	10.00	95	13.50	2.03	20.33
x0	11.00	11.00	0.70	700.00	11.00	303	92.57	75.56	755.60
w1	4.00	-	0.01	10.00	4.00	39	5.00	1.00	10.00
w2	14.00	-	0.40	400.00	14.00	235	19.50	18.50	185.00
w3	21.00	18.00	3.90	3900.00	18.00	677	186.00	306.25	3062.50
w4	32.00	27.00	61.70	61700.00	27.00	1730	225.00	5224.67	52246.67

Table II - CGA comparison to MCO

To compare different number of replications with respect to execution time, the total time of all the replications was computed based on average of one trial execution time. Thus, the total time of *CGA* experiments is 10 times the average of one trial execution. By its turn, the total time of *MCO* experiments is 1000 times the average one trial execution. The "-" execution time, reported in [10] as close to zero, to our computations, was turned to 0.01 seconds, that represents the smallest fraction of one second with two decimal digits.

The "-" symbol in the "tracks: 1000 replications" column means that *MCO* was not run 1000 times. According to the authors, for these problems, it was not necessary to improve the results obtained with 10 replications. The "gen" column is the number of generations (average) in 10 replications of *CGA*.

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

In [10], the authors remark the frequency of 36.3% to find the best result for *wli* problem (relatively small) in 1000 replications. No remarks are reported about larger problems. In our experiments, this percentage is 100% in 10 replications. In *CGA*, smaller percentages of best tracks were found only in larger problems (w3 and w4). There is no available data for a complete comparison of these two approaches, but, at least, *CGA* seems to be more robust than *MCO* in small problems, like the *wli* problem.

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 solution.

	CGA **5 best solutions**										**% best tracks**
wsn	104	8	104	8	105	8	107	8	113	8	100.00
wli	18	4	18	4	18	4	18	4	18	4	100.00
v4050	41	5	41	5	41	5	42	5	42	5	100.00
v4000	53	5	53	5	53	5	54	5	55	5	100.00
v4470	246	9	253	9	265	9	266	9	282	9	100.00
v4090	95	10	96	10	96	10	98	10	99	10	90.00
x0	303	11	304	11	305	11	306	11	308	11	80.00
w1	39	4	39	4	39	4	39	4	43	4	100.00
w2	235	14	236	14	267	14	273	14	275	14	100.00
w3	677	18	687	18	834	18	843	18	717	18	50.00
w4	1730	27	1836	27	1849	27	1745	28	1745	28	30.00

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

V. CONCLUSION

This work describes an application of the *Constructive Genetic Algorithm* (*CGA*) to *Gate Matrix Layout problems (GMLP).* The *CGA* adapted to work with *GMLP* presents some new specific features, like the filling operator, and a *2-Opt* heuristic used as mutation and on definition of the two fitness functions (*f* and *g*). Regarding the computational tests, the *CGA* reached all the best-known results (number of tracks) for instances taken from the literature, but it appears to be more robust than the *Microcanonical Optimization* approach. There is no published information about wire length in literature to completely compare the approaches. Applications of *CGA* to other classes of permutation problems are foreseen for future works.

ACKNOWLEDGMENTS

The first author acknowledges the *Programa Institucional de Capacitação Docente e Técnica* - PICDT/CAPES for financial support. The second author acknowledges *Conselho Nacional de Desenvolvimento Científico e Tecnológico* - CNPq (proc. 300837/89-5) and *Fundação para o Amparo a*

Pesquisa no Estado de S. Paulo - FAPESP (proc. 99/06954-7) for partial financial support. The authors acknowledge the editors and the useful comments and suggestions of the three referees.

REFERENCES

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

2 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.

3 Goldberg, D.E., Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA, p. 11-172, 1989.

4 Holland, J.H., Adaptation in natural and artificial systems. MIT Press, p. 11-147, 1975.

5 Michalewicz, Z., Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin, 1996.

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

7 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.

8 Kargupta, H. Search, Polynomial Complexity, and The Fast Messy Genetic Algorithm, Ph.D. thesis, IlliGAL Report No. 95008, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, 1995.

9 Lorena, L. A. N. and Furtado, J. C. Constructive Genetic Algorithm for Clustering Problems. Evolutionary Computation 9(3): 309-327, 2001.

10 Ribeiro Filho, G. and Lorena, L. A. N. A Constructive Evolutionary Approach to School Timetabling. In Applications of Evolutionary Computing, Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tijink, H., (Eds.) - Springer Lecture Notes in Computer Science vol. 2037, pp. 130-139 - 2001

11 Linhares, A., Yanasse H. and Torreão J. R. A. *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.

12 Chen S. J. and Hu. Y. H. *GM_Learn: an interactive learning algorithm for CMOS gate matrix layout*. IEE Proc E, vol 137, pp 301-309, 1990.

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