JOÃO VITOR MASCARENHAS DOS SANTOS

Orientador: Marco Antonio Moreira de Carvalho

MÉTODOS EVOLUCIONÁRIOS APLICADOS À PRODUÇÃO EM MICROELETRÔNICA

Ouro Preto
Fevereiro de 2017

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

MÉTODOS EVOLUCIONÁRIOS APLICADOS À PRODUÇÃO EM MICROELETRÔNICA

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

JOÃO VITOR MASCARENHAS DOS SANTOS

Ouro Preto
Fevereiro de 2017

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas
Graduate Program in Computer Science

EVOLUTIONARY METHODS APPLIED TO MICROELECTRONICS PRODUCTION

Monograph presented to the Graduate Program in Computer Science of the Universidade Federal de Ouro Preto in partial fulfillment of the requirements for the degree of in Computer Science.

JOÃO VITOR MASCARENHAS DOS SANTOS

Ouro Preto
February 2017

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

Métodos Evolucionários Aplicados à Produção em Microeletrônica

JOÃO VITOR MASCARENHAS DOS SANTOS

Monografia defendida e aprovada pela banca examinadora constituída por:

Ouro Preto, Fevereiro de 2017

Resumo

Este trabalho apresenta dois métodos evolucionários aplicados à solução do Problema de Determinação do Leiaute de Matrizes de Portas (ou GMLP, do inglês *Gate Matrix Layout Problem*), um problema de aplicação prática na indústria microeletrônica. No contexto de projeto de circuitos VLSI (*Very Large Scale Integration*), este problema pede por uma permutação das colunas de uma matriz de portas, um dispositivo lógico usado comumente para este fim, tal que o número exigido de trilhas para implementação dos circuitos – e consequentemente a área – sejam minimizados. Os métodos propostos são um Algoritmo Genético Híbrido (HGA) e a implementaçãs do algoritmo de Otimização do Enxame de Partículas (PSO, do inglês *Particle Swarm Optimization*), em uma versão específica para o problema. Experimentos computacionais compararam o desempenho dos dois métodos utilizando instâncias reais e artificiais disponíveis na literatura.

Abstract

This work presents two evolutionary methods applied to the solution of the Gate Matrix Layout Problem (GMLP), a problem with practical application in the microelectronics industry. In the context of Very Large Scale Integration (VLSI) design, this problem asks for a permutation of columns of a gate matrix, a logic device commonly used to this end, such that the number of required tracks to implement the corresponding integrated circuit, and thus the area, are minimized. The proposed methods are a Hybrid Genetic Algorithm (HGA) and a implementation of the Particle Swarm Optimization algorithm (PSO), tailored to the problem characteristics. Computational experiments compared the performance of the two methods considering real-world and artificial instances available in the literature.

Dedico este trabalho aos meus familiares e amigos.

Agradecimentos

Agradeço a todas as pessoas que me ajudaram na realização deste trabalho. Meu orientador, Prof. Marco Antonio, pelo apoio e orientação. Ao Prof. Scott Gordon pela co-orientação e a todos os meus professores, amigos e familiares.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Nowadays, microelectronics is present in a wide range of fields and is fundamental in areas such as computer science, telecommunications, industrial processes control, banking and commercial services automation, and also in the consumer goods industry, with practical applications in engineering and industry.

In the process to create components for this subfield of electronics, Very Large Scale Integration (VLSI) circuits are used. These circuits are designed using either programmable logical arrays (PLAs) or programmable logical matrices (PLMs). In order to improve the production of these components, a computational tool to determine their optimized layout is necessary to produce these circuits cheaper, faster and smaller. The circuit total area is the main factor to minimize the production cost, as they are produced on silicon disks, usually called wafers, where the circuit cost is inversely proportional to the number of circuits produced on a single wafer. Hence, the smaller the circuit size is, the larger is the number of circuits produced on a single wafer, thus, making it cheaper.

Figure 1.1 shows an example of a gate matrix layout, where six gates named from P_1 to P_6 are connected by a wire to implement the *nets*, named R_1 to R_5. Each net must be physically allocated on the circuit not overlapping others, resulting in circuit *tracks*. However, nonovelapping nets may share the same track. The Gate Matrix Layout Problem (GMLP) consists of determining a permutation of the gate matrix such that the number of tracks needed to implement all the nets is minimized. Consequently, the circuit area is also minimized. As shown in Figure 1.1, five tracks, one for each net, are required to accommodate all nets of the circuit.

In this work, it is proposed the application of two bio-inspired evolutionary methods: the Particle Swarm Optimization (PSO) – that simulates the emergent behavior of flocks of birds – and a Hybrid Genetic Algorithm (HGA) – that simulates natural reproduction and selection.

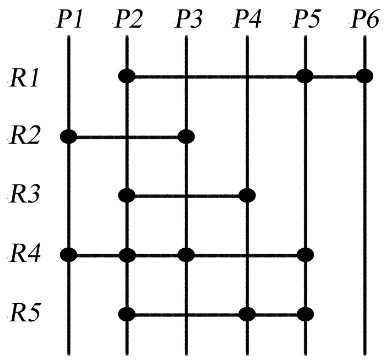

Figure 1.1: Example of a gate matrix layout

1.1 Motivation

The GMLP was proved NP-hard by a reduction of the *Interval Graph Augmentation* Problem Kashiwabara and Fujisawa (1979) (apud Linhares and Yanasse, 2002) and has applications in different areas and problems of equivalent formulation, according to Linhares and Yanasse (2002): *Modified Cutwidth*, *Minimization of Open Stacks Problem* (MOSP), *Programmable Logic Array Folding* (*PLA Folding*), *Interval Thickness*, *Node Search Game*, *Edge Search Game*, *Narrowness*, *Split Bandwidth*, *Graph Pathwidth*, *Edge Separation*, and *Vertex Separation.*

Recently, interest in the practical application of this problem has grown. With the installation in Minas Gerais of the first semiconductor factory in the southern hemisphere, this work has the potential to generate innovation applicable to this new industrial niche. Additionally, there is no record of the PSO applied to the GMLP or equivalent problems, which represents a new approach to this kind of problems.

1.2 Goals

The main goal of this work is to construct a tool that through computational intelligence methods, such as metaheuristic methods, can determine the layout of a gate matrix with solutions close to the optimum in a short period of time. The specific goals are presented below.

- To build a theoretical foundation on the problem;
- To study evolutionary methods and formulate their application to the GMLP solution;
- To implement metaheuristic methods to solve the GMLP;

- To perform extensive computational experiments, considering artificial and real-world instances;
- To present an optimization model that matches the characteristics of different applications of the problem using the literature models.

1.3 Work Organization

This work is organized as follows. A brief review of the literature is made in Chapter 2. In Chapter 3, the basis of the problem and the implemented methods are described in detail, each step being illustrated. Chapter 4 presents the methodology. Extensive computational experiments that took into consideration 220 real instances from the industry and artificial instances from the literature are reported in Chapter 5. Finally, in Chapter 6, conclusions about this research work are drawn and future work is indicated.

Chapter 2

Literature Review

Heuristic methods have been applied to GMLP since the publication of Wing et al. (1985), in which two real circuits were modeled using interval graphs, considering common gates in different tracks as an intersection. Subsequently, Hwang et al. (1987) achieved better results with the modified *min-net-cut* algorithm whose time complexity is $O(nlogn)$. The same algorithm was used again by Chen and Hou (1988) to achieve better results than the previous ones of the literature in the five circuits considered. In order to minimize the number of tracks needed to implement one-dimensional logic arrays, Hong et al. (1989) developed a constructive heuristic and implemented a *Simulated Annealing* metaheuristic. For the seven instances considered, the constructive heuristic achieved the same results as the *Simulated Annealing* in five of them and improved one solution. In only one case the constructive heuristic had a worse result. However, the proposed heuristic method was up to 50 times faster the *Simulated Annealing.*

On the following decade, the works of Chen and Hu (1990) and Hu and Chen (1990) obtained, or determined, the best results for all the available instances on the literature up to that moment. Two algorithms were presented, *GM-Plan* and *GM-Learn*, both based on artificial intelligence paradigms.

At the end of 1980s and a good part of 1990s, the GMLP was predominantly approached by metaheuristic techniques and hybrid methods. Shahookar et al. (1994) proposed an implementation of *Genetic Algorithm* and *Beam Search*, named *Genetic Beam Search.* The algorithm was tested in three sets of benchmark instances and achieved better results compared with the *Simulated Annealing* used by Hong et al. (1989). With the *Predatory Search* strategy, Linhares (1999) achieved the same results obtained by the *GM-Plan* and *GM-Learn.* In addition, in 12 of the 25 instances, the strategy was able to match the lower bound to the solution, and in 4 instances it matched the best results of the literature. Linhares et al. (1999) presented the *Microcanonical Optimization*, a process derived from the statistical physics, as well as the Simulated Annealing. This method was able to match all previous results from Chen and Hu (1990) and Hu and Chen (1990), but it did not reach the results of the *Predatory Search.*

Constructive Genetic Algorithms were used by Oliveira and Lorena (2002), reaching all the best-known results of the literature, previously obtained by the Microcanonical Optimization. According to the authors, this constructive genetic algorithm is more robust than the Microcanonical Optimization. An evolutionary approach of multiple populations was introduced by Mendes and Linhares (2004), aggregating genetic and memetic algorithms. This work presented similar results to those of Microcanonical Optimization, surpassing all previous metaheuristic and heuristic approaches in the literature – including *Simulated Annealing*, *GM-Plan*, *GM-Learn*, and constructive heuristics – both in quality of the obtained solutions and in efficiency in terms of convergence of the algorithm. The faster convergence is justified by the addition of a heuristic local search that provided a reduction of the search space, considering only the bottleneck of the problem, which was named *critical columns*.

Exact methods applied to the GMLP are found on the literature with lower frequency. Some works such as the formulation with dynamic programming presented by Deo et al. (1987) can be highlighted. However, computational experiments to measure the quality of the formulation were not made. In this same work, the authors prove that there is no absolute approximation algorithm for the GMLP.

The problem of minimizing the wire length used on the nets, with restrictions on the number of tracks, was considered by De Giovanni et al. (2013). A genetic algorithm and a *Branch-and-Cut* procedure were proposed and tested in more than three hundred instances from the literature. Specifically, the genetic algorithm was used to set an upper bound for the number of tracks needed and the branch-and-cut was used to minimize the wire length. For some instances, bad results were obtained with a large gap or it was not even possible to solve the linear relaxation in one hour.

Recently, Gonçalves et al. (2016) applied a *Biased Random-Key Genetic Algorithm* (BRKGA) to the MOSP solution. The authors made comprehensive experiments with the algorithm in more than six thousand MOSP and GMLP instances, since these problems are equivalent. In order to verify the quality of their approach, they used instances from Smith and Gent (2005), Faggioli and Bentivoglio (1998), Hu and Chen (1990), and Sheet Cutting and Process Optimization for Furniture Industry (SCOOP) consortium[1]. The reported results show the method was able to achieve equal or better results than other approaches from the literature. Specifically, the method was able to match all known optimal solution values, although not all solutions are known. Thus, this work is the current the state-of-the-art concerning the GMLP.

[1]Available at: `www.scoop-project.net`

Chapter 3

Theoretical Foundation

In this chapter, we present the definitions and foundations of the studied problem as well as the techniques used from the literature. A formal definition of the GMLP is given, followed by descriptions of the preprocessing operations, problem representation adopted, the graph search and the method for sequecing gates. Finally, the foundations of Genetic Algorithm and Particle Swarm Optimization are given.

3.1 The Gate Matrix Layout Problem

A gate matrix is a logical device that has two dimensions of gates, one AND and another OR, both programmable. Connections between these gates can be made to achieve specific results. Thus, the gate matrix is used to design combinatorial logic circuits composed of n logical gates and m nets – a subset of gates connected by a wire. If different nets use the same gate, each net must be allocated on a different track, thus, ensuring that nets do not overlap on the printed circuit, because it would change the logic of the circuit.

Figure 3.1 (a) shows a gate matrix with six gates enumerated from p_1 to p_6, represented as vertical lines, columns, and six nets enumerated from r_1 to r_6, represented by sets of horizontal points. Following, Figure 3.1 (b) shows the connections between the gates of each net, where, in order to connect one gate to another, it may be necessary to cross other gates, without connecting them. A different layout of the gate matrix is presented in Figure 3.1 (c), in which it is possible to notice that different nets can be allocated at the same track. For example, the nets r_3 and r_4 are allocated on the same track, as well as r_6 and r_5.

As described by De Giovanni et al. (2013), an instance of the GMLP can be defined by a matrix $M \in \{0, 1\}\ m \times n$. The lines of the matrix M are associated to m nets and the columns are associated to n gates, such that $M(i, j) = 1$ if and only if net i includes gate j and $M(i, j) = 0$ otherwise. A solution to the GMLP is a sequence $\phi : [1, ..., n] \rightarrow [1, ..., n]$, where ϕ_j indicates the position of gate j in the gate matrix. This solution defines a new matrix M_ϕ, obtained by a permutation of the columns of M according to ϕ. From the matrix M_ϕ, which

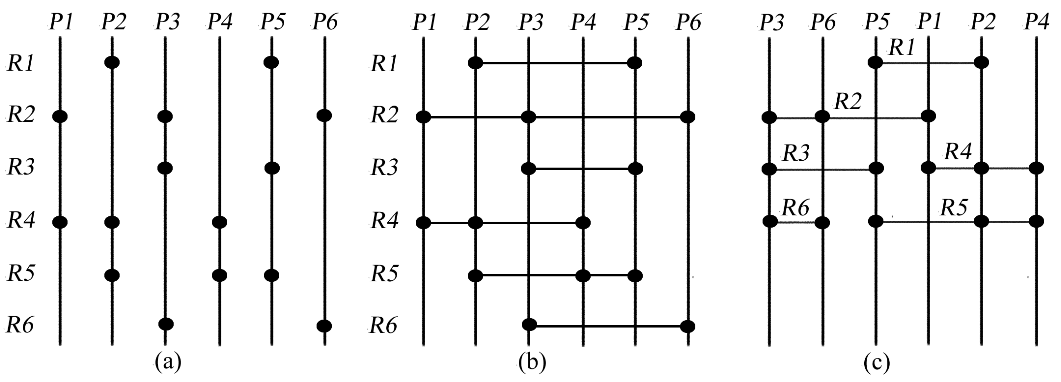

Figure 3.1: Gate Matrix (a), with highlighted nets (b), and with a different gate permutation (c).

holds the consecutive 1s property, we obtain a new matrix $\bar{M}_\phi$ such that $\bar{M}_\phi(i,j) = 1$ if and only if, according to ϕ, the net i includes or crosses the gate j. Equation 3.1 shows the formal representation of matrix $\bar{M}_\phi$.

$$\bar{M}_\phi(i,j) = \begin{cases} 1 & \text{if } \exists x, \exists y \mid \phi(x) \le j \le \phi(y) \text{ and } M_{ix} = M_{iy} = 1, \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Therefore, given a matrix $M \in \{0,1\}\ m \times n$, the goal of the GMLP is to determine a permutation of its columns that results in a corresponding matrix $\bar{M}_\phi$ such that the largest sum of elements in any column is minimized, which is equivalent to minimize the maximum number of tracks required to implement the corresponding circuit. The columns of maximum sum are also called *bottlenecks* or *critical columns.* Equation 3.2 shows GMLP objective function, where Φ is set of all possible permutations.

$$\min_{\phi \in \Phi} Z^{\phi}_{GMLP}(M) \tag{3.2}$$

Table 3.1 (a) shows matrix M that represents the circuit shown previously in Figure 3.1 (a).

Table 3.1: Example of an instance M (a), M_ϕ (b), and $\bar{M}_\phi$ (c).

	p_1	p_2	p_3	p_4	p_5	p_6
1	0	1	0	0	1	0
2	1	0	1	0	0	1
3	0	0	1	0	1	0
4	1	1	0	1	0	0
5	0	1	0	1	1	0
6	0	0	1	0	0	1

(a)

	p_3	p_6	p_5	p_1	p_2	p_4
1	0	0	1	0	1	0
2	1	1	0	1	0	0
3	1	0	1	0	0	0
4	0	0	0	1	1	1
5	0	0	1	0	1	1
6	1	1	0	0	0	0

(b)

	p_3	p_6	p_5	p_1	p_2	p_4
1	0	0	1	1	1	0
2	1	1	1	1	0	0
3	1	1	1	0	0	0
4	0	0	0	1	1	1
5	0	0	1	1	1	1
6	1	1	0	0	0	0

(c)

Matrix M_ϕ in Table 3.1 (b) represents the circuit with permuted gates where, after applying the consecutive 1s property, we obtain the new matrix $\bar{M}_\phi$, as shown in 3.1 (c). Each position between the first and the last nonzero entry of each net also receive the value 1 on matrix $\bar{M}_\phi$. Thus, determining the cost of the solution means finding the column with the largest sum. The importance of the consecutive 1s property is due to the need to represent a wire crossing a gate, even if it is not part of the net. For example, in the circuit of Figure 3.1 (c), the wire of r_1 crosses the gate p_1 and, consequently, in the matrix of Table 3.1 (c) the position $\bar{M}_\phi(1, 4)$ has a value of 1. This representation also helps to avoid overlapping of wires.

3.1.1 Graph Representation

According to Yanasse (1997b), in the representation of GMLP by graphs, nodes represent the nets and the edges indicate that two nets share some common gate. As an example, in the circuit of Figure 3.1 (a), nets r_2 and r_4 share gate p_1, therefore, in the graph there is an edge between nodes 2 and 4, as it can be observed in Figure 3.2.

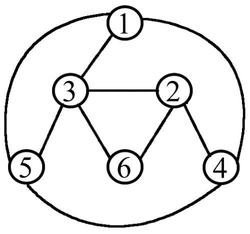

Figure 3.2: A representation of the problem in graph.

3.1.2 Preprocessing Operations

In order to reduce the dimensions of the problem, without causing any damage to the quality of the final solution, two different preprocessing methods introduced by Yanasse and Senne (2010) are applied to the instances.

3.1.2.1 Dominance Among Gates

According to Yanasse and Senne (2010), a gate p_i dominates another gate p_j, if and only if the set of nets that uses gate p_j is a subset of the nets that uses gate p_i. Dominated gates can be disregarded during the problem solving and sequenced in the final solution, right after the gates that dominate them, without interfering on the solution cost, since both of them can use the same tracks.

As an example, in the circuit of Figure 3.3 (a), p_2 is used by the set of nets $\{r_1, r_4, r_5\}$, p_4 by $\{r_4, r_5\}$, p_3 by the set $\{r_2, r_3, r_6\}$, and p_6 by $\{r_2, r_6\}$. Therefore, $p_4 \subset p_2$ and $p_6 \subset p_3$.

Thus, p_2 dominates p_4 and p_3 dominates p_6. The original problem has 6 gates, however, p_2 and p_4 can be considered as a single gate, as well as p_3 and p_6. Hence, the reduced problem, shown in Figure 3.3 (b), has only 4 gates. To check the dominance among gates is required

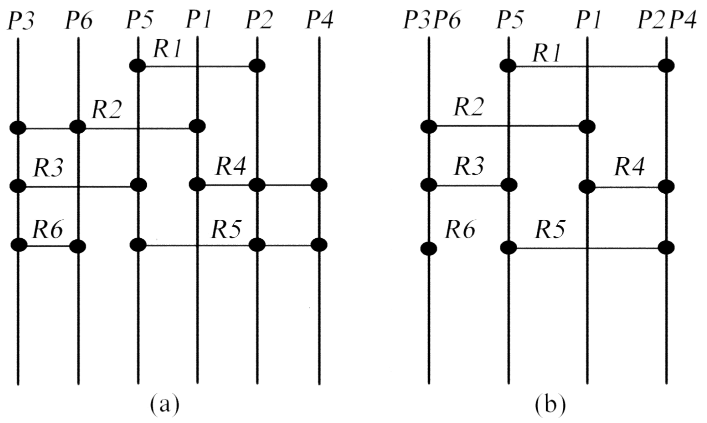

Figure 3.3: A representation of a gate matrix (a), and its reduced representation (b).

to compare the sets of nets that contain them. Consequently, the complexity of this process is limited by $O(n^2m)$.

3.1.2.2 Elimination of Twin Nodes

Again, according to Yanasse and Senne (2010), considering the graph representation of the GMLP, if two adjacent nodes share the same set of adjacent nodes, they are considered as twin nodes. Such nodes can be considered as one, reducing the size of the problem with no loss for the solution value, since the respective nets are similar.

Figure 3.2 shows the graph representation of the circuit from Figure 3.3 (a). Nodes 1 and 5 are adjacent to nodes 3 and 4, and are also adjacent to each other. These are twin nodes and, therefore, can be considered as a single node, reducing the size of the graph, as shown in Figure 3.4. The detection of twin nodes requires examining and comparing the neighborhood

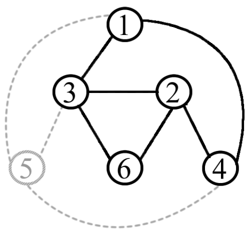

Figure 3.4: A representation of the reduced graph where nodes 1 and 5 are twins.

of all pairs of nodes, thus, its complexity is bounded by $O(m^3)$.

3.1.3 Breadth-First Search Applied to the Graph Representation

An effective strategy to solve the GMLP and equivalent problems is to sequence the gates considering the relationship among the nets, as established by Yanasse (1997a). Based on this principle, the proposed heuristic generates an initial solution that relates nets that have some similarities. Subsequently, this heuristic reduces the number of required tracks on the printed circuit, guiding a search in the graph using a greedy criterion.

For this purpose, as introduced by Carvalho and Soma (2015), a Breadth-First Search (BFS) is performed in the graph. In order to sequence the nets with fewer gates firstly and preferentially contiguously, the used BFS chooses the next node to be explored according to the number of gates the net has. Thus, the BFS initiates the exploration of the graph from the vertex corresponding to the net with the lowest number of gates and uses this same greedy criterion to determine the order in which the other nodes of the neighborhood will be explored. At the end of the BFS, a sequence S is obtained with the order of exploration of the nodes.

Figure 3.5 (a) shows the graph that corresponds to the circuit of Figure 3.1 (a), with twin nodes 1 and 5 joined by the preprocessing and with the respective gate numbers represented inside the nodes. Note that, since gates p_4 and p_6 were dominated, the number of gates of nets r_2, r_4 and r_6 was decremented because they use some of the dominated gates. Starting from the node with the lowest number of gates, node 6, Figure 3.5 (b) exemplifies the first iteration of the BFS with the exploration of node 6 and its neighborhood. As the nodes are visited, they are inserted into the S queue, which at the end, Figure 3.5 (d), produces the solution for this step of the algorithm.

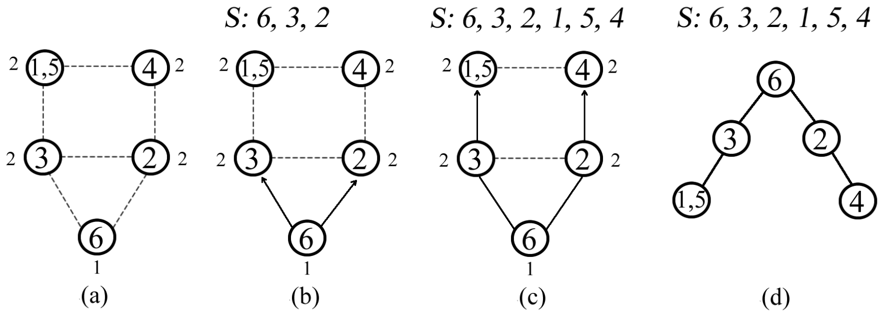

Figure 3.5: Application of the proposed BFS to the circuit of Figure 3.1.

Clearly, the rule used to guide the search does not necessarily lead to optimal solutions. In fact, the breadth-first search and the greedy criterion of preferentially exploiting nets with fewer gates proposed by Carvalho and Soma (2015) constitute a heuristic, which, according to the reported results, produces good solutions for the GMLP.

The proposed breadth-first search requires that the neighborhood of each node is non-decreasing ordered according to the number of gates per net. Considering the representation of the graph by adjacency lists, the complexity of this algorithm is limited by $O(m^2 log\ m)$.

3.1.3.1 Sequencing of Gates

The sequence of nets generated by the BFS is not a solution for the GMLP. In order to obtain the sequence of gates required to solve the problem, the method presented by Becceneri et al. (2004) was used. The principle of this method is to maintain the relationship established in the sequence of nets. Thus, gates not related to certain nets are not sequenced prematurely, which would require a greater number of tracks in the solution.

This method reads the net sequence previously generated by the BFS and examines it. For each new net analyzed, it is searched among the gates not sequenced yet which ones are used by the nets already examined up to that moment. Then, those gates are included in the solution. In this step, the dominated gates are considered again in order to obtain a complete solution to the problem: they are sequenced after the gates that dominate them.

Considering the exploration sequence obtained by the BFS, S = [6, 3, 2, 1, 5, 4], the gate sequencing initially considers gates, p_1, p_2, p_3 and p_5 as the others are dominated. When the set of nodes {6, 3, 2} is examined, gate p_3 can be sequenced, followed by the dominated gate p_6, as shown in Figure 3.6 (a). Subsequently, in Figure 3.6 (b), with the examination of the next nodes 1 and 5, gate p_5 is sequenced. Finally, with the addition of node 4 to the set of examined nodes, gates p_1 and p_2 can finally be sequenced, followed by the dominated gate p_4, see Figure 3.6 (c). Thus, a sequence of gates ϕ: [3, 6, 5, 1, 2, 4] is obtained, that is, a solution to the problem, as shown in the circuit of Figure 3.1 (c).

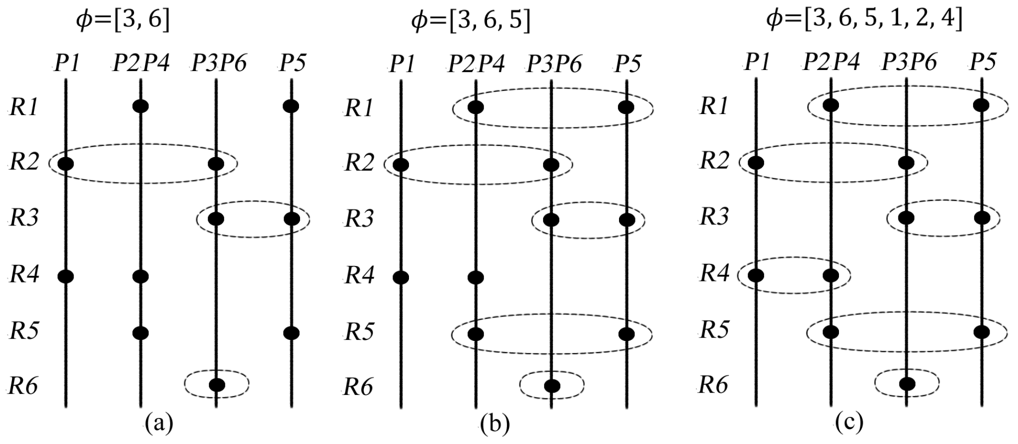

Figure 3.6: Gate sequencing. The dotted ellipses indicate the nets already examined.

This method for gate sequencing requires the comparison of the set of nets already examined and the set of nets that contain each gate, requiring time complexity bounded by $O(mn)$.

3.2 Genetic Algorithms

Since its creation and popularization with the work of Holland (1975), the Genetic Algorithm (GA) metaheuristic is being broadly used to solve a variety of optimization problems. Its inspiration comes from the reproduction of beings and the nature strategies to evaluate and create strongest and improved versions of animals on the planet. The standard genetic algorithm represents candidate solutions as a population of individuals encoded, for example, on a binary string, like the DNA, and a fitness value associated with it, like the ability to survive and to adapt. As the algorithm runs, individuals pass through phases of selection, reproduction with crossover and mutation, and then the population is renewed. Figure 3.7 illustrates the general form of the algorithm with the steps mentioned.

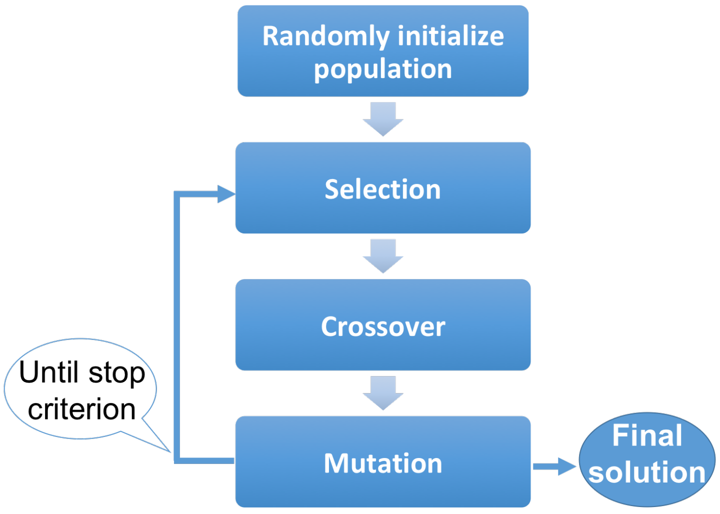

Figure 3.7: Simple Genetic Algorithm flowchart.

The approach proposed by Holland (1975) is usually called Simple Genetic Algorithm (SGA), and represent the most canonical form of the technique. The individuals from the binary encoded population are selected proportionally according to its fitness. Pairs of individuals from this selection are randomly chosen and a one-point crossover applied to them, resulting in a new pair of individuals. A bit-flip mutation is applied on the offspring, having a probability to occur in each bit of all them. This resulting population replaces the previous one and the process repeats until a stop criterion.

Algorithm 1 presents an overview of the SGA. The population P is randomly initialized in line 1 and updated on lines 2-7 on a loop through selection of parents (line 3), crossover (line 4), mutation of offspring (line 5) and the actual update of population on line 6.

Algorithm 1: Simple Genetic Algorithm pseudocode.

```
1 $P \leftarrow InitializePopulation()$;
2 repeat
3     $P_{selected} \leftarrow Selection(P)$;
4     $Offspring \leftarrow OnePointCrossover(P_{selected})$;
5     $Mutation(Offspring)$;
6     $P \leftarrow Offspring$;
7 until stop criterion is false;
```

3.3 Particle Swarm Optimization

Swarm intelligence algorithms have been used in order to solve non-linear continuous optimization problems since 1980's. This bioinspired approach creates artificial individuals, the so called "boids" ("bird-oid object" or "bird-like object"), to simulate flocks of birds. With simpler and fewer computation, these "boids" search on solution space of the problem alternating between exploration and exploitation. In Kennedy and Eberhart (1995), the authors introduced the Particle Swarm Optimization (PSO), whose name was chose particle to not define a specific specie of flock, such as birds or bees. The popularity of this algorithm has grown due to its application in both scientific and engineering field, also because it is simple to implement and adapt to a optimization problems.

In the PSO, like in the Genetic Algorithm, each individual (*particle*) of a population (*swarm*) is a candidate solution for the problem and has a *fitness* (objective value) associated to it. The self-organization characteristic of flocks that inspired this approach emerges due to principles of autonomy within a large number of individuals on a swarm, which is guided by simple rules to achieve a common goal. A particle knows its *position* on the search space and needs a *direction* to follow, thus, a *velocity* is associated to each one in order to guide them. This velocity is updated according to the previous best solution achieved by the particle and the current global best solution of the swarm. Figure 3.8 presents a flowchart that represents the main idea of the PSO strategy.

Equation 3.3, proposed by Kennedy (1997) and Kennedy et al. (2001), shows that each particle i will have its position x_i at time t updated according to its previous position $x_i(t-1)$ and to the new velocity $v_i(t)$. The velocity $v_i(t)$ is defined upon three components: the current velocity $v_i(t-1)$, the previous best solution found by the particle, p_i, and the global best solution of the swarm p_g. The constants ϕ_1 and ϕ_2 are stochastic parameters defined by an upper limit. They represent, respectively, the exploration (tendency to follow its own previous best) and exploitation (tendency to follow global best) characteristics of the particle.

$$\begin{cases} \vec{v}_i(t) = & \vec{v}_i(t-1) + \phi_1(\vec{p}_i - \vec{x}_i(t-1)) + \phi_2(\vec{p}_g - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = & \vec{x}_i(t-1) + \vec{v}_i(t) \end{cases} \tag{3.3}$$

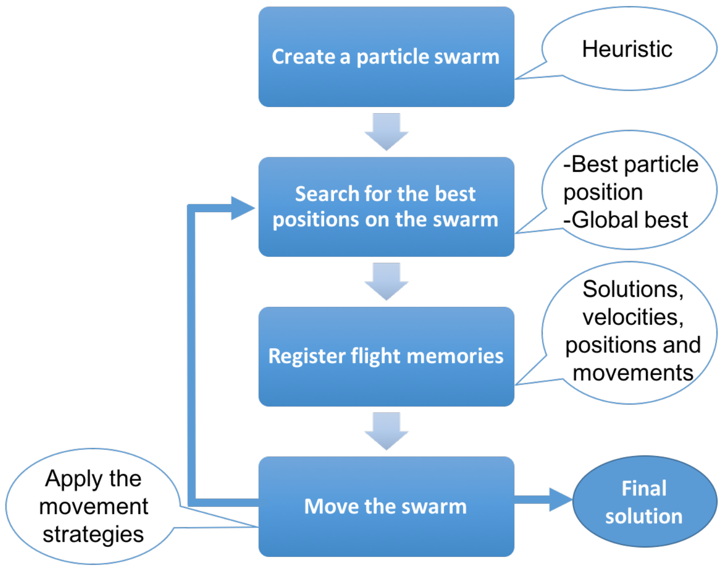

Figure 3.8: Particle Swarm strategy flowchart. Adapted from Goldbarg et al. (2015).

At each step of the algorithm, the particle velocity assumes a new value according to its past results and the best one of the swarm. Then, the position is individually updated according to this new velocity. Figure 3.9 illustrates the composition of movements to define the new position of particle i at time $t + 1$.

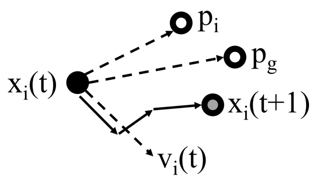

Figure 3.9: The composition to define a new position. Adapted from Goldbarg et al. (2015).

A general implementation of the metaheuristic is presented in Algorithm 2. Given a randomly initialized swarm of particles P, the algorithm will loop until a stop criterion is met (lines 1 - 15). In each step, the particles are evaluated (line 3), and $pbest_i$ and $gbest$ are updated according to the particles fitness value (lines 4 - 9). Finally, the swarm is analyzed again to have its velocity and current position updated (lines 11 - 14).

In order to avoid a high diversity of velocities on the swarm, which can cause overflow, compromising the convergence of the algorithm, a maximum velocity v_{max} is defined to limit the swarm velocities. Equation 3.4 demonstrates how this limit is implemented, as defined by Kennedy (1997).

$$v_i = \begin{cases} v_{max} & \text{if } v_i > v_{max} \\ -v_{max} & \text{if } v_i < -v_{max} \end{cases} \tag{3.4}$$

Algorithm 2: Basic PSO pseudocode.

```
Data: particle swarm $P$
1  while $stop\ criterion\ is\ false$ do
2      foreach $each\ particle\ i \in P$ do
3          evaluate($x_i$);
4          if $fitness(x_i) < pbest_i$ then
5              $pbest_i \leftarrow x_i$;
6          end
7          if $fitness(x_i) < gbest$ then
8              $gbest \leftarrow x_i$;
9          end
10     end
11     foreach $particle\ i \in P$ do
12         updateVelocity($v_i$);
13         updatePosition($x_i$);
14     end
15 end
```

3.4 Particle Swarm Optimization for Discrete Optimization

Originally, the PSO was proposed to optimize continuous non-linear problems. However, adaptations on the approach were made in order to apply the same metaheuristic to discrete problems, such as the Traveling Salesman Problem in Clerc (2004) and other permutational problems, such as the GMLP. The main difference of this adaptation relies on how the operations with the positions and velocity will happen. Goldbarg et al. (2015) proposed special definitions for four operations, based on Clerc (2004), that changes the position and velocity of the particles: movement, addition, multiplication and subtraction. The particle *position* is a solution to the problem addressed, a sequence of elements $x = [3, 5, 6, 1, 2, 4]$ for example, and the *velocity* can be defined according to the problem specifications, for example swapping the position of elements $v = (1, 3)$, where the elements on positions 1 and 3 will be swapped.

Movement Represented by the operator $\circ$, the movement operation applies a given velocity on a position and results in a new position. Figure 3.10 gives a set of elements of a permutation problem (a), an initial position x_1 (b) and the application of the velocity v_1 on the position x_1 resulting on x_2 (c), where the elements on positions 1 and 4 are swapped with elements on positions 3 and 6, respectively. This operation can be represented with a pseudovector notation: $\vec{x_2} \leftarrow \vec{x_1} \circ \vec{v_1}$.

Addition The addition, operator $\oplus$, considers two arbitrary velocities v_1 and v_2, for example, and results in a third one, v_3. It is defined as the transposition list with the elements of v_1 followed by those of v_2. Each element is analyzed and the resulting list can be reduced because of equivalent operations. Equation 3.5 and 3.6 show two addition operations

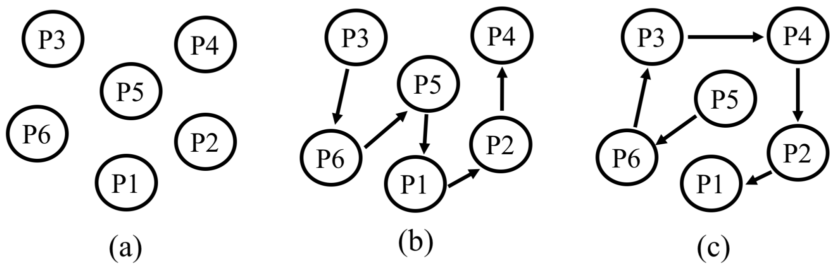

Figure 3.10: A set of elements (a), a possible position $x_1 = [3, 6, 5, 1, 2, 4]$ (b), and the application of $v_1 = ((1, 3), (4, 6))$ on x_1 resulting in $x_2 = [5, 6, 3, 4, 2, 1]$ (c).

examples. In Equation 3.5 is presented a simple addition where the result is the two lists concatenated, since no operation is equivalent to other. The example on Equation 3.6 illustrates a resulting empty list, which is equivalent to applying v_1 and then v_2, since one velocity undo what the other has done. The pseudovector representation is $\vec{v_3} \leftarrow \vec{v_1} \oplus \vec{v_2}$.

$$\begin{aligned} v_1 =&((1,2),(3,4),(1,3)) \\ &\text{and} \\ v_2 =&((1,3),(5,6)) \\ v_1 \oplus v_2 \rightarrow v_3 =&((1,2),(3,4),(1,3),(1,3),(5,6)) \end{aligned} \tag{3.5}$$

$$\begin{aligned} v_1 =&((2,3),(3,4)) \\ &\text{and} \\ v_2 =&((2,4),(3,4)) \\ v_1 \oplus v_2 \rightarrow v_3 =&\varnothing \end{aligned} \tag{3.6}$$

Subtraction A subtraction operation Θ is applied to two positions, x_1 and x_2, and results on a velocity v, such that v applied to x_1 results on x_2 (i.e., $\vec{x_2} \leftarrow \vec{x_1} \circ \vec{v}$). It can be interpreted as the velocity needed by x_1 to be equal to x_2. If the positions are the same, i.e. $x_1 = x_2$, then the resulting velocity is an empty set, $x_1 \Theta x_2 = \varnothing$. For example, consider $x_1 = [3, 6, 5, 1, 2, 4]$ and $x_2 = [5, 6, 3, 4, 2, 1]$, Equation 3.7 shows an example of a subtraction operation.

$$\begin{aligned} x_1 =& (3,6,5,1,2,4) \\ &\text{and} \\ x_2 =& (5,6,3,4,2,1) \\ x_1 \Theta x_2 =& v = ((1,3),(4,6)) \end{aligned} \tag{3.7}$$

Multiplication The multiplication, represented by $\otimes$, takes a real number c and a velocity v_1 and results in a new velocity v_2. It admits four different cases for the real number c, as follows:

1. c = 0: $c \otimes v_1 \rightarrow v_2 = \varnothing$
2. $c \in (0,1]$: $c \otimes v_1 \rightarrow v_2 = v_1$ truncate in $\lfloor c * |v_1| \rfloor$ positions.

$$v_1 = ((1,2),(3,4)), c = 0.5$$
$$\lfloor c * |v_1| \rfloor \rightarrow \lfloor 0.5 * 2 \rfloor = 1$$
$$0.5 \otimes v_1 = ((1,2))$$

3. $c > 1$: $c \otimes v_1 \rightarrow v_2 = \underbrace{v_1 \oplus ... \oplus v_1}_{\text{k times}} \oplus c' \otimes v_1$, where $c = k + c', k \in N^\star, c' \in (0,1]$

$$v_1 = ((1,2),(3,4)), c = 1.5$$
$$k = 1, c' = 0.5, \text{ then } c = 1 + 0.5$$
$$1.5 \otimes v_1 = (\underbrace{(1,2),(3,4)}_{\text{1 time}}, c' \otimes v_1)$$
$$1.5 \otimes v_1 = ((1,2),(3,4),(1,2))$$

4. $c < 0$: $c \otimes v_1 \rightarrow v_2 = (-c)\neg v_1$, where $\neg v_1$ denotes the opposite velocity of $v_1(v_1 \oplus \neg v_1 = \varnothing)$

$$v_1 = ((1,2),(3,4)), \neg v_1 = ((4,3),(2,1)), c = -0.5$$
$$v_1 \oplus \neg v_1 = \varnothing$$
$$c \otimes v_1 = -c \otimes \neg v_1$$
$$c \otimes v_1 = 0.5 \otimes ((4,3),(2,1))$$
$$c \otimes v_1 = ((4,3))$$

Thus, with the operations properly described, we can define the general movement equation of the particle swarm. Equation 3.8 shows a particle i at time t having its velocity v_i and its position x_i updated. Here, the constants ϕ_1, ϕ_2, and ϕ_3 are -social/cognitive confidence coefficients, in which, according to Clerc (2004), specify how much the particle is influenced at the moment (ϕ_1), by its own experience (ϕ_2), and by its neighbors' experiences (ϕ_3).

$$\begin{cases} v_i(t) = & \phi_1 v_i(t-1) \;\oplus\; \phi_2(p_i \Theta x_i(t-1)) \;\oplus\; \phi_3(p_g \Theta x_i(t-1)) \\ x_i(t) = & x_i(t-1) \circ v_i(t) \end{cases} \tag{3.8}$$

In order to diversify the swarm movement, stochastic parameters can be added to Equation 3.8, resulting on Equation 3.9. The function $rand(0,1)$ returns a random number between zero and one with uniformly distributed probability on the interval.

$$\begin{cases} v_i(t) = & \phi_1 v_i(t-1) \;\oplus\; (\phi_2 \text{ rand(0,1)})(p_i \Theta x_i(t-1)) \;\oplus\; (\phi_3 \text{ rand(0,1)})(p_g \Theta x_i(t-1)) \\ x_i(t) = & x_i(t-1) \circ v_i(t) \end{cases} \tag{3.9}$$

Chapter 4

Methodology

In this work we propose to implement the metaheuristics described in Chapter 3, that is, a Genetic Algorithm, and a Particle Swarm Optimization. The genetic algorithm and the tailored PSO proposed in this work use the same initial solution and were fine tuned using a local search procedure. The next sections describe a fast heuristic proposed to generate initial solutions, a tailored local search method and the proposed metaheuristics.

4.1 Initial Population Generation

Each individual is represented as a sequence of gates and its fitness value associated. The heuristic based on the Breadth-First Search, described in Section 3.1.3, was the strategy employed to initialize part of the population for the tailored version of the PSO and HGA. For each new individual, we start the BFS from a different node. To ensure diversity, the other part of the initial population was created randomly.

Algorithm 3 presents an overview of the proposed heuristic method, which was published with its results on dos Santos and Carvalho (2015), also found in the attachments. Two different preprocessing methods (lines 1 and 3), described on Section 3.1.2, are applied in order to reduce the dimensions of the problem, without causing any damage to the final solution value. First, in line 1, a reduced matrix M_r is generated. After the creation of the original graph in line 2, a reduced graph G_r is generated in line 3. Subsequently, an initial solution is generated by applying the *BFS* from Section 3.1.3 to the reduced graph (line 4). This initial solution consists of the list S that contains the order in which the nodes were explored. The list of nodes (the representation of nets) is then transformed into a sequence of gates ϕ (line 5), according to the method of Section 3.1.3.1, which is used to determine the permuted matrix $\bar{M}_\phi$ (line 6). At the end, the matrix $\bar{M}_\phi$ with the solution is returned.

Algorithm 3: Proposed heuristic method overview

```
Data: Matrix $M$, set of gates $P$
1 $M_r \leftarrow$ GateDominancePreprocessing($M$);
2 $G \leftarrow$ BuildGraph($M_r$);
3 $G_r \leftarrow$ TwinNodesPreprocessing($G$);
4 $S \leftarrow$ BFS($G_r$);
5 $\phi \leftarrow$ GatesSequencing($S, P$);
6 $\bar{M}_\phi \leftarrow$ GenerateMatrix($M, \phi$);
7 return $\bar{M}_\phi$;
```

4.2 A Local Search Procedure

Intuitively, when we think of optimizing the layout of a gate matrix and decreasing the area of the implemented circuit, one solution would be to approximate the gates of the same net as much as possible, also reducing the wire length of each net. As in Figure 3.1 (c), for the nets r_1 and r_5 it might be interesting that gates p_2 and p_4 are sequenced before gate p_1 since this last gate is crossed by both nets and therefore requires longer wire length. However, minimizing the number of tracks is not equivalent to reducing their wire length, as shown by Linhares and Yanasse (2002).

Despite this non-equivalence between problems, this principle is used as a heuristic rule on a local search to try to decrease the maximum number of tracks required by the critical columns. In more detail, in a GMLP solution there are one or more gates with maximum sum of connections and crossings – since it is not possible to reduce the number of connections of a gate, the idea is to decrease the amount of wire that crosses it. It is important to emphasize that the goal is not to minimize the total wire length used, but to compact nets that cross the critical columns of the problem, which is to reduce the consecutive 1s of the columns with the largest sum in the matrix $\bar{M}\phi$. Thus, the local search tries to compress some nets, grouping its gates in two steps: one moving the gates to the right and another moving them to the left. All movements that do not generate worse solution values are performed.

Figure 4.1 (a) shows the circuit of the previous examples with the solution ϕ: [3, 6, 5, 1, 2, 4], obtained by the gate sequencing. With the purpose to illustrate the local search, consider the network r_2: in the first stage of the local search, there will be an attempt to reallocate the first gates of the net (p_3 and the gate dominated by it, p_6), to the right, just before gate p_1, in order to join the gates connected by the net. However, as shown in Figure 4.1 (b), this change increases the solution cost, from 4 to 5 tracks, so the sequence of gates ϕ_2: [5, 3, 6, 1, 2, 4] is discarded. In the second step, there will be an attempt to reallocate the gates of the same net to the left, just after the first gate of the net, thus, changing the order of gates p_1 and p_5 and obtaining the solution ϕ_3: [3, 6, 1 , 5, 2, 4], with cost 4 too. Thus, the algorithm considers solution ϕ_3 as the final one in order to improve the solution cost in the next iterations.

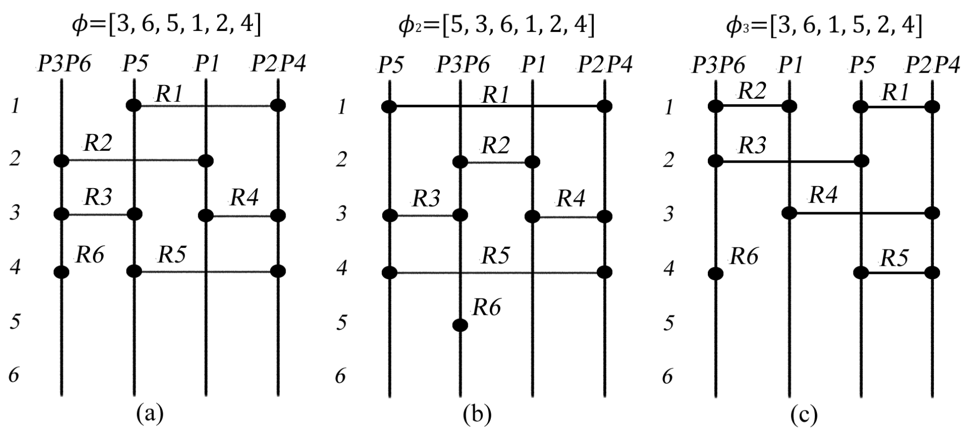

Figure 4.1: Application of the local search.

The two local search procedures have the same computational complexity. Both require an attempt to reallocate the gates of each net and an evaluation of the solution, which limits the computational complexity by $O(m^2n^2)$.

4.3 A Hybrid Genetic Algorithm with Auto-Adaptation

In order to solve the GMLP, a Hybrid Genetic Algorithm (HGA) was created based on the specifications of the problems and well known approaches presented on the past sections. Figure 4.2 presents a flowchart that represents the main idea of the HGA strategy. The hybridization of the Genetic Algorithm was made by creating initial individuals based on a heuristic method designed for the GMLP and by applying a local search method on the offspring after the reproduction.

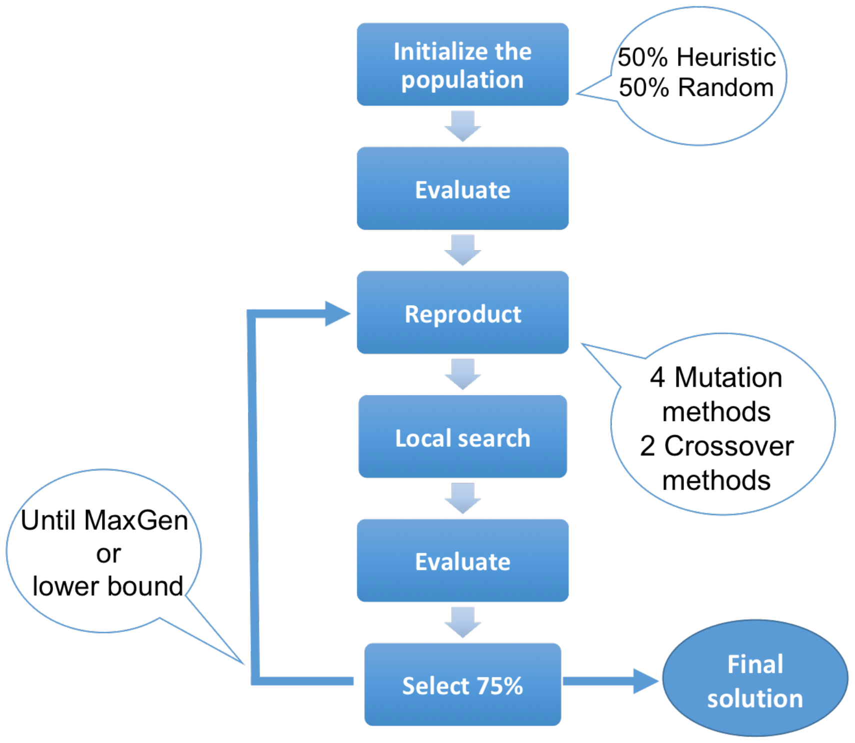

Figure 4.2: Genetic Algorithm strategy flowchart.

4.3.1 Reproduction

Four different mutation methods and two crossover operators were implemented. All the strategies were specially designed to not violate any restriction of the problem. Since it is a permutational problem, the gates cannot be repeated. These operators are briefly described next.

Mutation:

- Swap: Takes two genes and swaps their contents.
- Insertion: Takes a random gene content and inserts it in a random position.
- Inversion: Takes two random genes and reverses all the contents on the interval, including the selected genes.
- Shuffle: Takes two random genes and shuffle the contents on the interval, including the selected genes.

Crossover:

- One cut: Splits the parents on a random position and combines its parts to create the offspring.

- Two cut: Splits the parents in two random positions and swaps the middle part within the parents to create the offspring.

4.3.2 Selection

After the reproduction phase, the new population has double the size of the previous population, considering the parents and offspring. The natural selection concept is brought from the biology to this context to maintain the individuals with best fitness in the next generation.

At each generation, parents and offspring are sorted by its fitness and a percentage k of the new population is generated by the best individuals in this pool. The other part of the new population is generated from the parents to guarantee diversity of individuals. Figure 4.3 shows an illustration of the selection phase, where k is the percentage of the best individuals.

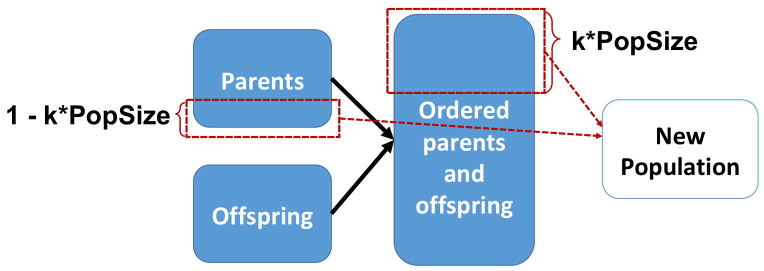

Figure 4.3: Selection.

4.3.3 Stopping Criteria

The Genetic Algorithm is not an exact method, which means that, in most cases, it cannot determine if a solution is optimal. Since the algorithm cannot identify when its best possible result was found, we have to define how long it will run with the population reproducing and exploring the solution space. Thus, stopping criteria need to be defined.

One strategy to create a stopping criterion is to define a lower bound, which, according to the constraints of a problem, it is not possible to further improve the fitness. In the GMLP, a lower bound on the number of tracks is the maximum column sum of the matrix, before filling the consecutive ones.

Another possible stopping criterion is to determine a maximum number of generations the algorithm will run. Both strategies were implemented on the proposed methods: while the lower bound is not reached, the algorithm runs until the maximum number of iterations.

4.3.4 Auto Adaptation

The genetic algorithm has a set of parameters with great influence over its performance. The configuration of these parameters itself can be considered as part of the individual's gene and reproduced along execution of the algorithm. This approach is called *auto adaptation* and explores various combinations of parameters and its effects on the solution during execution time.

An auto adaptation approach was used to probabilistically choose within the different crossover and mutation methods and define the best crossover and mutation rates. They were represented on a vector of probabilities and crossover and mutation operations were applied on these vectors while the algorithm was executed.

4.3.5 Pseudocode

Algorithm 4 presents an overview of the proposed HGA method. First, the initial population is generated (line 1), a part by applying the Heuristic Method for the GMLP previous defined and the other part is generated randomly. In line 3 the Genetic algorithm starts, first the reproduction, followed by a local search (line 4) on the offspring and its evaluation (line 5). The population is selected in line 6. At the end, the best individual with the solution is returned.

Algorithm 4: Proposed HGA method overview

```
1 $Population \leftarrow$ InitializePopulation();
2 repeat
3     $Offspring \leftarrow$ Reproduction();
4     $Offspring_{LS} \leftarrow$ LocalSearch($Offspring$);
5     Evaluate($Offspring_{LS}$);
6     $Population \leftarrow$ Selection($Offspring_{LS}, 0.75$);
7 until $StopCriterion$;
8 return gbest($Population$);
```

4.4 A Tailored Discrete Particle Swarm Optimization

In the PSO, developed by Kennedy and Eberhart (1995), each individual of the flock is a candidate solution and know its location. The main difference of this implementation regards how an individual can go towards a direction. Because the GMLP is a discrete problem, the traditional calculation of velocity doesn't work for it, as presented in the previous section. The population was initialized with the method described before.

4.4.1 Movements

Similar to what was defined by Goldbarg et al. (2006) while solving the TSP, in the discrete PSO the individuals are able to make one of three movements:

ϕ_1: follow a generic local search method, towards to a local optimum;

ϕ_2: follow a tailored local search method, towards to a local optimum;

ϕ_3: move towards the global best solution.

The particle choice is made randomly according to the probabilities of each movement. If a particle chooses to follow its own way, a local search procedure is applied to improve the current solution. A general two-swap technique was implemented to swap a number of pairs of gates on the solution and keep the ones which improved the solution value. Instead, if the particle chooses to follow a heuristic, the previously defined heuristic method is applied to the solution, in an attempt to compact a net by eliminating the fill-ins of the matrix. To do so, each line of the matrix is individually analyzed and the gates are shifted to the left or to the right if there is no increase in the number of required tracks.

The concept introduced by Glover and Laguna (1997) named *Path Relinking* brings the idea of identify a path between a set of solutions, usually two with one better than other, and follow this path to create a new solution. Having an origin solution, a target solution, and a movement, the algorithm creates this path by exploring the neighborhood space. The authors state that each movement applied on the origin solution must introduce attributes that will contribute to reduce the distance between the origin and target solutions.

Figure 4.4 illustrates how the path relinking operation works. Given an original solution [3, 5, 6, 1, 4, 2], a target solution [3, 6, 5, 1, 2, 4], and a movement (swap, in this case), the algorithm can identify the required steps to move to the target solution. At Step 1, a swap between positions 1 and 2 creates a intermediary solution [3, 6, 5, 1, 4, 2], that is closer from the target solution. Step 2, with a swap between positions 4 and 5, progressively reduce the distance between the current and the target solutions, achieving the target with two steps.

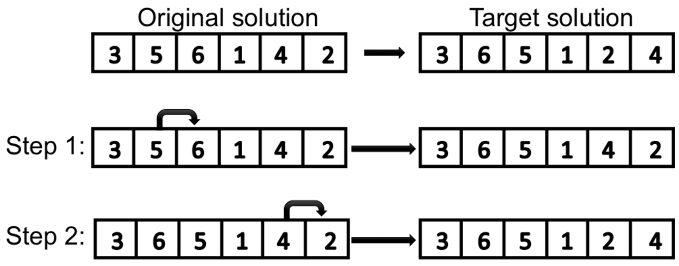

Figure 4.4: Path relinking applied to an origin to a target solution.

Finally, if the particle chooses to go towards the global best solution a path relinking operation is applied on the solution targeting the global best, until an improvement is achieved, if not, nothing is changed on the solution.

4.4.2 Pseudocode

Algorithm 5 presents the proposed PSO method. The initial swarm is generated (line 1) as described in the beginning of this chapter, a portion by applying the *BFS* to the reduced graph multiple times starting from different nodes, and then transformed into a sequence of gates, and the other portion is randomly generated. The algorithm will loop until a stop criterion is met (lines 2 - 12). In each step, the particles are evaluated (line 4), and *gbest* is updated according to the particles fitness value (line 6). Finally, each particle in the swarm has to execute one movement (line 10). The individual with best fitness of the population is returned at the end of the method (line 13).

Algorithm 5: Proposed PSO method overview

```
1  $Swarm \leftarrow$ InitializeSwarm();
2  repeat
3      foreach particle p in Swarm do
4          $p_{fitness} \leftarrow$ Evaluate($p$);
5          if $p_{fitness} < gbest$ then
6              $gbest \leftarrow p_{fitness}$
7          end
8      end
9      foreach particle p in Swarm do
10         $p_{solution} \leftarrow$ Movement($p, \alpha$);
11     end
12 until StopCriterion;
13 return gbest;
```

Chapter 5

Computational Experiments

The computational experiments were carried out on a computer with a 3.2 GHz Intel i5 Quad Core processor with 16 GB RAM using the operating system Ubuntu 12.4.1. The proposed methods' code was written in C++, compiled with g++ 4.4.1 and the optimization option -O3. Four sets of instances were used in the experiments, and the optimal results reported by the exact method of Chu and Stuckey (2009) were used in the comparisons, since none of the literature methods on GMLP were previously tested using all these sets of instances. Due to the randomness factor of the methods, 10 independent executions were performed for each of the instances and the average results were presented.

Among the sets of instances, there are a set of GMLP-specific instances and three sets of MOSP (Minimization of Open Stacks Problem) instances, an industrial problem whose formulation is equivalent to the GMLP formulation, as mentioned in Chapter 1. The gap, or percentage distance, is calculated as 100 × (*heuristic solution value*) / (*reference value*). The running times are not compared in these experiments because, as the methods were executed in different architectures, a fair comparison is not possible.

5.1 Parameters Tuning

The particles on the PSO algorithm have to choose one of three movements at each iteration. This chose is made based on given probabilities of the movements:

p_{ϕ_1}: probability to follow a generic local search method;

p_{ϕ_2}: probability to follow a tailored local search method;

p_{ϕ_3}: probability to follow the global best solution.

The tailored local search procedure, associated to p_{ϕ_2}, as described on Section 4.2, analyzes a predefined percentage of gates in the *lookahead* mechanism. Other important parameters on the algorithm are the *population size* and the *number of generations*, which will define the amount of particles and for how long they will be searching the solution space. Finally, the global best solution has a probability of undergoing a local search procedure, which also must be tuned.

In order to fine tuning the method, the irace package, implemented by López-Ibáñez et al. (2016), was used. A set of possible parameters and values and a representative set of the problem instances were fed to the package, so it could determine the best configuration for the PSO. Table 5.1 presents the range of each possible parameter value. In addition to these parameters, the population was initialized with the method described at the beginning of the section, with 40% of the particles created with the heuristic and 60% randomly initialized.

Table 5.1: PSO parameters tuning using the irace package.

Parameters	Values
p_{ϕ_1} (%)	25, 30, 35, 40
p_{ϕ_2} (%)	15, 20, 25, 30, 35
Population size	40, 50, 60, 70, 80, 90, 100
Number of generations	20, 30, 40, 50, 60, 70
Global Best Solution Local Search Probability (%)	30, 40, 50, 60, 70, 80
Lookahead (%)	20, 30, 40, 50

The best configuration found by the irace package are presented following:

- $p_{\phi_1} = 30\%$;
- $p_{\phi_2} = 25\%$;
- $p_{\phi_3} = 45\%$;
- Population size = 60;
- Number of generations = 20;
- Global Best Solution Local Search Probability= 30%;
- Lookahead = 30%.

As in the PSO, the number of individuals and generations on the HGA are important parameters on the algorithm and can determine its performance. Thus, these parameters were fine tuned with the irace package using the set of possible parameters shown in Table 5.2. Other important parameters, the mutation and crossover rates, were auto adapted, as mentioned in Section 4.3.4. During the execution of the algorithm, crossover and mutation methods were applied to these parameters to find the best configuration.

The best configuration found by the irace package are presented following:

Table 5.2: HGA parameters tuning using the irace package.

Parameters	Values
Population size	40, 50, 60, 70, 80, 90, 100
Number of generations	20, 30, 40, 50, 60, 70

- Population size = 80;
- Number of generations = 40;

Additionally, 50% of the initial population was created by applying the heuristic method for the GMLP previously defined, and the other 50% is generated randomly. During the selection phase, 75% of the best individuals within parents and offspring are selected, and 25% are randomly picked form the parents.

5.2 Real-World GMLP Instances

The first set of instances contains 25 real GMLP instances of Asian companies, introduced by Hu and Chen (1990). Dimensions range from 7 rows and columns to 202 rows and 141 columns. Table 5.3 presents the results (expressed in number of tracks) and the running times (expressed in seconds) of the proposed methods for these instances. The optimal results for this set are also presented as reference values and the mean values of each column are given on the last row.

For this set, the methods were able to match most of the optimal solutions. The PSO obtained two worse solutions, in instances *w4* and *v400*, against one worse solution from HGA, in instance *v4470*. Both methods differed by only a fraction on the average. The total gap between the reference values and the PSO was 0.37% and the HGA was 0.28%, on average. Comparing the average improvement from the initial population best fitness to the final global best fitness, both method had similar results, PSO reduced 3.05 tracks while HGA 3.01 tracks. In addition to best results, the HGA converged faster than the PSO, it took only 1.67 generation on average to find the best solution, needing less than half of PSO running time to solve the problems, which shows that HGA had better performance for this set of instances.

Table 5.3: Results for the real instances VLSI.

Instance	Optimal Solution	PSO		HGA	
		Solution	Time (s)	Solution	Time (s)
v4000	5.00	5.33	0.04	5.00	0.03
v4050	5.00	5.00	0.01	5.00	0.01
v4090	10.00	10.00	0.30	10.00	1.09
v4470	9.00	9.00	3.75	9.50	3.84
vc1	9.00	9.00	0.02	9.00	0.02
vl	3.00	3.00	0.00	3.00	0.00
vw1	4.00	4.00	0.00	4.00	0.00
vw2	5.00	5.00	0.02	5.00	0.10
w1	4.00	4.00	0.02	4.00	0.02
w2	14.00	14.00	0.25	14.00	0.20
w3	18.00	18.00	14.99	18.00	16.36
w4	27.00	27.33	150.45	27.00	46.79
wan	6.00	6.00	0.00	6.00	0.00
wli	4.00	4.00	0.01	4.00	0.00
wsn	8.00	8.00	0.23	8.00	0.57
x0	11.00	11.00	3.14	11.00	3.53
x1	5.00	5.00	0.04	5.00	0.14
x2	6.00	6.00	0.09	6.00	0.28
x3	7.00	7.00	0.29	7.00	0.70
x4	2.00	2.00	0.02	2.00	0.05
x5	2.00	2.00	0.09	2.00	0.13
x6	2.00	2.00	0.79	2.00	0.52
x7	4.00	4.00	0.03	4.00	0.12
x8	4.00	4.00	0.14	4.00	0.41
x9	4.00	4.00	1.13	4.00	2.09
Average	7.12	7.15	7.03	7.14	3.08

5.3 Real-World MOSP Instances

This set of instances, provided by the *Sheet Cutting and Process Optimization for Furniture Industry* (SCOOP) consortium[1], contains 187 real-world MOSP instances from two European companies. However, most of these instances are very small, e.g. 2 rows and columns, with trivial solutions. From this set, 24 instances with size ranging from 10 rows and 14 columns to 49 rows and 134 columns were selected to the experiments. Table 5.4 presents the results using the same configuration as the previous table.

Both methods achieved the best solutions on 87.5% of these instances. They had the same average results on this set, with a 1.25% gap between the methods and the optimal solutions, or 0.10 additional track on average. Individually, the methods had worse results

[1]Available at: `www.scoop-project.net`

Table 5.4: Results for the real-world MOSP instances.

Instance	Optimal Solution	PSO		HGA	
		Solution	Time (s)	Solution	Time (s)
A_AP-9.d-10	6.00	6.00	0.11	6.00	0.27
A_AP-9.d-11	6.00	6.00	0.26	6.00	0.51
A_AP-9.d-3	6.00	6.00	0.12	6.00	0.29
A_AP-9.d-6	5.00	5.00	0.22	5.00	0.35
A_FA+AA-1	12.00	12.67	4.63	12.83	5.54
A_FA+AA-11	11.00	11.00	1.99	11.00	2.57
A_FA+AA-12	9.00	9.00	0.45	9.00	0.89
A_FA+AA-13	17.00	17.67	5.64	17.83	7.12
A_FA+AA-15	9.00	9.00	0.41	9.00	0.86
A_FA+AA-2	11.00	11.00	0.41	11.00	0.91
A_FA+AA-6	13.00	13.00	1.09	13.00	2.88
A_FA+AA-8	11.00	12.00	2.07	11.67	3.40
B_12F18-11	6.00	6.00	0.14	6.00	0.38
B_12M18-12	6.00	6.00	0.09	6.00	0.06
B_18AB1-32	6.00	6.00	0.06	6.00	0.29
B_18CR1-33	4.00	4.00	0.08	4.00	0.17
B_22X18-50	10.00	10.00	0.08	10.00	0.39
B_23B25-52	5.00	5.00	0.24	5.00	0.44
B_39Q18-82	5.00	5.00	0.01	5.00	0.01
B_42F22-93	5.00	5.00	0.05	5.00	0.18
B_CARLET-137	5.00	5.00	0.01	5.00	0.01
B_CUC28A-138	6.00	6.00	0.07	6.00	0.05
B_GTM18A-139	5.00	5.00	0.16	5.00	0.33
B_REVAL-145	7.00	7.00	4.43	7.00	4.66
Average	7.75	7.85	0.95	7.85	1.36

for three instances, *A_FA+AA-1*, *A_FA+AA-13*, and *A_FA+AA-8*, some of the largest of this group. Differently from the previous set, the PSO converged 0.29 generation earlier than HGA, on average, and execute 1.34 times faster. As the individual with best fitness on the initial population had similar value on both algorithms, an average of 10.88 on the PSO and 10.84 on the HGA, the PSO had a better overall performance on this set.

5.4 Artificial MOSP Instances

A third set of 150 large artificial MOSP instances [2] was also considered. These instances were generated randomly, without any of the specific structures pointed out by Yanasse and Senne (2010) that could make the solution easier to determine. The dimensions range from

[2] Available at http://www.decom.ufop.br/marco/esquisa/problem-instances/

150 rows and columns to 200 rows and columns. In Table 5.5, *m*-*n*-*d* identifies each set of 10 instances, where m is the number of nets, n the number of gates, and d is the number of nets per gate, that is, the fixed demand for each gate. Again, the mean values of reference are the proven optimal solutions or the upper bound reported by Chu and Stuckey (2009).

Table 5.5: Results for the artificial MOSP instances.

Instance	Optimal Solution	PSO		HGA	
		Solution	Time (s)	Solution	Time (s)
150-150-2	25.90*	27.60	375.21	28.50	60.44
150-150-4	61.60	62.72	583.98	63.30	194.29
150-150-6	93.20*	95.17	908.13	95.32	428.92
150-150-8	111.70*	113.77	1245.94	113.88	808.12
150-150-10	123.90*	125.72	1610.28	125.67	1330.02
175-175-2	30.30	32.68	651.70	34.15	86.79
175-175-4	73.70	75.87	1106.80	76.58	275.34
175-175-6	107.70	109.77	1675.35	109.90	685.21
175-175-8	128.30*	130.40	2248.20	131.05	1055.12
175-175-10	143.50*	145.52	2900.24	145.75	1791.37
200-200-2	36.00	38.43	975.10	40.22	101.39
200-200-4	84.30	85.62	1534.26	86.68	311.53
200-200-6	121.50	123.32	2305.54	123.93	694.15
200-200-8	147.10	149.70	2985.41	150.02	1290.08
200-200-10	162.80*	165.63	4075.71	166.32	2058.29
Average	96.77	98.79	1678.79	99.42	744.74

*Proven optimal solution.

Of the 150 instances of this set, the exact method of Chu and Stuckey (2009), developed to solve the MOSP, was able to find optimal solutions for only 91 of them. Using a similar architecture to the one used in the experiments of this work, the average running time was 3.78 hours per instance.

The total gap considering the reference values is 2.09% for the PSO and 2.74% for the HGA. Although the PSO gap is smaller, its running time was again more than twice the HGA running time. The PSO needed about 28 minutes per instance, on average, while HGA needed only 12 minutes. However, despite the increase of these values, they are still considered low and desirable, since it is a metaheuristic and the dimensions of the instances are large.

On both methods, the initial population global best presented the same fitness value and the final global best solution was achieved by the PSO 7.4 generations later than for HGA, on average. Meaning that the method was able to improve the solutions during more generations, which worth the extra time consumed compared with HGA. Comparing with the average running time of the reference method, with only 12.34% of the time, PSO achieved a result with a 2.09% gap. Considering that these instances are larger and do not have special structures, such result is considerable.

5.5 First Constraint Modeling Challenge Instances

The fourth set of instances was provided by the First Constraint Modeling Challenge (Smith and Gent, 2005). From the original set, 21 larger MOSP instances ranging from 20 to 100 lines and from 10 to 100 columns, without any of the specific structures pointed out by Yanasse and Senne (2010), were selected. Table 5.6 presents the results for this set with the optimal solutions from Chu and Stuckey (2009) as reference values.

Table 5.6: Results for the First Constraint Modeling Challenge instances.

Instance	Optimal Solution	PSO		HGA	
		Solution	Time (s)	Solution	Time (s)
Miller	13.00	13.00	4.41	13.00	9.60
GP1	45.00	45.00	156.72	45.00	284.01
GP2	40.00	40.00	137.05	40.00	236.52
GP3	40.00	40.83	136.93	40.67	299.55
GP4	30.00	30.00	16.49	30.00	27.88
GP5	95.00	95.00	2166.92	95.00	6415.05
GP6	75.00	75.00	1678.79	75.00	4308.16
GP7	75.00	75.50	1730.09	75.33	3831.47
GP8	60.00	61.17	1399.46	60.00	2932.03
NWRS1	3.00	3.00	0.01	3.00	0.01
NWRS2	4.00	4.00	0.02	4.00	0.02
NWRS3	7.00	7.00	0.40	7.00	1.68
NWRS4	7.00	7.00	0.55	7.00	2.27
NWRS5	12.00	12.00	1.49	12.00	6.85
NWRS6	12.00	12.00	1.97	12.00	8.33
NWRS7	10.00	10.00	4.99	10.00	12.28
NWRS8	16.00	16.00	9.21	16.00	21.66
SP1	9.00	9.00	0.34	9.00	0.88
SP2	19.00	19.50	6.88	19.67	8.48
SP3	34.00	34.00	43.49	34.00	35.11
SP4	53.00	53.67	147.02	53.67	91.80
Average	31.38	31.56	363.96	31.49	882.55

The methods achieved the best results for the majority of the 21 instances of this group. The average gap between them and the optimal solutions was 0.56% for the PSO and 0.35% for the HGA. Both of them got worse results for the larger instances *GP3*, *GP7*, *SP2*, and *SP4*, and the PSO had a worse result for the instance *GP8* too. For this set, the PSO algorithm converged 1.23 generations earlier and was 2.42 times faster than the HGA, and again, the later convergence led for a longer running time.

The average difference between the initial global best fitness and the final global best was 5.48 tracks on the PSO and 5.45 tracks on HGA. Although the HGA had lower average results than PSO, if we analyze the starting point and progress of the population, PSO made an equidistant, or even slightly better, improvement on the solutions. Finally, the PSO achieved better performance because with less time and faster convergence, managed to improve the solution as much as the HGA.

Chapter 6

Conclusions

The Gate Matrix Layout Problem (GMLP) is an NP-hard problem with industrial applications in the design of very large scale integration circuits that also successfully models a variety of graph theory and scheduling problems. The purpose of the GMLP is, given a matrix of logical gates and nets that connect these gates, to determine a permutation of the gates (i.e., a layout for gate matrix) such that the area required to implement the electronic circuit is minimized. This way, these circuits can be cheaper and faster produced.

Along with a brief theoretical foundation on the problem, this work proposes two evolutionary methods applied to the solution of the GMLP, a Hybrid Genetic Algorithm (HGA) and tailored implementation of the Particle Swarm Optimization (PSO) algorithm. The coupling of the methods to the problem is mainly made by a two-stage heuristic and a local search procedure. The heuristic was designed to create the initial population and, in its first phase, the problem is modeled as a graph with a specific structure and uses the well-known breadth-first search to generate a sequence of nets. In the second phase, from sequence of nets, the heuristic finds a possible solution to the original problem, i.e., a permutation of the gates. The local search was designed to compact certain sets of circuit components, trying to decrease the maximum number of tracks required by the critical columns.

Computational experiments involving 220 instances of four different sets, including real-world instances, were reported. The proposed methods were able to match a good part of the best available results. In the real-world MOSP instances set, both methods had the same average results. The PSO was better than HGA with artificial MOSP instances, while the HGA had a smaller total gap in the sets of real-world GMLP and First Constraint Modeling Challenge instances. Experiments with an unprecedented set of instances in the GMLP context were also reported.

Future works will be focused on improving the local search procedure and experimenting with different methods, as an attempt to reduce its running time. Additionally, we consider it is necessary to develop new crossover and mutations methods also tailored to the GMLP characteristics, in order increase HGA coupling to the GMLP.

Attachments

Full paper published in conference proceedings.

SANTOS, J. V. M.; CARVALHO, M. A. M. Uma Heurística Aplicada à Produção em Microeletrônica In: XLVII Simpósio Brasileiro de Pesquisa Operacional, 2015, Porto de Galinhas. Anais do XLVII Simpósio Brasileiro de Pesquisa Operacional. 2015.

Uma Heurística Aplicada à Produção em Microeletrônica

João Vitor Mascarenhas dos Santos
Departamento de Ciência da Computação, Universidade Federal de Ouro Preto
Campus Morro do Cruzeiro, Ouro Preto, Minas Gerais, 35400-000, Brasil.
joaovitormascarenhas@yahoo.com.br

Marco Antonio Moreira de Carvalho
Departamento de Ciência da Computação, Universidade Federal de Ouro Preto
Campus Morro do Cruzeiro, Ouro Preto, Minas Gerais, 35400-000, Brasil.
mamc@iceb.ufop.br

RESUMO

Este artigo apresenta uma heurística para o Problema de Determinação do Leiaute de Matrizes de Portas (ou GMLP, do inglês *Gate Matrix Layout Problem*), um problema de aplicação prática na indústria microeletrônica. No contexto de projeto de circuitos VLSI (*Very Large Scale Integration*), este problema pede por uma permutação das colunas de uma matriz de portas tal que o número exigido de trilhas para implementação dos circuitos – e consequentemente a área – sejam minimizados. A heurística se baseia em busca em grafos seguida por procedimentos de busca local e é comparada com dois métodos da literatura em experimentos computacionais abrangentes que consideraram conjuntos de instâncias reais e artificiais da literatura. Os resultados reportados mostram que a heurística proposta, em um curto intervalo de tempo, pôde igualar boa parte dos resultados da literatura, aprimorar alguns deles e também obter um baixo *gap* quando comparada às soluções ótimas e limitantes inferiores disponíveis.

PALAVRAS CHAVE. Leiaute de Matrizes de Portas. Heurísticas. Escalonamento.

Área Principal: IND

ABSTRACT

This paper introduces a heuristic method for the Gate Matrix Layout Problem (GMLP), a problem with practical application in the microelectronics industry. In the context of Very Large Scale Integration (VLSI) design, this problem asks for a permutation of columns of a gate matrix such that the number of required tracks necessary to implement the corresponding integrated circuit, and thus the area, are minimized. The proposed heuristic is based on graph search followed by local search procedures and is compared to two methods from the literature in comprehensive computational experiments that considered four datasets publicly available of real and artificial instances. The reported results show that the proposed method, in a short amount of time, was able to match a good part of the best results available, to improve some of them and to obtain a low gap when compared to optimal solutions and lower bounds available.

KEYWORDS. Gate Matrix Layout. Heuristics. Scheduling.

Main Area: IND

1. Introdução

Atualmente, a microeletrônica está presente nas mais diversas áreas, sendo fundamental nas áreas de informática, telecomunicações, controles de processos industriais, automação dos serviços bancários e comerciais e também na fabricação de bens de consumo, com aplicações práticas diretas na engenharia e indústria.

Na criação de componentes deste ramo da eletrônica, são utilizados circuitos integrados em larga escala (ou VLSI, do inglês *Very Large Scale Integration*), projetados pelo uso de uma matriz lógica programável. Com o intuito de otimizar a produção dos referidos componentes, busca-se uma ferramenta computacional para determinação do leiaute físico otimizado dos componentes para que seja possível produzir circuitos mais baratos, mais rápidos e mais compactos. A área do circuito é fator de suma importância na minimização do custo da produção, já que os mesmos são produzidos sobre discos de silício, normalmente chamados de *wafers*, onde o custo do circuito é inversamente proporcional a quantidade de circuitos feitos em um só *wafer*, ou seja, quanto menor o tamanho do circuito, uma maior quantidade poderá ser produzida em um único *wafer*, tornando-o, assim, mais barato.

Uma *Matriz de Portas* é um dispositivo lógico que dispõe de duas dimensões de *portas*, uma AND e outra OR, ambas programáveis. Conexões entre estas portas podem ser realizadas para alcançar resultados específicos. Desta forma, a matriz de portas é utilizada para elaborar circuitos lógicos combinatórios, composta por n portas lógicas e m *redes* – um subconjunto de portas ligadas por um *fio*. No caso de diferentes redes utilizarem uma mesma porta, cada rede deve ser alocada em uma *trilha* diferente garantindo assim, que não haja sobreposição das redes.

Na Figura 1(a) é apresentada uma matriz de portas com seis portas enumeradas de $P1$ a $P6$, representadas como as linhas verticais (colunas), e seis redes enumeradas de $R1$ a $R6$, representadas pelos conjuntos de pontos na horizontal. Em seguida, a Figura 1(b) mostra as conexões entre as portas da rede, onde, para ligar uma porta a outra, pode ser necessário passar por outras portas, sem utilizá-las. Um diferente leiaute das portas da matriz é apresentado na Figura 1(c), no qual é possível que diferentes redes sejam alocadas a uma mesma trilha.

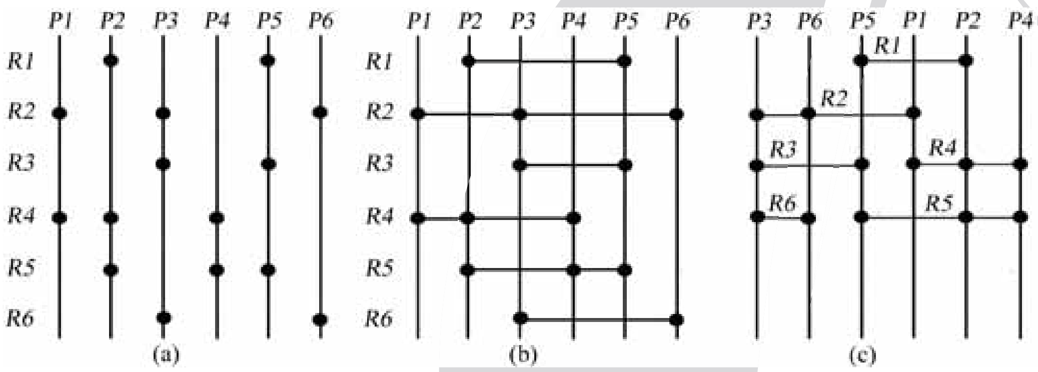

Figura 1. Matriz de Portas (a), com redes destacadas (b) e com uma permutação diferente das portas (c).

O Problema de Determinação do Leiaute de Matrizes de Portas (ou GMLP, do inglês *Gate Matrix Layout Problem*), consiste em determinar uma permutação das portas de uma matriz tal que a quantidade de trilhas necessárias para implementar todas as redes seja minimizada, e, por consequência, a área do circuito também seja minimizada. Como demonstrado na Figura 1(b), são necessárias seis trilhas para acomodar todas as redes do circuito. Já na Figura 1(c), com outra permutação das portas, são necessárias quatro trilhas apenas.

Como definido por De Giovanni et al. (2013), uma instância do GMLP pode ser descrita por uma matriz $M \in \{0,1\}\ m \times n$. As linhas da matriz M são associadas às m redes e as colunas são associadas às n portas, tal que $M(i,j) = 1$ se e somente se a rede i incluir a porta j. Uma solução para o GMLP é uma sequência $\phi : [1, \ldots, n] \rightarrow [1, \ldots, n]$, em que $\phi(j)$ indica posição da porta j na matriz de portas. Tal solução define uma nova matriz M_ϕ, obtida pela permutação das colunas de M de acordo com ϕ. A partir da matriz M_ϕ, possuidora da propriedade dos 1's consecutivos, obtemos uma nova matriz $\bar{M}_\phi$ de forma que $\bar{M}_\phi(i, j) = 1$ se e somente se, de acordo com ϕ, o fio da rede i incluir ou cruzar a porta j. Portanto, dada uma matriz $M \in \{0, 1\}\ m \times n$, o objetivo do GMLP é determinar uma permutação das colunas que resulte em uma matriz $\bar{M}_\phi$ correspondente tal que a maior soma de elementos em qualquer coluna seja minimizada, o que equivale a minimizar o número máximo de trilhas necessárias para implementação do circuito correspondente.

A Figura 2(a) apresenta a matriz M do circuito apresentado anteriormente na Figura 1(a). O circuito com as portas permutadas exibido na Figura 1(c) é representado pela matriz M_ϕ na Figura 2(b), onde, após aplicar a propriedade dos 1's consecutivos, obtemos a nova matriz $\bar{M}_\phi$ onde as posições entre o primeiro e o último 1 de cada rede recebem também o valor 1, como mostra a Figura 2(c). Assim, determinar o custo da solução significa encontrar a maior soma entre as colunas. A importância da propriedade dos 1's consecutivos se deve à necessidade de representar que um fio cruza uma porta, mesmo que esta não faça parte da rede. Por exemplo, no circuito da Figura 1(c), o fio da rede $R1$ cruza a porta $P1$ e, consequentemente, na matriz da Figura 2(c) a posição $\bar{M}_\phi$(1, 1) possui valor 1.

$$M= \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \qquad M_\phi= \begin{array}{cccccc} 3 & 6 & 5 & 1 & 2 & 4 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{array} \qquad \bar{M}_\phi= \begin{array}{cccccc} 3 & 6 & 5 & 1 & 2 & 4 \\ \hline 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{array}$$

(a) (b) (c)

Figura 2. Matrizes M (a), M_ϕ (b) e $\bar{M}_\phi$ (c).

O Problema de Determinação do Leiaute de Matrizes de Portas foi provado ser NP-difícil por uma redução do problema de aumento grafos de intervalo (Kashiwabara e Fujisawa, 1979, *apud* Linhares e Yanasse, 2002) e possui aplicações em diferentes áreas, bem como diferentes problemas de formulação equivalente (Linhares e Yanasse, 2002): Problema de Corte Modificado (*Modified Cutwidth*), Problema de Minimização de Pilhas Abertas (*Minimization of Open Stacks Problem* – MOSP), Dobradura de Arranjos Lógicos Programáveis (*Programmable Logic Array Folding*, ou *PLA Folding*), *Interval Thickness*, *Node Search Game*, *Edge Search Game*, *Narrowness*, *Split Bandwidth*, *Graph Pathwidth*, *Edge Separation* e *Vertex Separation*.

Recentemente, o interesse na aplicação prática deste problema têm crescido. Com a recente instalação, em Minas Gerais, da primeira fábrica de semicondutores do hemisfério sul, este trabalho possui o potencial para gerar inovação aplicável a este novo nicho industrial.

Este artigo está organizado como a seguir. Uma breve revisão da literatura é realizada na Seção 2. Na Seção 3, a heurística proposta é descrita em detalhes, cada etapa é ilustrada. Experimentos computacionais extensivos que levaram em consideração 190 instâncias reais de empresas e artificiais da literatura são reportados na Seção 4, e finalmente, na Seção 5, conclusões sobre este trabalho de pesquisa são realizadas e trabalhos futuros são indicados.

2. Revisão da Literatura

Métodos heurísticos vêm sendo aplicados as GMLP desde o trabalho de Wing, Huang e Wang (1985), no qual dois circuitos reais foram modelados usando grafos de intervalos, considerando portas em comum em diferentes trilhas como uma interseção. Posteriormente, Hwang, Fuchs e Kang (1987) conseguiram resultados mais relevantes com o algoritmo *min-net-cut* modificado cuja complexidade de tempo é $O(n\, log\, n)$. O mesmo algoritmo foi utilizado novamente por Chen e Hou (1988), para atingir melhores resultados que os anteriores da literatura nos 5 circuitos testados.

Na década seguinte, os trabalhos de Chen e Hu (1990) e Hu e Chen (1990) alcançaram, ou determinaram, os melhores resultados para todas as instâncias disponíveis na literatura até aquele momento. Foram apresentados dois algoritmos, *GM-Plan* e *GM-Learn*, ambos baseados em paradigmas de inteligência artificial.

No final da década de 80 e boa parte da década de 90, o GMLP foi abordado predominantemente por técnicas metaheurísticas e métodos híbridos. Shahookar et al. (1993) propuseram uma implementação de algoritmos genéticos e *Beam Search*, denominada *Genetic Beam Search*. Com a estratégia de *Busca Predatória*, Linhares (1999) alcançou os resultados obtidos por *GM-Plan* e *GM-Learn*. Além disto, das 25 instâncias, em 12 foi possível igualar o número de trilhas ao limite inferior, e em 4 foram atingidos os melhores resultados da literatura. Linhares, Yanasse e Torreão (1999) apresentaram a *Otimização Microcanônica*, um procedimento derivado da física estatística assim como o *Simulated Annealing*. Este método foi capaz de igualar todos os resultados anteriores de Chen e Hu (1990) e Hu e Chen (1990), mas não os resultados obtidos pela *Busca Predatória*.

Algoritmos Genéticos Construtivos foram utilizados por Oliveira e Lorena (2002), alcançando todos os melhores resultados conhecidos da literatura, obtidos anteriormente pela *Otimização Microcanônica*. Segundo os autores, este algoritmo genético construtivo é mais robusto do que a *Otimização Microcanônica*. Uma abordagem evolutiva de múltiplas populações foi introduzida por Mendes e Linhares (2004), agregando algoritmos genético e memético. Este trabalho apresentou resultados iguais aos da *Otimização Microcanônica*, superando todas as abordagens metaheurísticas e heurísticas anteriores da literatura – incluindo *Simulated Annealing*, *GM-Plan*, *GM-Learn*, e heurísticas construtivas – tanto em qualidade das soluções obtidas quanto em eficiência, em termos de convergência do algoritmo. A rápida convergência é justificada pela adição de uma busca local heurística que proporcionou a redução do espaço de busca, considerando apenas o gargalo do problema, o que foi denominado *colunas críticas*.

Os métodos exatos aplicados ao GMLP são encontrados na literatura com frequência muito menor. Destaca-se a formulação por programação dinâmica apresentada em Deo, Krishnamoorthy e Langston (1987). No entanto, não foram realizados experimentos computacionais que permitissem aferir a qualidade da formulação. Neste mesmo trabalho o autor prova não haver algoritmo de aproximação absoluta para o GMLP. Recentemente, De Giovanni et al. (2013) considerou o problema de minimização do comprimento dos fios utilizados nas redes, com restrições quanto ao número de trilhas utilizadas. Um algoritmo genético e um procedimento *branch-and-cut* foram propostos e testados em mais de trezentas instâncias da literatura. Especificamente, o algoritmo genético foi utilizado para estabelecer um limitante superior para o número de trilhas necessárias e o procedimento *branch-and-cut* foi utilizado para minimizar o comprimento dos fios. Para algumas instâncias foram obtidos resultados ruins com *gap* alto ou sequer foi possível resolver a relaxação linear em uma hora.

3. Método Proposto

A heurística proposta neste trabalho se baseia na representação do problema por grafos e é composta por dois métodos de pré-processamento, geração da solução inicial e aprimoramento por busca local. O Algoritmo 1 apresenta uma visão geral da heurística proposta. Nas seções seguintes, cada um dos componentes da heurística são descritos.

Conforme Yanasse (1997a), na representação do GMLP por grafos, vértices representam as redes e arestas indicam que duas redes compartilham alguma porta em comum. No circuito de exemplo da Figura 1(a) as redes $R2$ e $R4$ compartilham a porta $P1$, portanto, no grafo existe uma aresta entre os vértices 2 e 4, conforme pode ser observado na Figura 3(a).

Dois diferentes métodos de pré-processamento (linhas 1 e 3) são aplicados no intuito de diminuir as dimensões do problema, sem que seja causado algum prejuízo à qualidade da solução final, gerando uma matriz reduzida M_r. Após à criação do grafo original (linha 3), um grafo reduzido G_r é gerado. Posteriormente, uma solução inicial é gerada por meio da aplicação da Busca em Largura ao grafo reduzido (linha 4). Esta solução inicial consiste da lista S que contém a ordem em que os vértices foram explorados.

A lista de vértices/redes é então transformada em uma sequência de portas ϕ (linha 5) que é utilizada para determinar a matriz permutada $\bar{M}_\phi$ (linha 6). Em seguida, dois procedimentos de busca local (linhas 8 e 9) são aplicados à solução corrente enquanto houver melhoria da mesma, no intuito de aprimorá-la por meio da compactação de redes individuais. Ao final, a matriz $\bar{M}_\phi$ com a solução é retornada.

```
   Entrada: Matriz $M$, conjunto de portas $P$
 1 $M_r \leftarrow$ PreProcessamentoDominancia($M$);
 2 $G \leftarrow$ ConstroiGrafo($M_r$);
 3 $G_r \leftarrow$ PreProcessamentoGemeos($G$);
 4 $S \leftarrow$ BFS($G_r$);
 5 $\phi \leftarrow$ SequenciamentoPortas($S, P$);
 6 $\bar{M}_\phi \leftarrow$ GeraMatriz($M, \phi$);
 7 repita
 8   | BuscaLocal1($\bar{M}_\phi$);
 9   | BuscaLocal2($\bar{M}_\phi$);
10 até não haver melhoria em $\bar{M}_\phi$;
11 retorna $\bar{M}_\phi$;
```

Algoritmo 1. Visão geral da heurística proposta.

3.1. Pré-processamento de Portas Dominadas

De acordo com Yanasse e Senne (2010), uma porta Pi *domina* outra porta Pj qualquer, se e somente se o conjunto de redes que utilizam a porta Pj for um subconjunto das redes que utilizam a porta Pi. Portas dominadas podem ser desconsideradas no processo de resolução do problema e sequenciadas na solução final logo após as portas que as dominam, sem prejuízo à solução, uma vez que ambas podem utilizar as mesmas trilhas.

Como exemplo, no circuito da Figura 1(b), $P2$ é utilizada pelo conjunto de redes $\{R1, R4, R5\}$, $P4$ por $\{R4, R5\}$, $P3$ pelo conjunto $\{R2, R3, R6\}$ e $P6$ por $\{R2, R6\}$. Portanto, $P4 \subset P2$ e $P6 \subset P3$. Logo $P2$ domina $P4$ e $P3$ domina $P6$. O problema original possui 6 portas, porém, $P2$ e $P4$ podem ser consideradas como uma única porta, assim como $P3$ e $P6$. Desta maneira o problema reduzido conta apenas com 4 portas.

A verificação de dominância entre portas exige que para cada par de portas sejam comparados os conjuntos de redes que as contém. Consequentemente a complexidade deste procedimento é limitada por $O(n^2m)$.

3.2. Pré-processamento de Vértices Gêmeos

Novamente de acordo com Yanasse e Senne (2010), considerando a representação do GMLP por grafos, se dois vértices adjacentes entre si compartilham do mesmo conjunto de vértices adjacentes, estes são considerados *vértices gêmeos*. Tais vértices podem ser considerados como um só, reduzindo o tamanho do problema novamente sem prejuízo à solução, uma vez que as respectivas redes são similares.

A Figura 3(a) apresenta a representação do circuito da Figura 1(a) como um grafo. Os vértices 1 e 5 são adjacentes aos vértices 3 e 4, e também são adjacentes entre si. Estes são vértices gêmeos e, portanto, podem ser considerados como um único vértice, reduzindo o tamanho do grafo, vide Figura 3(b).

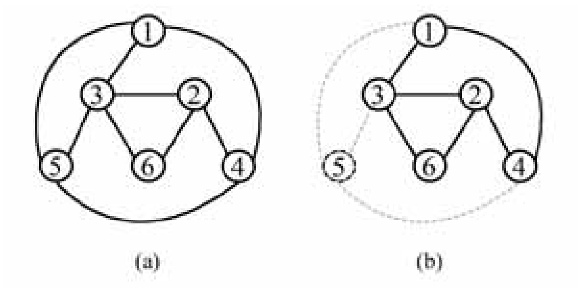

Figura 3. Representação do problema em grafo (a), em que os vértices 1 e 5 são gêmeos e (b) grafo reduzido.

A detecção de vértices gêmeos requer o exame e comparação da vizinhança de todos os pares de vértices, portanto, sua complexidade é limitada por $O(m^3)$.

3.3. Busca em Largura

Uma estratégia efetiva para solução do GMLP e problemas equivalentes é sequenciar as portas levando em consideração o relacionamento entre as redes, conforme estabelecido por Yanasse (1997b). Partindo deste princípio, a heurística proposta gera uma solução inicial que relaciona as redes que possuem algum grau de similaridade e que posteriormente diminuam o número de trilhas necessárias, guiando uma busca no grafo por meio de um critério guloso.

Uma Busca em Largura (ou BFS, do inglês *Breadth-First Search*) é realizada no grafo para este fim. No intuito de que as redes com menos portas sejam sequenciadas preferencialmente e de maneira contígua, a BFS empregada utiliza como critério de escolha do próximo vértice a ser explorado a quantidade de portas que a rede possui. Desta maneira, a BFS inicia a exploração do grafo a partir do vértice correspondente à rede com menor número de portas, e usa esta mesma regra gulosa para determinar a ordem em que os demais vértices da vizinhança serão explorados. Ao final da BFS, obtêm-se uma sequência S com a ordem de exploração dos vértices.

A Figura 4(a) mostra o grafo correspondente ao circuito da Figura 1(a), com os vértices gêmeos 1 e 5 unidos pelo pré-processamento e com os respectivos números de portas representados ao lado dos vértices. Note que, como as portas $P4$ e $P6$ foram dominadas, o número de portas das redes $R2$, $R4$ e $R6$ foi decrementado, já que as mesmas utilizavam alguma das portas dominadas. Começando pelo vértice de menor número de portas, o vértice 6, a Figura 4(b) exemplifica a primeira iteração da BFS com a exploração do vértice 6 e de sua vizinhança. À medida que os vértices são visitados, os mesmos são inseridos na fila S, que no fim, Figura 4(d),gera a solução para esta etapa do algoritmo.

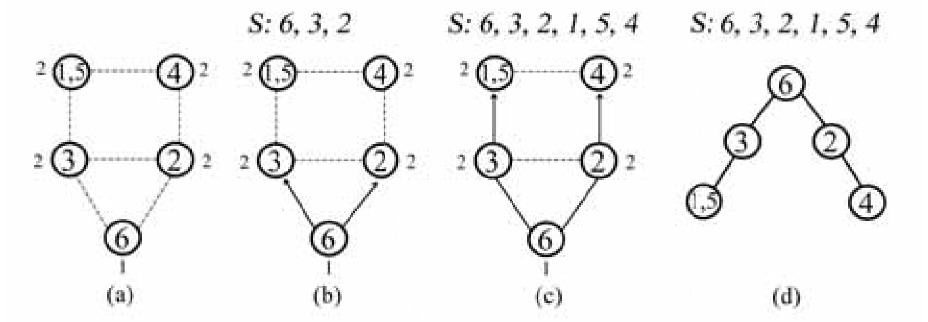

Figura 4. Aplicação da Busca em Largura proposta ao circuito da figura 1.

Claramente, o critério utilizado para guiar a busca não necessariamente leva à soluções ótimas. Com efeito, a busca em largura e o critério guloso de explorar preferencialmente redes com menos portas propostos neste trabalho compõem uma heurística, que, de acordo com os resultados reportados, produz boas soluções para o GMLP.

A busca em largura proposta requer que a vizinhança de cada vértice esteja ordenada de forma não decrescente de número de portas por rede. Considerando a representação do grafo por listas de adjacências, a complexidade deste procedimento é limitada por $O(m^2 \; log \; m)$.

3.4. Sequenciamento de Portas

A sequência de redes gerada pela BFS não é a solução para o GMLP. Para obter a sequência de portas necessária para solução do problema, foi utilizado o método apresentado por Becceneri et al. (2004). O princípio deste método é manter a relação estabelecida na sequência de redes, de maneira que portas não relacionadas a determinadas redes não são sequenciadas prematuramente, o que exigiria um maior número de trilhas na solução.

Este método percorre a sequência de redes gerada anteriormente, examinando-as. A cada nova rede examinada, busca-se entre as portas ainda não sequenciadas aquelas que são utilizadas por todas as redes já examinadas até aquele instante, incluindo-as na solução. Nesta etapa, as portas dominadas voltam a ser consideradas, a fim de se obter uma solução completa para o problema. As portas dominadas são sequenciadas logo após as portas que as dominam.

Considerando a sequência de exploração obtida pela BFS, $S = [6, 3, 2, 1, 5, 4]$, o sequenciamento de portas considera inicialmente as portas, $P1$, $P2$, $P3$ e $P5$. Quando o conjunto de vértices {6, 3, 2} é examinado, a porta $P3$ pode ser sequenciada, seguida pela porta dominada $P6$, como mostra a Figura 5(a). Posteriormente, na Figura 5(b), com o exame dos próximos vértices 1 e 5, a porta $P5$ é sequenciada. Por fim, com a adição do vértice 4 ao conjunto de vértices examinados, as portas $P1$ e $P2$ finalmente podem ser sequenciadas, seguidas pela porta dominada $P4$, vide Figura 5(c). Assim, obtém-se uma sequência de portas ϕ: [3, 6, 5, 1, 2, 4], ou seja, uma solução para o problema, como mostrado no circuito da Figura 1(c).

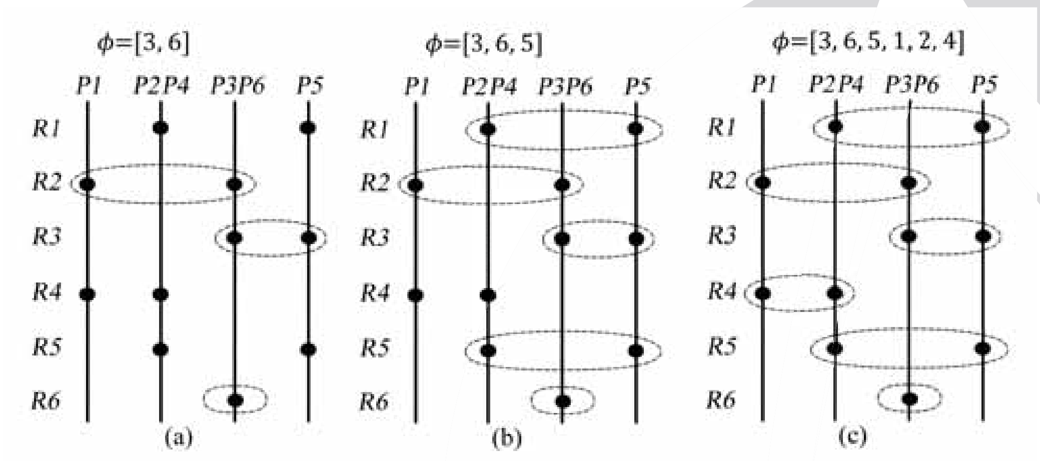

Figura 5. Sequenciamento de portas. As elipses pontilhadas indicam as redes já examinadas.

Este método para sequenciamento de portas requer a comparação do conjunto de redes já examinadas e o conjunto de redes que contêm cada porta, podendo ser executado em tempo $O(mn)$.

3.5. Aprimoramento da Solução

Intuitivamente, quando pensamos na otimização do leiaute de uma matriz de portas e na diminuição da área do circuito implementado, uma solução seria aproximar as portas de uma mesma rede tanto quanto possível. Como na Figura 1(c), para as redes $R1$ e $R5$, poderia ser interessante que as portas $P2$ e $P4$ fossem sequenciadas antes da porta $P1$, uma vez que esta última porta é cruzada por ambas as redes e, portanto, exige maior comprimento de fios. Todavia, minimizar o número de trilhas não é equivalente a reduzir o comprimento das mesmas, como mostrado por Linhares e Yanasse (2002).

Apesar desta não equivalência entre os problemas, este raciocínio é utilizado como regra heurística em uma busca local para tentar diminuir o número máximo de trilhas exigido pelo gargalo do problema. Mais detalhadamente, em uma solução do GMLP há uma ou mais portas com soma máxima de conexões e cruzamentos – como não é possível reduzir o número de conexões de uma porta, a idéia é diminuir a quantidade de fios que a cruzam. É importante ressaltar que o objetivo não é minimizar o comprimento total dos fios utilizados, mas sim, compactar as redes que cruzam os gargalos do problema, o que equivale a reduzir os 1's consecutivos das colunas de maior soma na matriz $\bar{M}_\phi$. Desta forma, a busca local tenta compactar algumas redes, agrupando suas portas em duas etapas: uma movendo as portas para a direita e outra movendo-as para a esquerda. Todos os movimentos que não gerem piora na qualidade da solução são realizados.

A Figura 6(a) exibe o circuito dos exemplos anteriores com a solução ϕ: [3, 6, 5, 1, 2, 4], obtida pelo sequenciamento de portas. Para fins de ilustração da busca local, consideremos a rede $R2$: Na primeira etapa da busca local, haverá uma tentativa de realocação das primeiras portas da rede ($P3$ e a porta dominada por ela, $P6$), para a direita, logo antes da porta $P1$, a fim de unir as portas conectadas pela rede. No entanto, como mostra a Figura 6(b), tal alteração causa um aumento no custo da solução, de 4 para 5 trilhas, logo, a sequência de portas ϕ_2: [5, 3, 6, 1, 2, 4] é descartada. Na segunda etapa, haverá uma tentativa de realocar as portas da mesma rede para a esquerda, logo depois da primeira porta da rede, desta forma, trocando a ordem das portas $P1$ e $P5$ obtém-se a solução ϕ_3: [3, 6, 1, 5, 2, 4] de custo 4, também. Assim o algoritmo passa a considerar a solução ϕ_3 como solução final com o intuito de, nas próximas iterações, melhorar o custo da solução.

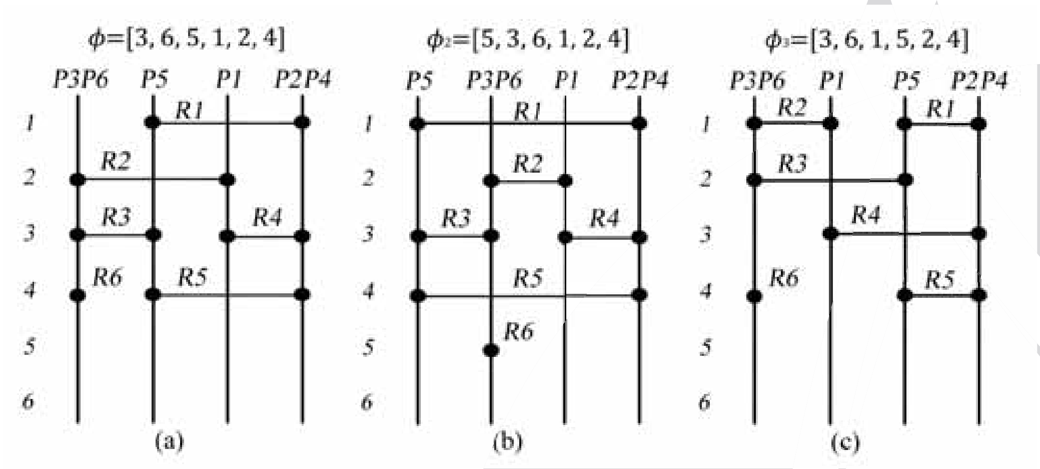

Figura 6. Aplicação da busca local.

Os dois procedimentos de busca local possuem a mesma complexidade computacional. Ambos exigem que haja uma tentativa de realocação das portas de cada rede e posterior avaliação da solução, o que limita a complexidade computacional por $O(m^2n^2)$. Todavia, na ausência de um valor específico para o número máximo de portas em qualquer rede, o cálculo realizado considera que este número seja n, muito embora este não seja o caso na prática, de acordo as instâncias reais utilizadas e com os resultados reportados a respeito do tempo de execução da heurística.

4. Experimentos Computacionais

Os experimentos computacionais foram realizados em um computador com processador *Intel i5 Quad Core* de 3.2 GHz com 16 GB RAM sob o sistema operacional Ubuntu 12.4.1. O código da heurística proposta foi escrito em C++, compilado com g ++ 4.4.1 e a opção de otimização -O3. Três conjuntos de instâncias foram utilizados nas experiências computacionais, e dois métodos diferentes da literatura foram utilizados nas comparações. Entre os conjuntos de instâncias, há um conjunto de instâncias específicas do GMLP e dois conjuntos de instâncias MOSP (*Minimization of Open Stacks Problem*), um problema industrial de formulação equivalente à formulação do GMLP, conforme mencionado na Seção 1. Onde mencionado, o *gap* (ou distância percentual) é calculado como *100×(valor da solução heurística) / valor de referência*). Os tempos de execução não são comparados nestes experimentos pois, como os métodos foram executados em diferentes arquiteturas, uma comparação justa não se torna possível.

4.1. Instâncias Reais GMLP

O primeiro conjunto contém 25 instâncias GMLP reais de empresas asiáticas, introduzidas por Hu e Chen (1990). As dimensões variam entre 7 linhas e colunas a 202 linhas e 141 colunas. A Tabela 1 apresenta os resultados (expressos em número de trilhas) e os tempos de execução (expressos em milissegundos) da heurística proposta para estas instâncias. Os melhores resultados para este conjunto, obtidos pela Busca Predatória de Linhares (1999), também são apresentados como valores de referência.

Tabela 1. Resultados para instâncias reais GMLP.

Instância	Busca Predatória	Heurística Proposta	
	Resultado	Resultado	Tempo (ms)
v4000	5	6	0,44
v4050	5	6	0,59
v4090	10	10	1,33
v4470	10	10	4,43
vc1	9	9	7,82
vl	3	3	0,50
vw1	4	4	7,67
vw2	5	5	0,57
w1	4	5	0,79
w2	14	14	4,44
w3	21	22	18,33
w4	32	29	55,12
wan	6	6	2,68
wli	4	4	0,92
wsn	8	8	0,75
x0	11	12	3,24
x1	5	5	0,29
x2	6	6	0,30
x3	7	7	0,45
x4	2	2	0,26
x5	2	2	0,34
x6	2	2	0,74
x7	4	4	0,68
x8	4	4	0,74
x9	4	4	1,21

A heurística proposta foi capaz de, em uma fração de segundo, alcançar 19 dos melhores resultados obtidos pela Busca Predatória de Linhares (1999), e melhorar um deles. No entanto, a melhor solução para a instância *w4* (27) foi determinada previamente pelo Algoritmo Genético Construtivo apresentado por Oliveira e Lorena (2002). Em outros cinco casos, piores soluções foram obtidas, todos diferindo por apenas uma trilha adicional.

4.2. Instâncias Reais MOSP

Este conjunto de instâncias, fornecido pelo *SCOOP Consortium* (2009), contém 187 instâncias MOSP reais de duas empresas européias. Entretanto, a maioria destas instâncias são muito pequenas (por exemplo, 2 linhas e colunas), possuindo soluções triviais. Deste conjunto, 15 instâncias, com dimensões que variam de 10 linhas e colunas a 202 linhas e 141 colunas foram selecionadas para serem usadas nos experimentos. A Tabela 2 apresenta os resultados utilizando a mesma configuração da tabela anterior, porém, agora *Wood. A (4)* indica um grupo de quatro instâncias e *Wood. B (11)* indica um grupo de 11 instâncias, portanto, os valores médios são apresentados. Os valores de referência são aqueles obtidos pelo algoritmo genético (GA) proposto por De Giovanni et al. (2013).

Tabela 2. Resultados para as instâncias reais MOSP.

Instância	AG	Heurística Proposta	
	Resultado	Resultado	Tempo (ms)
Wood. A (4)	5,75	5	0,63
Wood. B (11)	5,73	5,91	0,69

Para as instâncias agrupadas (*Wood. A* e *Wood. B*) não são reportados em outros trabalhos os valores individuais de cada uma das instâncias, o que impossibilita uma comparação detalhada. Os melhores resultados médios anteriores para as instâncias do grupo *Wood. A* foram melhorados pela heurística proposta, ao passo que para o grupo *Wood. B*, o *gap* foi 3,14%, ou 0,18 trilha adicional na média – o que equivale a duas trilhas adicionais considerando todo o conjunto. Mais uma vez, os tempos de execução são baixos, sendo o máximo menos de 0,02 segundos.

4.3. Instâncias Artificiais MOSP

Um terceiro conjunto de 150 instâncias artificiais MOSP grandes também foi considerado, disponível em `http://www.decom.ufop.br/marco/pesquisa/problem-instances/`. Estes casos foram gerados aleatoriamente, não possuindo nenhuma das estruturas específicas apontadas por Yanasse e Senne (2010) que possam facilitar a solução. As dimensões variam entre 150 linhas e colunas a 200 linhas e colunas. Na Tabela 3, *Random-m-n-d* identifica cada conjunto de 10 instâncias, em que m significa o número de redes, n o número de portas e d é o número de redes por porta, isto é, a demanda fixa para cada porta. Os valores médios de referência são as soluções comprovadamente ótimas ou o limitante inferior reportados por Chu e Stuckey (2009), já que nenhum dos métodos da literatura sobre o GMLP foram testados anteriormente usando estas instâncias.

Tabela 3. Resultados para as instâncias artificiais MOSP.

Instância	Solução	Heurística Proposta	
	Ótima	Resultado	Tempo (ms)
Random_150_150_2	25,90*	31,80	63,06
Random_150_150_4	61,60	67,90	243,66
Random_150_150_6	93,20*	98,70	577,61
Random_150_150_8	111,70*	118,80	769,03
Random_150_150_10	123,90*	129,80	970,50
Random_175_175_2	30,30	36,00	113,12
Random_175_175_4	73,70	81,30	377,44
Random_175_175_6	107,70	115,70	813,65
Random_175_175_8	128,30*	135,90	1313,20
Random_175_175_10	143,50*	150,40	1572,03
Random_200_200_2	36,00	44,00	144,65
Random_200_200_4	84,30	91,80	477,34
Random_200_200_6	121,50	129,00	1223,08
Random_200_200_8	147,10	155,10	1813,38
Random_200_200_10	162,80*	172,00	2000,83

*Solução ótima comprovada.

Das 150 instâncias contidas neste conjunto, apenas para 91 delas foram obtidas soluções comprovadamente ótimas pelo método exato de Chu e Stuckey (2009), desenvolvido para solução do MOSP. Em uma arquitetura semelhante à utilizada nos experimentos deste trabalho, o tempo de execução médio foi de 3,78 horas por instância.

O *gap* total obtido pelo método proposto em relação aos valores de referência, é de 7,35%. Considerando que estas são instâncias maiores e sem estruturas especiais, tal marca é considerável. Os tempos de execução tiveram um aumento, rompendo a casa de dois segundos. Porém, apesar do aumento destes valores, eles ainda são considerados baixos e desejáveis, haja vista que se trata de uma heurística e dadas as dimensões das instâncias.

5. Conclusões e Trabalhos Futuros

O Problema de Determinação do Leiaute de Matrizes de Portas é um problema NP-difícil com aplicação industrial no projeto de circuitos VLSI que também modela com sucesso uma variedade de problemas de teoria dos grafos e de escalonamento. O objetivo do mesmo é, dada uma matriz de portas lógicas e redes que conectam estas portas, determinar uma permutação das portas (ou seja, um leiaute para matriz de portas) de tal modo que a área necessária para implementar um circuito eletrônico seja minimizada. Deste modo, estes circuitos podem ser produzidos de forma mais barata e mais rápida.

Este trabalho propõe uma heurística de duas fases. Na primeira fase o problema é modelado como um grafo com estrutura específica e usa a bem conhecida busca em largura para gerar um sequenciamento das redes do problema. Na segunda fase, a partir deste sequenciamento, a heurística encontra a solução para o problema de fato, uma permutação das portas. No intuito de melhorar a solução gerada, dois procedimentos de busca local são aplicados para compactar determinados conjuntos de componentes do circuito.

Experimentos computacionais abrangentes envolvendo 190 instâncias de três conjuntos diferentes, incluindo instâncias reais, foram apresentados. A heurística proposta foi capaz de igualar uma boa parte dos melhores resultados disponíveis, e melhorar alguns deles. Foram reportados também experimentos com um conjunto de instâncias inédito no contexto do problema abordado.

Os trabalhos futuros serão concentrados em melhorar a busca local utilizada, experimentando diferentes técnicas e também no desenvolvimento de um método exato para ajudar a orientar a busca heurística, transformando o método em uma heurística híbrida, ou *matheuristic*.

A heurística sugerida pode ser incorporada a métodos exatos, pois fornece um limite superior razoável para o problema, ou pode ser utilizada diretamente como uma opção viável para obter rapidamente boas soluções para o GMLP.

6. Agradecimentos

Esta pesquisa foi apoiada pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq (processo 441565/2014-0) e pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES (processo 5157/11).

Referências

Becceneri, J.C., Yanasse, H. H. e Soma, N. Y. (2004), A method for solving the minimization of the maximum number of open stacks problem within a cutting process. *Comput. Oper. Res.*, 31(14):2315–2332.

Chen C. Y. R. e Hou C. Y. (1988), A new algorithm for cmos gate matrix layout. In Computer-Aided Design, 1988. ICCAD-88. *Digest of Technical Papers., IEEE International Conference on*, pages 138–141.

Chen S. J. e Hu, Y. H. (1990). GM-learn: an iterative learning algorithm for cmos gate matrix layout. *Computers and Digital Techniques, IEE Proceedings*, 137(4):301–309.

Chu, G. e Stuckey, P. J. (2009). Minimizing the maximum number of open stacks by customer search. In Proceedings..., volume 5732 of *Lecture Notes in Computer Science*, pages 242–257, Berlin. INTERNATIONAL CONFERENCE ON PRINCIPLES AND PRACTICE OF CONSTRAINT PROGRAMMING, Springer.

De Giovanni, L., Massi, G., Pezzella, F., Pfetsch, M. E., Rinaldi, G. e Ventura, P. (2013), A heuristic and an exact method for the gate matrix connection cost minimization problem. *International Transactions in Operational Research*, 20(5):627–643.

Deo, N., Krishnamoorthy, M. S. e Langston, M. A. (1987), Exact and approximate solutions for the gate matrix layout problem. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 6(1):79–84.

Hu, Y. H. e Chen, S. J. (1990), GM-plan: a gate matrix layout algorithm based on artificial intelligence planning techniques. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 9(8):836–845.

Hwang, D. K., Fuchs, W. K. e Kang, S. M. (1987). An efficient approach to gate matrix layout. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 6(5):802–809.

Kashiwabara T, Fujisawa T. (1979), NP-completeness of the problem of finding a minimum clique number interval graph containing a given graph as a subgraph. *Proceedings of the 1979 IEEE International Symposium on Circuits and Systems*, Tokyo, Japan, p.657–60.

Linhares, A. e Yanasse, H. H. (2002). Connections between cutting-pattern sequencing, VLSI design, and flexible machines. *Comput. Oper. Res.*, 29(12):1759–1772.

Linhares, A. e Yanasse, H. H. e Torreão J. R. A. (1999), Linear gate assignment: a fast statistical mechanics approach. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(12):1750–1758.

Linhares, A. (1999), Synthesizing a predatory search strategy for VLSI layouts. *Evolutionary Computation, IEEE Transactions on*, 3(2):147–152.

Mendes, A. e Linhares, A. (2004), A multiple-population evolutionary approach to gate matrix layout. *Int. J. Systems Science*, 35(1):13–23.

Oliveira A. C. M. e Lorena, L. A. N. (2002), A constructive genetic algorithm for gate matrix layout problems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(8):969–974.

SCOOP (Sheet Cutting and Process Optimization for Furniture Industry) Consortium (2009). Disponível em: http://www.scoop-project. net. Acessado em 19 de Abril de 2015.

Shahookar, K., Khamisani, W., Mazumder, P. e Reddy, S. M. (1994), Genetic beam search for gate matrix layout. *Computers and Digital Techniques, IEE Proceedings on*, 141(2):123–128.

Smith B. e Gent, I. *Constraint modeling challenge*. In Barbara Smith and Ian Gent, editors, The fifth workshop on modelling and solving problems with constraints, Edinburgh, Scotland, 2005. IJCAI.

Wing, O., Huang, S. e Wang, R. (1985), Gate matrix layout. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 4(3):220– 231.

Yanasse, H. H. e Senne, E. L. F. (2010), The minimization of open stacks problem: A review of some properties and their use in preprocessing operations. *European Journal of Operational Research*, 203(3):559–567.

Yanasse, H. H. (1997a), A transformation for solving a pattern sequencing in the wood cut industry. *Pesquisa Operacional*, 17(1):57–70.

Yannasse, H. H. (1997b), On a pattern sequencing problem to minimize the maximum number of open stacks. *European Journal of Operational Research* 100, 454–463.

Bibliography

Becceneri, J. C., Yanasse, H. H., and Soma, N. Y. (2004). A method for solving the minimization of the maximum number of open stacks problem within a cutting process. *Comput. Oper. Res.*, 31(14):2315–2332.

Carvalho, M. A. M. d. and Soma, N. Y. (2015). A breadth-first search applied to the minimization of the open stacks. *Journal of the Operational Research Society*, 66(6):936–946.

Chen, C. and Hou, C. (1988). A new algorithm for cmos gate matrix layout. In *Computer-Aided Design, 1988. ICCAD-88. Digest of Technical Papers., IEEE International Conference on*, pages 138–141.

Chen, S.-J. and Hu, Y. H. (1990). Gm-learn: an iterative learning algorithm for cmos gate matrix layout. *Computers and Digital Techniques, IEE Proceedings E*, 137(4):301–309.

Chu, G. and Stuckey, P. J. (2009). Minimizing the maximum number of open stacks by customer search. In *Proceedings...*, volume 5732 of *L*ecture Notes in Computer Science, pages 242–257, Berlin. Association for Constraint Programming, Springer.

Clerc, M. (2004). *Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem*, pages 219–239. Springer Berlin Heidelberg, Berlin, Heidelberg.

De Giovanni, L., Massi, G., Pezzella, F., Pfetsch, M., Rinaldi, G., and Ventura, P. (2013). A heuristic and an exact method for the gate matrix connection cost minimization problem. *International Transactions in Operational Research*, 20(5):627–643.

Deo, N., Krishnamoorthy, M., and Langston, M. (1987). Exact and approximate solutions for the gate matrix layout problem. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 6(1):79–84.

dos Santos, J. V. M. and Carvalho, M. A. M. (2015). Uma heurística aplicada a produção em microeletrônica. In *Proceedings of the XLVII Brazilian Symposium on Operations Research*, volume 1, pages 1463–1474, Porto de Galinhas, Brazil.

Faggioli, E. and Bentivoglio, C. A. (1998). Heuristic and exact methods for the cutting sequencing problem. *European Journal of Operational Research*, 110(3):564–575.

Glover, F. and Laguna, M. (1997). *Tabu Search.* Kluwer Academic Publishers, Norwell, MA, USA.

Goldbarg, E. F. G., de Souza, G. R., and Goldbarg, M. C. (2006). Particle swarm for the traveling salesman problem. In Gottlieb, J. and Raidl, G. R., editors, *E*volutionary Computation in Combinatorial Optimization, pages 99–110, Berlin, Heidelberg. Springer Berlin Heidelberg.

Goldbarg, M., Goldbarg, E., and Luna, H. (2015). *Otimização Combinatória E Meta-heurísticas: Algoritmos e Aplicações.* ELSEVIER EDITORA.

Gonçalves, J. F., Resende, M. G. C., and Costa, M. D. (2016). A biased random-key genetic algorithm for the minimization of open stacks problem. *International Transactions in Operational Research*, 23(1-2):25–46.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI. second edition, 1992.

Hong, Y.-S., Park, K.-H., and Kim, M. (1989). A heuristic algorithm for ordering the columns in one-dimensional logica arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(5):547–562.

Hu, Y. H. and Chen, S.-J. (1990). Gm plan: a gate matrix layout algorithm based on artificial intelligence planning techniques. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 9(8):836–845.

Hwang, D., Fuchs, W., and Kang, S. M. (1987). An efficient approach to gate matrix layout. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 6(5):802–809.

Kashiwabara, T. and Fujisawa, T. (1979). Np-completeness of the problem of finding a minimum-clique-number interval graph containing a given graph as a subgraph. In *Proc. Symposium of Circuits and Systems.*

Kennedy, J. (1997). The particle swarm: social adaptation of knowledge. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 303–308. IEEE.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, IEEE Service Center, Piscataway, NJ.

Kennedy, J., Kennedy, J., Eberhart, R., and Shi, Y. (2001). *Swarm Intelligence.* Evolutionary Computation Series. Morgan Kaufmann Publishers.

Linhares, A. (1999). Synthesizing a predatory search strategy for vlsi layouts. *Evolutionary Computation, IEEE Transactions on*, 3(2):147–152.

Linhares, A., Yanasse, H., and Torreão, J. (1999). Linear gate assignment: a fast statistical mechanics approach. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(12):1750–1758.

Linhares, A. and Yanasse, H. H. (2002). Connections between cutting-pattern sequencing, VLSI design, and flexible machines. *Comput. Oper. Res.*, 29(12):1759–1772.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Mendes, A. and Linhares, A. (2004). A multiple-population evolutionary approach to gate matrix layout. *Int. J. Systems Science*, 35(1):13–23.

Oliveira, A. and Lorena, L. (2002). A constructive genetic algorithm for gate matrix layout problems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(8):969–974.

Shahookar, K., Khamisani, W., Mazumder, P., and Reddy, S. (1994). Genetic beam search for gate matrix layout. *Computers and Digital Techniques, IEE Proceedings -*, 141(2):123–128.

Smith, B. and Gent, I. (2005). Constraint modeling challenge. In Bessiere, C., Hnich, B., Kiziltan, Z., and Walsh, T., editors, *T*he fifth workshop on modelling and solving problems with constraints, Edinburgh, Scotland. IJCAI 2005.

Wing, O., Huang, S., and Wang, R. (1985). Gate matrix layout. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 4(3):220–231.

Yanasse, H. H. (1997a). On a pattern sequencing problem to minimize the maximum number of open stacks. *European Journal of Operational Research*, 100(3):454–463.

Yanasse, H. H. (1997b). A transformation for solving a pattern sequencing in the wood cut industry. *Pesquisa Operacional*, 17(1):57–70.

Yanasse, H. H. and Senne, E. L. F. (2010). The minimization of open stacks problem: A review of some properties and their use in pre-processing operations. *European Journal of Operational Research*, 203(3):559–567.