

UNIVERSIDADE FEDERAL DE PELOTAS
Programa de Pós-Graduação em Computação



Tese

**SmartDR: Algorithms and Techniques for Fast Detailed Routing with Good
Design Rule Handling**

Stéphano M. M. Gonçalves

Pelotas, 2020

Stéphano M. M. Gonçalves

**SmartDR: Algorithms and Techniques for Fast Detailed Routing with Good
Design Rule Handling**

Tese apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Orientador: Felipe de Souza Marques

Co-Orientador: Leomar Soares da Rosa Jr.

Pelotas, 2020

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

G635s Gonçalves, Stèphano Machado Moreira

Smartdr : algorithms and techniques for fast detailed routing with good design rule handling / Stèphano Machado Moreira Gonçalves ; Felipe de Souza Marques, orientador ; Leomar Soares da Rosa Jr., coorientador. — Pelotas, 2020.
130 f. : il.

Tese (Doutorado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2020.

1. Roteamento detalhado. 2. Acesso a pinos. 3. Busca de caminhos. 4. Regras de projeto. 5. ISPD 2018 Contest. I. Marques, Felipe de Souza, orient. II. Jr., Leomar Soares da Rosa, coorient. III. Título.

CDD : 005

Stéphano M. M. Gonçalves

**SmartDR: Algorithms and Techniques for Fast Detailed Routing with Good
Design Rule Handling**

**Tese aprovada, como requisito parcial, para obtenção do grau de Doutor em
Ciência da Computação, Programa de Pós-Graduação em Computação,
Universidade Federal de Pelotas.**

Data da Defesa: 15 de janeiro de 2020

Banca examinadora:

Prof. Dr. Felipe de Souza Marques (Orientador, PPGC UFPEL)

Prof. Dr. Leomar Soares da Rosa Jr. (Co-Orientador, PPGC UFPEL)

Prof. Dr. Bruno Zatt (PPGC UFPEL)

Prof. Dr. Marcelo Johan (UFRGS)

Dr. Renato Fernandes Hentschke (Intel)

Resumo

GONÇALVES, Stèphano Machado Moreira. **SmartDR: Algorithms and Techniques for Fast Detailed Routing with Good Design Rule Handling**. 2020. 130p. Tese (Doutorado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2020.

O roteamento detalhado é uma das etapas mais desafiadoras e demoradas do projeto de circuitos integrados. A solução do roteamento deve obedecer a todas as regras de projeto para que o circuito possa ser corretamente fabricado. No entanto, o tratamento de regras de projeto pode ser muito desafiador, quanto a suas soluções algorítmicas e sua implementação, e pode facilmente levar a tempos de execução inviáveis. Para tornar a resolução do roteamento detalhado mais factível, ele é dividido em duas etapas, onde, na primeira, chamada de roteamento detalhado inicial, uma solução quase completa é obtida mediante a flexibilização das regras de projeto. Na segunda etapa as violações remanescentes de regras de projeto são resolvidas. No entanto, quanto mais o tratamento dessas regras é deixado para a segunda etapa, maior é a chance de elas não serem resolvidas completamente, e isto costuma ocorrer. Assim, é necessário enfrentar o desafio de lidar com o maior número possível de regras na etapa inicial sem comprometer o desempenho do roteamento. Dessa forma, este trabalho propõe um roteador detalhado inicial, chamado SmartDR, para atender essas necessidades, isto é, apresentar uma boa lida com regras de projeto ao mesmo tempo que mantendo um bom desempenho. As principais características do roteador, que atendem a esses objetivos, são técnicas de acesso a pinos e de busca de caminhos. Este trabalho propõe um novo método de acesso a pinos, em que os caminhos de acesso a pinos compartilham os mesmos recursos de roteamento e são legalizados e implementados dinamicamente. Também é proposto um novo algoritmo de busca de caminhos, baseado na busca A* com intervalos, o qual é ciente de várias regras de projeto. É proposto também um novo método para melhorar a função heurística da busca A* levando em consideração peculiaridades do roteamento detalhado, o que leva a um melhor desempenho. Utilizando os benchmarks da competição ISPD 2018, todos os métodos propostos foram avaliados separadamente e em conjunto no roteador proposto, o qual foi comparado com os roteadores estado-da-arte. Os experimentos mostram que as técnicas propostas contribuem para uma melhoria em desempenho e um bom tratamento de regras de projeto, assim como demonstra que o SmartDR é superior aos roteadores estado da arte nesses mesmos quesitos.

Palavras-chave: Roteamento detalhado. Acesso a pinos. Busca de caminhos. Regras de projeto. ISPD 2018 Contest.

Abstract

GONÇALVES, Stèphano Machado Moreira. **SmartDR: Algorithms and Techniques for Fast Detailed Routing with Good Design Rule Handling**. 2020. 130p. Thesis (Doctorate in Computer Science) – Postgraduate Program in Computer Science, Technology Development Center, Federal University of Pelotas, Pelotas, 2020.

Detailed routing is one of the most challenging and time-consuming steps of the design of integrated circuits. The routing solution must obey all of the design rules so that the circuit can be properly manufactured. However, design rule handling may be very challenging, regarding its algorithmic solutions and implementation, and may easily lead to unfeasible runtimes. In order to make the detailed routing resolution more feasible, it is divided into two steps, where in the first, called initial detailed routing, an almost complete solution is achieved by relaxing design rules. In the second step the remaining design rule violations are solved. However, the more these rules are left to be handled in the second stage, the greater is the chance that they will not be completely solved, and this usually occurs. Thus, it is necessary to face the challenge of dealing with as many rules as possible in the first step without compromising the runtime. Thus, this work proposes an initial detailed router, called SmartDR, to meet these needs, that is, to provide a good design rule handling while keeping a good runtime. The main features of the router that meet these goals are pin access and path search techniques. This work proposes a novel pin access method, in which the pin access paths share the same routing resources and are dynamically legalized and implemented. It is also proposed a new path search algorithm, based on A*-interval search, which is aware of several design rules. A new method to improve the A* heuristic function is also proposed, taking into account the peculiarities of detailed routing, which leads to a better runtime. Using ISPD 2018 Contest benchmarks, all proposed methods were evaluated separately and altogether in the proposed router, which was compared with state-of-the-art routers. The experiments show that the proposed techniques contribute to runtime and design rule handling improvement, as well as it demonstrates that SmartDR is superior to the state-of-the-art routers in these metrics.

Keywords: Detailed routing. Pin access. Path search. Design rules. ISPD 2018 Contest.

Summary

1 Introduction	13
2 Background and Context	19
2.1 Design Styles	19
2.2 Standard Cell Design Flow of Integrated Circuits	20
2.3 Routing	21
2.3.1 Global Routing	23
2.3.2 Detailed Routing	26
2.3.2.1 Design Rules	28
3 Detailed Routing Literature	32
3.1 Detailed Routing Approaches and Techniques	32
3.1.1 Channel and Switchbox Routing	33
3.1.2 Over-the-Cell Routing Using Gcells as Switchboxes	34
3.1.3 Sequential Routing Using Path Search for Long Connections	35
3.1.4 Track Assignment	37
3.1.5 Multicommodity Flow and Integer Linear Programming	39
3.1.6 Multilevel Routing	40
3.1.7 Search Space Representation	41
3.1.7.1 Grid-Based	41
3.1.7.2 Gridless	42
3.1.8 Multiple Patterning Compliant Detailed Routing	44
3.2 Path Search	46
3.2.1 Lee's algorithm	46
3.2.2 A* Algorithm	47
3.2.3 Line Probe Algorithms	52
3.2.4 A*-interval-based Path Search	53
3.2.5 Improved Heuristic Function for A*-based Path Search in Detailed Routing	55
3.2.6 Design Rule Aware Path Search	58
3.3 Pin Access	60
3.3.1 Escape Routing	61
3.3.2 Intra-cell with Conflict-Free Solution and Static Implementation	62
3.3.3 Inter-cell with Conflict-Free Solution and Dynamic Implementation	64
4 SmartDR Overview	66
4.1 Modeling Routing Information	66
4.2 Routing Flow	68
5 Pin Access	72
5.1 Calculating Pin Access Paths	74

5.1.1 Overall Procedure	74
5.1.2 Design Rule Handling.....	75
5.2 Dynamic Manipulation	77
5.3 Thick Metal Shape Detection.....	78
5.4 Comparison with Related Work	82
5.5 Experiments	83
5.6 Conclusions and Future Works	90
6 Path Search	91
6.1 Path Search Mechanics.....	91
6.2 Handling PAP Costs in the Source and Target Points.....	93
6.3 Tunnel Lowerbound.....	95
6.3.1 The Proposed Technique	95
6.3.2 Algorithm and Implementation.....	97
6.4 Design Rule Handling.....	99
6.4.1 Via Checking	100
6.4.1.1 Efficient Via Queries	101
6.4.2 Cut Spacing on Same-Path Vias	103
6.4.3 Minimum Area	104
6.5 Comparison with Related Work	105
6.5.1 DRAPS	105
6.5.2 Improved Lowerbounds for the A* Heuristic Function	107
6.6 Experiments	107
6.6.1 PAP Cost Aware Path Search	107
6.6.2 Tunnel Lowerbound.....	108
6.6.2.1 Comparison with (PEYER, 2009)	109
6.6.3 Design Rule Aware Path Search	112
6.7 Conclusions and Future Works	114
7 Comparison with State-of-the-Art Routers	115
8 Conclusions	122
References	122

Figure List

Figure 1- Flowchart of the design process of integrated circuits, emphasizing physical synthesis.	21
Figure 2- Illustration of a routed circuit layout.....	22
Figure 3– Illustration of the global routing	23
Figure 4– Illustration of a MRST (a), a RMST (b) and a Hannan grid (c), with a 4-pin net	24
Figure 5- Illustration of via shape configurations of a via library.	26
Figure 6- Picture of a standard cell in Cadence Innovus tool	27
Figure 7- Illustration of minimum edge rule.	28
Figure 8- Illustration of the parallel runlength rule.	29
Figure 9- Illustration of the parallel runlength rule with a polygon with multiple rectangles.....	29
Figure 10- Illustration of the cut spacing rule with adjacent via cuts constraint	30
Figure 11- Illustration of end-of-line rules	30
Figure 12 – Illustration of a switchbox routed	33
Figure 13- Illustration of routing each gcell as a switchbox and the problem involved in the crosspoint assignment.....	35
Figure 14 – Illustration of the Track Assignment problem.	37
Figure 15 – Illustration of the graphs used in the TA technique.....	38
Figure 16 – The V-shaped multilevel routing framework.	41
Figure 17 – The shape grid, proposed in BonnRoute	42
Figure 18 – Illustration of the search space representation using slit tree and interval trees	43
Figure 19 – Illustration of connection graph (a) and tile-based (b) (c) approaches....	43
Figure 20 – Illustration of the layout decomposition and stitch generation	45
Figure 21 – Illustration of some steps of the execution of Lee’s algorithm	47
Figure 22 – Pseudocode of A* algorithm.....	49
Figure 23 - Comparison of the search space of A* (a) and Lee’s algorithm (b).....	50
Figure 24- Illustration of the impact, in the number of search nodes, of the gap between $h(n)$ and $h^*(n)$	51
Figure 25 – Illustration of Mikami and Tabuchi’s algorithm.....	53

Figure 26 – Illustration of the principle of merging a group of redundant nodes into intervals	54
Figure 27 – Illustration of the interval expansion of Hetzel’s algorithm	55
Figure 28: Illustration of the expanded search nodes using $L1$ distance (a) and TL (b), using an A*-interval-based path search (as Hetzel’s algorithm), restricted by a tunnel of 3 rectangles	56
Figure 29- Illustration of the manipulation of the coefficients and the corresponding offset points (in red)	57
Figure 30- Illustration of the rectangle partitioning in a tunnel	58
Figure 31- Illustration of the pin access problem.	60
Figure 32- Illustration of an escape routing solution.	61
Figure 33- Illustration of <i>circuitclasses</i> (bottom) and their corresponding instances (top)	63
Figure 34- Illustration of PAP creation with (a) and without (b) violations	63
Figure 35- Illustration of inter-cell conflicts	64
Figure 36- Illustration of finding a non-conflicting pin access solution for the entire standard cell row.	65
Figure 37- Illustration of grid point occupancy criteria.	66
Figure 38- Illustration of trivial via cut handling (a), and the one adopted in SmartDR (b)	67
Figure 39- Flowchart of SmartDR detailed routing flow.	69
Figure 40: Illustration of pin access situations	73
Figure 41: Illustration of patch metal insertion and TMS creation	75
Figure 42: Illustration of same-pin PAP conflicts.	76
Figure 43: Illustration of the PAP blockage information and cache usage in metal1.	77
Figure 44: Illustration of our TMS detection algorithm	79
Figure 45- Pseudocode of the proposed TMS detection algorithm	81
Figure 46- Pseudocode of the method to create a CFS from the RS solution of SmartDR	85
Figure 47: Comparison of the CFS implementing 1, 2 and 3 via-PAPs, regarding total routing runtime (a) and number of failed searches (b) in Standard Routing step.	86
Figure 48: Comparison of different strategies of via-PAP implementation, and results in total routing runtime (a) and failed searches (b) in Standard Routing step.	88
Figure 49: Runtime results of PAP legality check with and without cache	88

Figure 50- Pseudocode of DRAPS.	92
Figure 51: Illustration of the issues of handling costs in the target points.....	94
Figure 52: Illustration of the <i>TL</i> technique. Ref points are red and yellow.	96
Figure 53- Pseudocode of the proposed algorithm to precompute the tunnel lowerbounds.	98
Figure 54- Pseudocode of the DRC function.	100
Figure 55: Via selection order and via blockage information (red dots).	101
Figure 56- Illustration of a situation where querying the grid with the via blockage information is not enough to avoid DRVs.	102
Figure 57- Illustration of the via check using the empty-space intervals stored by DRAPS.....	103
Figure 58: Illustration of a path search scenario with a cut-cut spacing violation within the path.	104
Figure 59: Illustration of situations of min area check.....	105

Table List

Table 1: Differences Between the Proposed Method and (NIEBERG, 2011).....	82
Table 2: Benchmark Information	84
Table 3: Comparison Between RS and CFS Approaches	86
Table 4: TMS Usage Results.....	89
Table 5: Comparison with design rule aware path search algorithms in literature...	106
Table 6: Results of Using PAP Costs in the Path Search.....	108
Table 7: Results of the Proposed Tunnel Lowerbound Technique	109
Table 8: Results of the method proposed in (PEYER, 2009).	110
Table 9: Comparison of relative results between TL and the method in (PEYER, 2009).	111
Table 10: Results of Design Rule Violations of DRAPS	113
Table 11: Detailed Design Rule Violation results over (KAHNG, 2018) and (CHEN, 2019b).	116
Table 12: Area Short and Total Violation Results.....	116
Table 13: Runtime Results.	116
Table 14: Results of WL, Vias and Out-of-Guide Usage	117
Table 15: Results of Wrong-Way WL, Off-Track Usage and Scores.	117
Table 16: ISPD18 Score Metrics.	119
Table 17: ISPD18 Score Results.....	120

List of Abbreviations and Acronyms

Application Specific Integrated Circuit	ASIC
Breadth-First Search	BFS
Computer Aided Design	CAD
Conflict-Free Solution	CFS
Cross Point Assignment	CPA
Depth-First Search	DFS
Design Rule Checking	DRC
Design Rule Violation	DRV
Field Programable Gate Array	FPGA
Global cell	gcell
Half-Perimeter-Wire-Length	HPWL
Hardware Description Language	HDL
Integer Linear Programming	ILP
Integrated Circuit	IC
International Symposium on Physical Design of 2018	ISPD18
Lookup Table	LUT
Maximum Weight Independent Set	MWIS
Multicommodity Flow	MCF
Pin Access Path	PAP
Rectilinear Minimal Spanning Tree	RMST
Resource Sharing	RS
Ripup-and-Reroute	RNR
Self-Aligned Multiple Patterning	SAMP
Thick Metal Shape	TMS
Track Assignment	TA
Wire-Length	WL

1 Introduction

As technology advances, integrated circuits have become increasingly complex. The shrinking of the transistors size, and consequently of other components, allowed the design of circuits with a huge number of logic gates, limited to very small physical spaces. In addition, the market has increased the demand for the technology, requiring greater agility in the production of integrated circuits. Therefore, due to their high complexity, the design of integrated circuits is performed by CAD (Computer Aided Design) tools, which automate the synthesis process of the circuits.

There are several design styles to perform the circuit synthesis. The most eminent design style is called standard cell. The standard cell synthesis process begins with the description of the logic behavior of the circuit using hardware description languages (HDL). After that, the set of logical functions that describe the circuit are optimized in the *logical synthesis* design step. Then, the *technology mapping* design step defines which available logic gates will be used to implement these logic functions. The next design step is *physical synthesis*, which aims to provide a geometrical description of the circuit. The physical synthesis is subdivided into two main steps: *placement* and *routing*. The placement determines the positions of the logic cells aiming to minimize some cost functions and alleviate the effort of the next design step. The routing design step obtains the wire routes needed to connect the components, while also minimizing some cost functions. Finally, a physical verification step is performed to ensure that all electrical and logical functionality are met.

Due to its high complexity, routing is subdivided into two stages: *global* and *detailed* routing. The global routing defines the areas in which the wires of nets should pass through, while controlling wire congestion to improve routability for detailed routing. Thus, the main purpose of global routing is to provide instructions for detailed routing, in addition to reducing its effort. The detailed routing finds the exact location of the wires inside the areas delimited by the global routing. Detailed routing must take into account a number of manufacturing constraints, referred as design rules.

In the old days, when there was no global routing, the first routers relied entirely on Lee's algorithm (LEE, 1961) or the line probe methods of (MIKAMI, 1968) and (HIGHTOWER, 1969). However, the circuit's layout had specific characteristics that could be exploited in order to perform a more efficient routing. The cells were grouped in rows, and between each row there was a space, called *channel*, which was used for routing between two adjacent rows. Thus, the *channel* (HASHIMOTO; DEUTSCH,

1971, 1976) and *switchbox* (JOBBANI; LUK; MAREK-SADOWSKA; HAMACHI, 1986, 1985, 1985, 1984) routers emerged to solve the routing problem more efficiently. As the complexity of the circuits increased, global routing was proposed to alleviate the routing problem. Later, this complexity reached a level such that the channels had to be very large in order to support all wires, and this was a problem, since the circuit area had to increase. Thereby, it was necessary to change the circuit layout paradigm, removing the channels between the rows of cells, and assigning the routing to take place above the cells. This paradigm is called *over-the-cell* routing, and it is used nowadays.

With the end of channel routing, several routing approaches have emerged. In the time of channel routers, global routing divided the circuit layout in irregular regions, which were channels and switchboxes, and most connections were performed inside each region. In over-the-cell routing, global routing partitions the layout in a regular grid of global cells (gcells), and most connections must cross the boundaries of the gcells. One approach to perform detailed routing in this new paradigm is to route each gcell of a global route separately. In order to do this, it is necessary to define the connecting interface between gcells. The problem to define such interfaces was performed as an intermediate step between global and detailed routing and was known as the Cross Point Assignment (CHANG, 2001) (KAO, 1995). Another new approach is the track assignment (BATTERYWALA, 2002), which was proposed as an intermediate step between global and detailed routing. It consists in assigning the segments of global routes to the routing tracks. It is followed by detailed routing, which connects the missing pieces of the nets. The detailed routing handling long connections with fast and optimal path search was introduced by Hetzel (HETZEL, 1998). In this approach, a single path search is performed for an entire path of gcells. This was possible due to the path search algorithm proposed (HETZEL, 1998), which combined the benefits of the maze search and line probe algorithms. At the first half of the past decade, the multilevel routing frameworks (LIN; HO; CONG; CONG, 2002, 2003, 2001, 2005) became popular. They consist in further decomposing the routing problem in even more levels of abstraction beyond global and detailed routing, also combining top-down and bottom-up methodologies.

Most recent detailed routing approaches combined previous techniques and specialized some of them. The multilevel routing frameworks did not persevere. Li et al. (2007) proposed a gridless router combining two common gridless routing

techniques, tile expansion and connection graphs. In (LI, 2011), the approach of (LI, 2007) was combined with track assignment (BATTERYWALA, 2002), which was aimed for grid-based routing. Ozdal (2009) proposed efficient algorithms for escape routing for dense pin clusters. Zhang et al. (2013) proposed RegularRoute, a detailed router that uses a track-assignment-based technique (BATTERYWALA, 2002) inside detailed routing, providing full connections, rather than the original track assignment, which was an intermediate step between global and detailed routing and which provided a partial solution. The GDRouter routing tool (ZHANG, 2012) combined a global router (XU, 2009) with RegularRoute (ZHANG, 2013) in an interleaved global and detailed routing procedure. Gester et al. (2013) proposed efficient data structures to handle design rules in a grid-based routing using a path search algorithm based on Hetzel's algorithm (HETZEL, 1998) for long connections. Jia et al. (2017) proposed a detailed router based on the multicommodity flow graph problem and used Integer Linear Programming (ILP) to solve the routing problem. TritonRoute (KAHNG, 2018) proposed an intra-layer parallel routing scheme and also solved it using ILP. In (SUN, 2018), a via location aware track assignment and a multi-thread two stage detailed routing algorithm is proposed. (CHEN, 2019b) proposed a technique to make the path search run in an optimized local graph, based on the obstacles inside the global routing guide. Considering the peculiarities of advanced technology nodes, many works have emerged addressing multiple-patterning compliant detailed routing (DING; YU; YUAN; GAO; MIRSAEEDI; MA; LIU, 2018, 2018, 2009, 2010, 2011, 2012, 2016).

Detailed routing also holds an important problem, which is pin access. The pin shapes are often irregular and not aligned on the routing grid, requiring special treatment when connecting with the wires. Also, this connection often leads to design rule violations (DRVs) between the wire and the pin, and within the pin access path itself. Since one of the main goals of detailed routing is to avoid DRVs, this problem deserves considerable attention. There are few works on literature that addressed the pin access problem. Nieberg (2011), proposed an approach that uses gridless pin access paths free of DRVs within the paths themselves. The method also explores redundancies in the cell instances such that it is not necessary to calculate pin access solutions for every placed cell instance, saving runtime. In (OZDAL, 2009), scape routing is proposed, aiming to provide pin access paths that possibly run out of the cell boundary, escaping from the pin access congestion, increasing routability. In (XU,

2016) and (XU, 2017), pin access considering self-aligned double patterning compliant detailed routing is proposed.

Almost all detailed routing approaches rely on path search algorithms. The goal of such algorithms is to find a path between two sets of points in a search space. There are two classes of path search algorithms in detailed routing: the maze search and the line probe algorithms. The maze search algorithms derive from Dijkstra's algorithm (CORMEN, 2001). They work on a graph, which is usually a grid, and expand the search point by point, while the line probe algorithms use line segments. Usually, the maze algorithms guarantee the optimal path, but they are slow, while the line probe algorithms are usually fast but do not guarantee the optimal path. The Lee's algorithm (LEE, 1961) is equivalent to the Dijkstra's algorithm in the specific context of grid graphs with uniform cost edges. Later, Rubin (RUBIN, 1974) used the A* search (HART, 1968) technique to speed up Lee's algorithm. The first line probe algorithm was proposed by Mikami and Tabuchi (1968). Later, Hightower (1969) proposed a modification on this algorithm. Finally, Hetzel (1998) united the advantages of both classes of algorithms, proposing an A*-based algorithm using intervals of grid graph vertices as search nodes. The algorithm was shown to be faster than A*, and it is optimal, unlike (MIKAMI, 1968) and (HIGHTOWER, 1969). Later, the algorithm was generalized in (PEYER, 2009) to handle more generic scenarios.

The mentioned path search algorithms are not aware of the design rules that detailed routing must attend to. These rules are commonly treated outside of the path search. Part of them are handled after routing, in post-processing steps. However, the attempt of solving DRVs after routing may either result in the inability of their resolution or in the creation of new violations. A good way to handle this situation is to make the path search algorithm aware of some design rules, so that it is able to provide DRV free paths in a correct-by-construction fashion.

There are few works in literature that propose design rule aware path search algorithms. MANA (CHANG, 2013) is aware of the minimum area design rule, but it is a maze algorithm, and such algorithms present high runtime. In (CHEN, 2019b), a maze algorithm that attends the minimum area rule is also proposed. It runs in a reduced graph, constructed on-the-fly, which mitigates the maze routing runtime problem, but there is still a time overhead to construct such graph. In (AHRENS, 2015), it is proposed a multi-label algorithm which also handled this design rule, and different-

mask rules for multiple patterning detailed routing approaches. The runtime was also high, making the algorithm worth to be used only in some circumstances.

Although detailed routing has been extensively studied in the past decades, there are still research possibilities. One of the biggest challenges of detailed routing is design rule handling. The routing solution must respect all design rules, since the circuit depends on it to properly work. Considering that design rule handling is hard, detailed routing is yet subdivided. The first step, called initial detailed routing, performs the majority of the effort, finding all routes and considering the simplest design rules. The last step solves the remaining DRVs and may perform some refinement on the routing solution. This subdivision is due to the fact that considering all design rules at once is hard and increases the likelihood of causing large runtime overheads. However, leaving more violations for the final step makes the DRV cleanup harder, and possibly unfeasible. Thus, it is ideal to try to solve as many violations as possible in the initial phase. However, eminent routers (GESTER; ZHANG, 2013, 2013) have shown to be unable to handle all design rules in the initial step, delegating industrial routers to perform DRV cleanup. Even so, the industrial routers are not always able to solve all violations. Also, runtime is another important objective to be optimized, since detailed routing is naturally time consuming, and making high effort to handle design rules is unfavorable for it.

Mantik et al. (2018), members of Cadence Design Systems, pointed another issue in part of the detailed routing works in the literature, which is the fact that they relied on small instances of the routing problem. Considering these issues, the Initial Detailed Routing Contest of the International Symposium on Physical Design of 2018 (ISPD18) (MANTIK, 2018) emerged, encouraging research in detailed routing field and bringing the first set of detailed routing academic benchmarks to the literature. Also, although the reality of the industry is probably far more advanced than in the literature, it is indeed pertinent to do research in detailed routing to provide public knowledge. Besides, although less likely, the academy research may even contribute to the industry.

Considering that time is an important metric for detailed routing, and having a design with no violations is essential, this work proposes an initial detailed router, called SmartDR, addressing these objectives. The key features of SmartDR that attend to these objectives are the pin access algorithm and path search techniques. The main contributions of this work are:

1. a novel pin access approach of flexible pin access path selection with resource sharing and dynamic legality check;
2. a design rule checking algorithm, used in pin access, to detect thick metal shapes that require higher spacing;
3. a new A*-interval-based design rule aware path search (DRAPS) algorithm capable of handling the minimum area rule, cut-to-cut spacing rules of same-path vias, and is aware of the via library, performing efficient via checking;
4. adaptations in the path search to properly incorporate costs in the source and target points to reduce pin access path usage, which impacts in via count and other routing metrics;
5. an efficient technique to improve the A* heuristic function (h) by making it aware of the global routing guides, providing a substantial speedup to the path search;
6. experimental results, using ISPD 2018 Contest benchmark suite (MANTIK, 2018), evaluating the proposed techniques, and comparing SmartDR with other state-of-the-art routers that were also tested using the same benchmarks. The results show that the proposed router is superior in runtime and design rule handling.

This work is structured as follows. Section 2 clarify the context of this work and presents some background information. Section 3 presents a review of the detailed routing literature. Section 4 presents an overview of SmartDR. Section 5 presents the proposed pin access method and the related experiments. Section 6 presents the proposed path search techniques and experiments. Section 7 presents the comparison of SmartDR with state-of-the-art routers and section 8 are the conclusions.

2 Background and Context

First, let us define some common nomenclature. An *integrated circuit* (IC) consists of *logic gates*, which are interconnected by wires. A *logic gate*, or *cell*, is the building block of the logical functionality of the circuit. Since it is a component that implements a logic function, it has inputs and outputs, which are called pins or terminals. An output pin carries the electrical signal through wires, which connect to input pins of other logic gates. This set of pins is called *net*. A *block*, or *logical block*, or *macro block*, is a circuit of logic gates that acts as a black-box component of the chip functionality, and commonly presents a functionality that is easily reusable. The term macro block is commonly used to designate a block with a large number of cells.

2.1 Design Styles

There are several ways to design integrated circuits. The design styles impact in time-to-market, cost and in the quality of the IC. Also, some design styles depend on the application of the IC.

The *Full Custom* design style allows the design to be highly customizable. For example, macro blocks can be placed anywhere in the chip. The high freedom in the design makes the IC more optimized. However, the development time is also high, due to a relative lack of automation, making the design more expensive. Also, due to the high manual interaction, this style is more susceptible to present errors. It is worth using when the IC production is high enough to compensate the large design effort, such as in microprocessors and FPGAs (Field Programmable Gate Arrays) (KAHNG, 2011a).

In *Gate Array* design style, all logic components have the same structure and a generic logical functionality, and there is no wiring connecting the components. The wire routing is performed after defining the application of the IC. Due to the standardization of the logic components, the only major design effort is in routing, and, consequently, time-to-market and design costs are favorable. However, since the placement of the components does not take into consideration routing, this may cause routing problems (KAHNG, 2011a).

Similarly to gate arrays, the *FPGAs* use standardized logic components, but also with prefabricated wiring. All logical functionality is configurable by the user. The design cost and time-to-market is even lower than in the gate array style. However, the

IC runs slower and dissipates more power than application specific integrated circuits (ASICs) (KAHNG, 2011a).

The *Standard Cell* design style is the more popular for ASICs. It uses a library of logical cells, and possibly some macro blocks, to implement the IC functionality. The cells have standardized height and are placed, in the chip, side by side in rows. Between the rows, power and ground rails feed the cells. The conception of the cell library may take substantial effort. Still, with design freedom reduced, this design style provides a good time-to-market and design cost (KAHNG, 2011a). The scope of this work is within standard cell-based design.

2.2 Standard Cell Design Flow of Integrated Circuits

The design of standard cell ICs is a complex procedure, containing many steps. Figure 1 presents a simplified scheme of the design flow. The procedure begins with the specification of the goals and the requirements of the system that the IC will implement. The logic behavior of the circuit is defined using hardware description languages (HDL). Some common HDLs are Verilog and VHDL.

In the *Logical Synthesis* step, a data structure called *netlist* is generated based on the circuit description. This structure stores all the information regarding the logical connectivity of the circuit. Then, the netlist is optimized by boolean algebra optimization techniques. Once the logical synthesis is finished, *Technology Mapping* step is started. This step consists in defining which available cells will be used to implement the logic functions that define the netlist.

The next design step is the Physical Synthesis, which aims to define a geometrical description of the circuit. The physical synthesis is subdivided in the following steps. *Partitioning* breaks up a circuit into smaller subcircuits or modules, which can each be designed or analyzed individually. *Floorplanning* determines the shapes and arrangement of subcircuits or modules, as well as the locations of external ports and IP or macro blocks. *Power and ground routing*, often intrinsic to floorplanning, distributes power and ground nets throughout the chip. *Placement* finds the spatial locations of all cells within each block, while minimizing the estimated wire-length (WL). *Clock network synthesis* determines the buffering, gating (e.g., for power management) and routing of the clock signal to meet prescribed skew and delay requirements. *Routing* provides the wire routes needed to connect the components, while also minimizing the WL and attending to other objectives (KAHNG, 2011b).

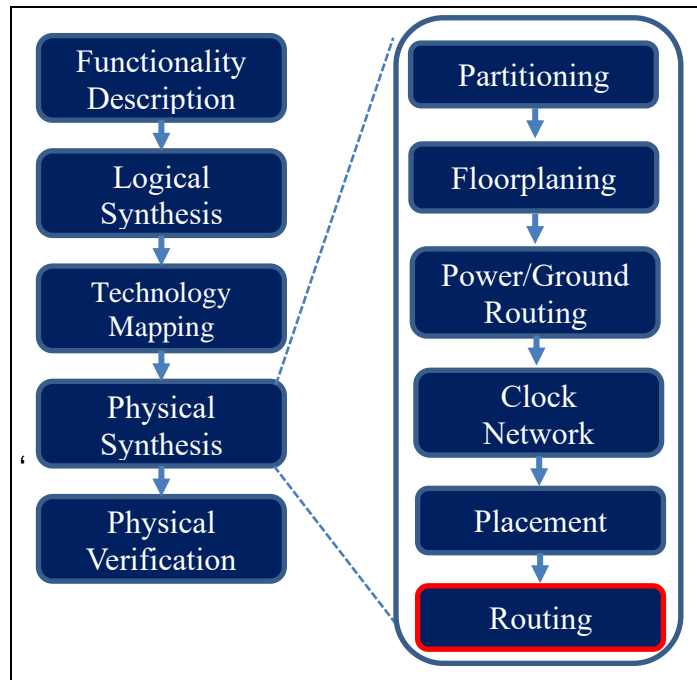


Figure 1- Flowchart of the design process of integrated circuits, emphasizing physical synthesis.

After the physical synthesis, the *Physical Verification* step is performed to ensure that all electrical and logical functionality are met. Here, a netlist is extracted from the layout and compared to the original netlist. Also, design rule checking (DRC) is performed to ensure all design rules are respected. The same way, electrical rule checking is performed to validate the correctness of the electrical signals and their timing.

2.3 Routing

The main goal of the routing problem is to determine the wire routes of all nets of the IC. Also, the routing solution *must* respect all design rules. The design rule attendance is necessary to ensure that the IC can be manufactured and will be electrically reliable.

The routing occurs in a tridimensional space with parallel planes, where each plane refers to a metal layer in which the wires are placed. Metal layers have a preferred routing direction, which is either horizontal or vertical. Adjacent layers have different preferred directions. This is to avoid capacitance effects of nearby parallel wires of adjacent layers and to improve routability. Between each metal layer, there is a cut layer. The cut layers hold the vias, which are the components that connect wires from different metal layers. This stack of metal and cut layers is placed above the standard cells. Figure 2 illustrates a section of a routed circuit layout.

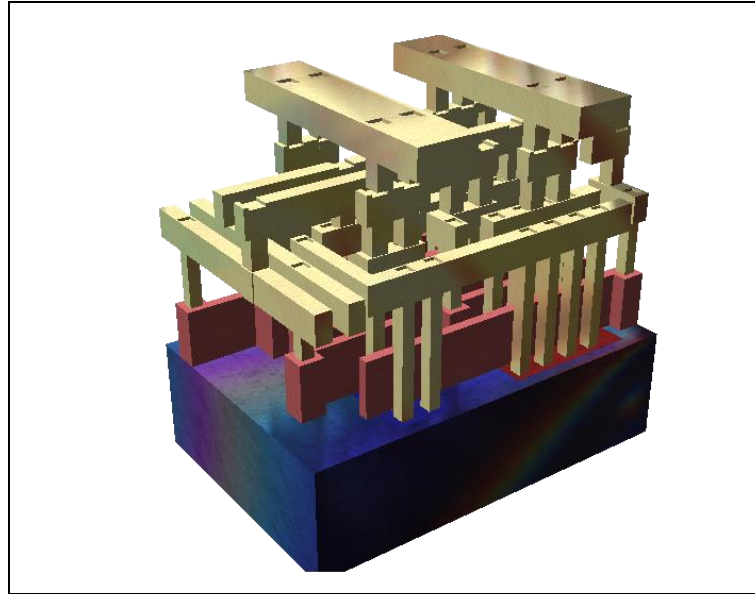


Figure 2- Illustration of a routed circuit layout. The red and blue layer belong to the standard cell, and the metal layers are above. Source: public domain.

Routing also tries to optimize some objective functions, such as runtime, WL, number of vias, timing and yield. The runtime is important because routing is a very time-consuming process, accounting for much of the time spent on physical synthesis. WL impacts power consumption, delay and signal degradation. The number of vias also contributes to signal degradation since the vias have high electrical resistance. Timing is related to the delay of the propagation of the electrical signal in the IC. Yield represents the likelihood of a circuit to be manufactured correctly, without presenting any defects. Some good practices may be beneficial to yield.

The number of wire connections that routing must complete may reach the millions. When modeled by a grid graph, the routing space may present billions of vertices (SCHULTE, 2012). This makes impracticable, in runtime terms, to perform the wire connections just by using path search algorithms, even the line probe ones, which are the fastest. Also, simply routing the wires without any planning will make many wires to unnecessarily congest routing areas needed to complete some wire connections, which compromises routability. Finally, the lack of planning will make the solution, even if possible to exist in a feasible time, to present low quality due to large wire detours, increasing WL, via count and timing.

These facts require routing to adopt a divide and conquer strategy. Thus, routing is divided into *global* and *detailed* routing. Global routing works on a simplified version of the problem and serves as an orientation to detailed routing. This separation in two steps is not always precise, since there are some works that propose multilevel routers

(LIN; HO; CONG; CONG, 2002, 2003, 2001, 2005), with more intermediate steps, or interleaved approaches (ZHANG, 2012), which will be mentioned in section 3.1.

2.3.1 Global Routing

The goal of global routing is to reduce the complexity of the problem by providing a pre-solution to detailed routing, which defines the final routing. Also, objective functions as WL, via count and timing are optimized. This complexity reduction occurs through a use of coarse-grained grid (Figure 3a and b), where each grid point covers many points of the real grid, used in detailed routing. Thus, each point represents an area, called the global cell (gcell). The grid can be both two-dimensional or three-dimensional. Each edge of the graph, which represents the grid, models the borders between two gcells. Each edge has a capacity, which represents the maximum number of wires that can pass through both regions. As routing occurs, the congestion level of each edge is calculated based on the capacity of the edge and the amount of wires passing through it. The global routing output are the gcell routes of each net. A gcell route of a given net, called *global routing guide*, is a region that restricts, with some flexibility, the detailed routing solution for this net as shown in Figure 3b, c and d.

The method to route a net depends on the its number of pins. The approaches can be divided into two categories: for two-pin nets and for nets with more than two pins (multi-pin nets). Considering two-pin nets, there are two ways to solve the problem: using maze routing or using pattern routing. In maze routing the A * algorithm (section 3.2.2) or some variation of it is used. In pattern routing, the search is restricted specifically to considering path patterns that follow the format L (one bend), Z (two

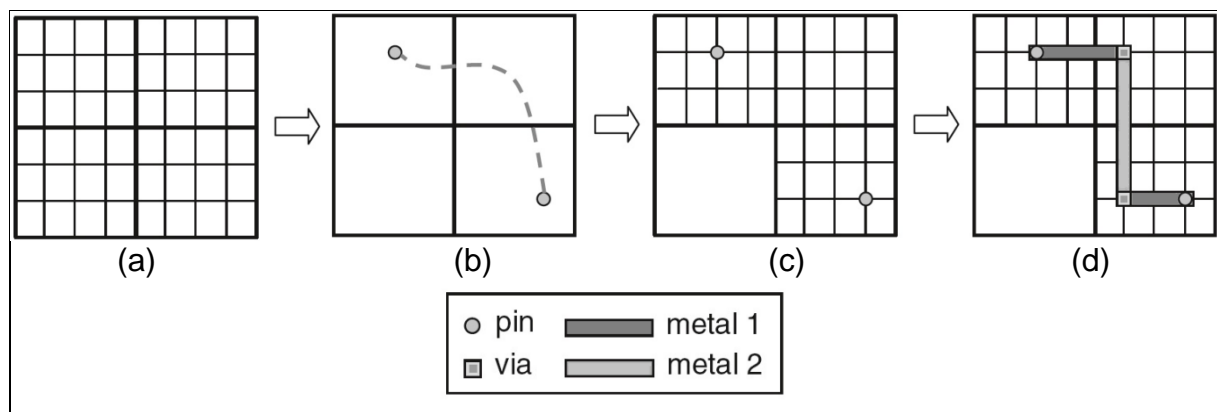


Figure 3– Illustration of the global routing. (a) Detailed grid (thin lines) and coarse grid (thick lines). The coarse grid contains four gcells. (b) Global routing determines the connection path, in gcells, between two pins. (c) The detailed routing area is restricted according to the resulting global routing path. (d) The connection is made in detailed routing. Source: (CHEN, 2009).

bends, as in Figure 3d) or U (a detour with two bends). This approach was proposed by Chen (1999) and Kastner (2002). Both search styles (maze and patterns) guarantee the optimal path, but pattern routing is restricted to a much smaller search space, presenting a superior performance in runtime. Thus, pattern routing is widely used in global routing of two-pin networks (CHEN, 2009).

The second category of algorithms used in global routing refers to multi-pin nets. Within this scope, there are two main approaches to solving the problem. The first is to decompose the net into multiple two-pin nets and solve each one by default two-pin routing. The second is to solve the Steiner tree problem (SHERWANI, 1998).

The first approach can be solved by finding a Rectilinear Minimal Spanning Tree (RMST) from the net pins. A RMST is a tree that connects all vertices of a graph, having the lowest possible cost, which is given by the sum of edge costs. In this case, the vertices of the grid are the pins and the edges represent the possible connections of each pin with the others. Edge cost is the lowest possible connection cost. RMSTs can be found in polynomial time by the algorithms of Kruskal and Prim (CORMEN, 2002).

Solving the problem by this approach can provide suboptimal results. Figure 4a shows the net resulting from a four-pin MST. Note that the net does not have optimal WL, which is the case in Figure 4b. In this case, the router split the net into three two-pin nets, which were routed separately. Considering separately the resulting path of each pair of pins, it is clear that the WL is optimal. However, the total WL of the net is not optimal. In order to obtain the optimal result, it is necessary that the paths share some sections, but this cannot be foreseen since the problem was decomposed. Thus,

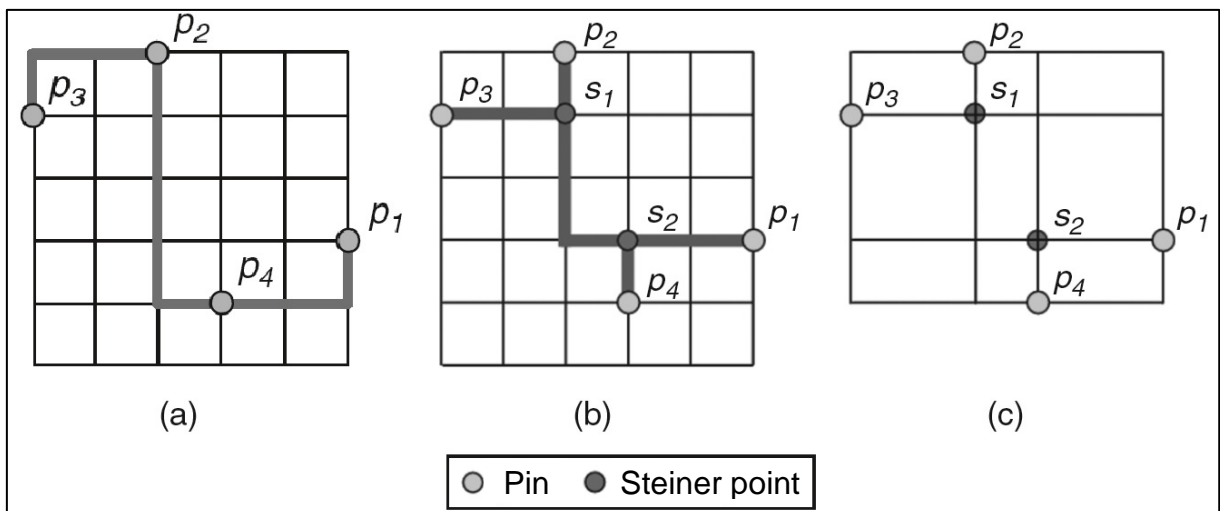


Figure 4— Illustration of a MRST (a), a RMST (b) and a Hannan grid (c), with a 4-pin net. Source: (CHEN, 2009).

it is necessary to have a global view of the problem, which is the case of the Steiner tree problem.

The second approach to routing a multi-pin net is to find the Minimal Rectilinear Steiner Tree (MRST) of the net. A MRST is an RMST with additional vertices called Steiner points. These points are strategically inserted into locations to force the net to reuse the same path sections to connect different pins. The problem in finding an MRST consists in determining the appropriate number of Steiner points and their locations to maximize this “path section sharing”. These points form the Hanan grid (HANAN, 1966), which is obtained by projecting vertical and horizontal lines over each pin (Figure 4c). The vertices of this grid are the intersection points of these lines. The problem of finding an MRST is NP-Complete. Thus, heuristics were created to solve the problem quickly, with little WL loss, compared to the optimum result. The FLUTE algorithm (CHU, 2008) is a good example of the use of such heuristics, showing optimum results for nets of 9 pins or less. However, these methods do not take into account important routing factors such as obstacles, congested areas, and delay. There are several methods that consider the presence of obstacles (LIN; WU; LI; HUANG; 2007, 2007, 2008, 2010). Regarding congestion consideration, the FastRoute router (PAN; PAN; ZHANG; XU; 2006, 2007, 2008, 2009) is one of the main references. Another method of calculating an MRST is by using maze routing. The advantage of this approach is that it easily handles all the factors involved in routing, providing a solution with better quality, but with a runtime disadvantage. This approach is used by the AMAZE tool (HENTSCHKE; 2007, 2009).

The mentioned approaches are used to route single nets. For routing all nets in the chip, there are two main approaches: concurrent, using ILP, and a sequential, using a negotiated congestion technique. The concurrent one routes all nets simultaneously, and the sequential routes them one after another. The sequential methodology relies on ripup-and-reroute (RNR)¹ techniques. The RNR consists in ripping nets already routed so that new nets can be successfully routed. Considering that the ILP-based global routers present higher runtime, the sequential approach is preferred (KAHNG, 2011c). The sequential methodology using negotiated congestion technique keeps a cost history of the nets that passed between each gcell. When the capacity of an edge of the gcell graph is overflowed, the cost of this edge is increased. Thus, congested

¹ RNR is sometimes called RRR, as in (KAHNG, 2011c)

regions present high penalty costs for the routing algorithms. This avoids the paths to pass through them, leaving the region to be used only by net nets that really need to do so (KAHNG, 2011d).

2.3.2 Detailed Routing

In detailed routing, all wire routes are determined within the global routing guides, with some flexibility. Detailed routing also tries to optimize metrics as WL and via count, but the major part of these optimizations was already performed in global routing and are reflected in the global routing guides. There are many approaches to solve detailed routing. They are described in section 3.

Normally, the routing is restricted, with some flexibility, to a predefined routing grid. Each metal layer has a set of tracks in the preferred routing direction. There is also a routing approach that does not use any predefined grid, which is called gridless routing. Anyway, even in gridless routing, non-uniform grids may be created on-the-fly to run path searches.

The physical components involved in routing are pins, wires, vias and obstructions. The pins are interconnected by wires and vias. The wires have a default width, but some special nets may require a different width. Upper metal layers have thicker wires. The vias connect wires of adjacent layers. A via has a bottom and a top pad, as shown in Figure 5. Thus, a via is defined by three shapes: in the bottom, in the










	Bottom	Cut	Top
Via 1			
Via 2			
Via 3			
Via 4		 	

Figure 5- Illustration of via shape configurations of a via library.

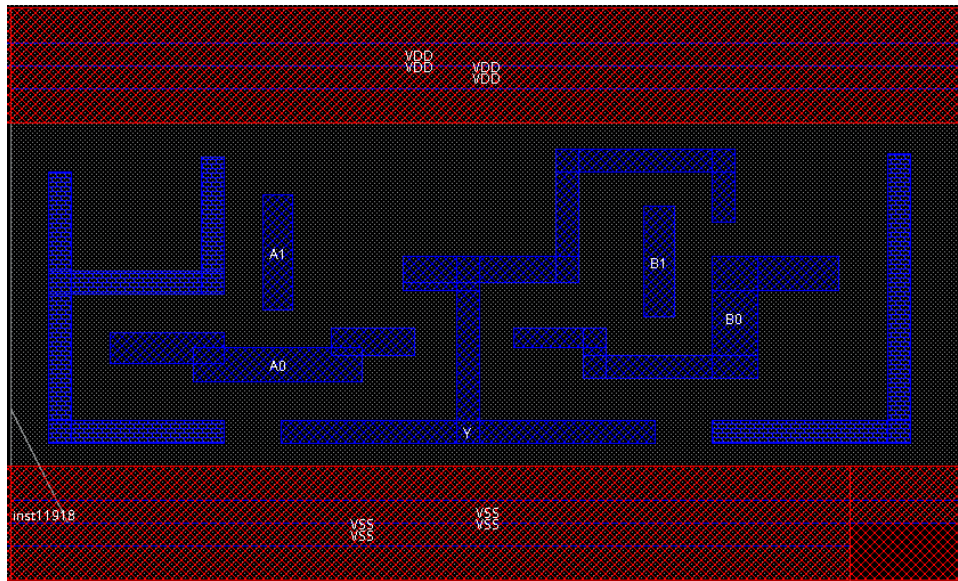


Figure 6- Picture of a standard cell in Cadence Innovus tool. Metal 1 is blue and metal 2 is red. The labeled polygons are the pins, and the other are the obstructions.

cut and in the top layer. Some vias present more than one cut shape. The vias that a detailed router can use are defined in a via library. Most of the pins are in the standard cells. These pins have more irregular shape and require much care when making contact with the wires, since it is easy to such connections cause DRVs, as will be discussed ahead. The other pins have a more regular shape, usually composed by just one rectangle, and are located in the chip boundaries (I/O pins) and in the macro cells. The obstructions are usually present in some standard cells and in the macro blocks. Figure 6 shows the pins and obstructions of a standard cell.

One of the biggest challenges of detailed routing, and possibly the biggest, is design rule handling. Dealing with all design rules during the construction of the routing solution is hard. Thus, this incentive the division of detailed routing in two parts. The first, called initial detailed routing, takes the major effort of the process. It finds all wire routes and handles the more feasible design rules. The last part, which is often referred as *design rule cleanup* tries to solve the remaining design rule violations (DRVs) and may perform some refinement on the solution. The problem is that it is not always possible to solve all DRVs in the cleanup step, and this has been constantly seen in the literature (AHRENS; GESTER; JIA; ZHANG, 2015, 2013, 2017, 2013). The more violations the initial phase leaves for the final one, the more likely is to be impossible (or unfeasible) to solve them all. Thus, the initial phase should handle DRVs as much as possible, while observing all metrics to be optimized. It is a challenging

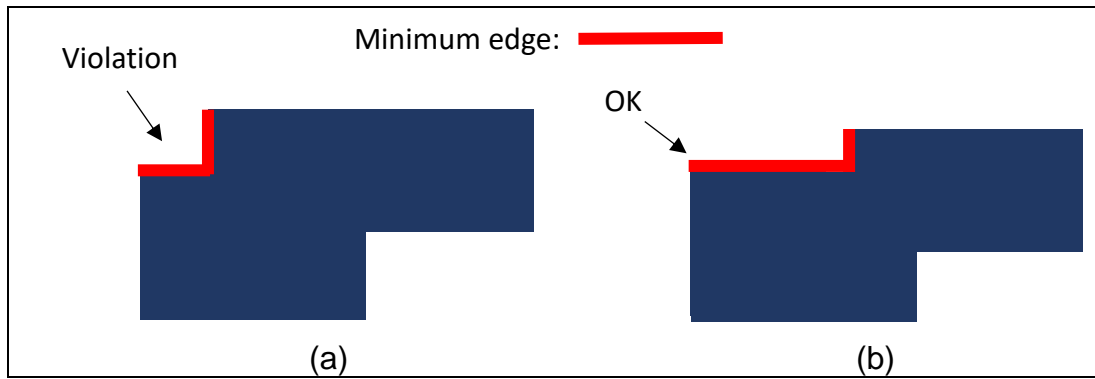


Figure 7- Illustration of minimum edge rule.

implementation work, but it is possible, and the experiments in this work are evidence of this.

2.3.2.1 Design Rules

This section presents some common design rules handled in detailed routing. Each layer, metal or cut, has a set of design rules. These rules can be grouped in two main categories: *shape rules* and *spacing rules*. Shape rules impose constraints in the shape of the objects. Spacing rules require the objects to obey a minimum distance from each other that may be influenced by many factors. There are also the *overlapping rules*, which impose restrictions on the overlapping between shapes, but they will not be treated here.

The *minimum area* is an example of a shape rule. It requires that any polygon in a metal layer meet a minimum area. The *minimum edge* rule is another example. In this case, two consecutive shape edges that do not meet a minimum length are forbidden. Figure 7 illustrates this rule. Also, any shape must respect a *minimum width*. The default wires of a metal layer have this width.

The *parallel runlength* is a spacing rule that requires that two metal shapes respect a minimum spacing from each other. The thicker the shapes, the higher is the spacing. The same way, the higher is the extent that both shapes run in parallel (i.e. the parallel runlength), the higher is the spacing. A spacing table defines these values. The rows and columns are the widths and parallel runlength ranges, and the table values are the required spacing. The minimum spacing value of the table is the default minimum spacing between any object. Usually, this spacing value plus the minimum width rule value, which is the default wire width, result in the *pitch* of the metal layer. The pitch is the distance between the grid points of the routing grid. Sometimes the

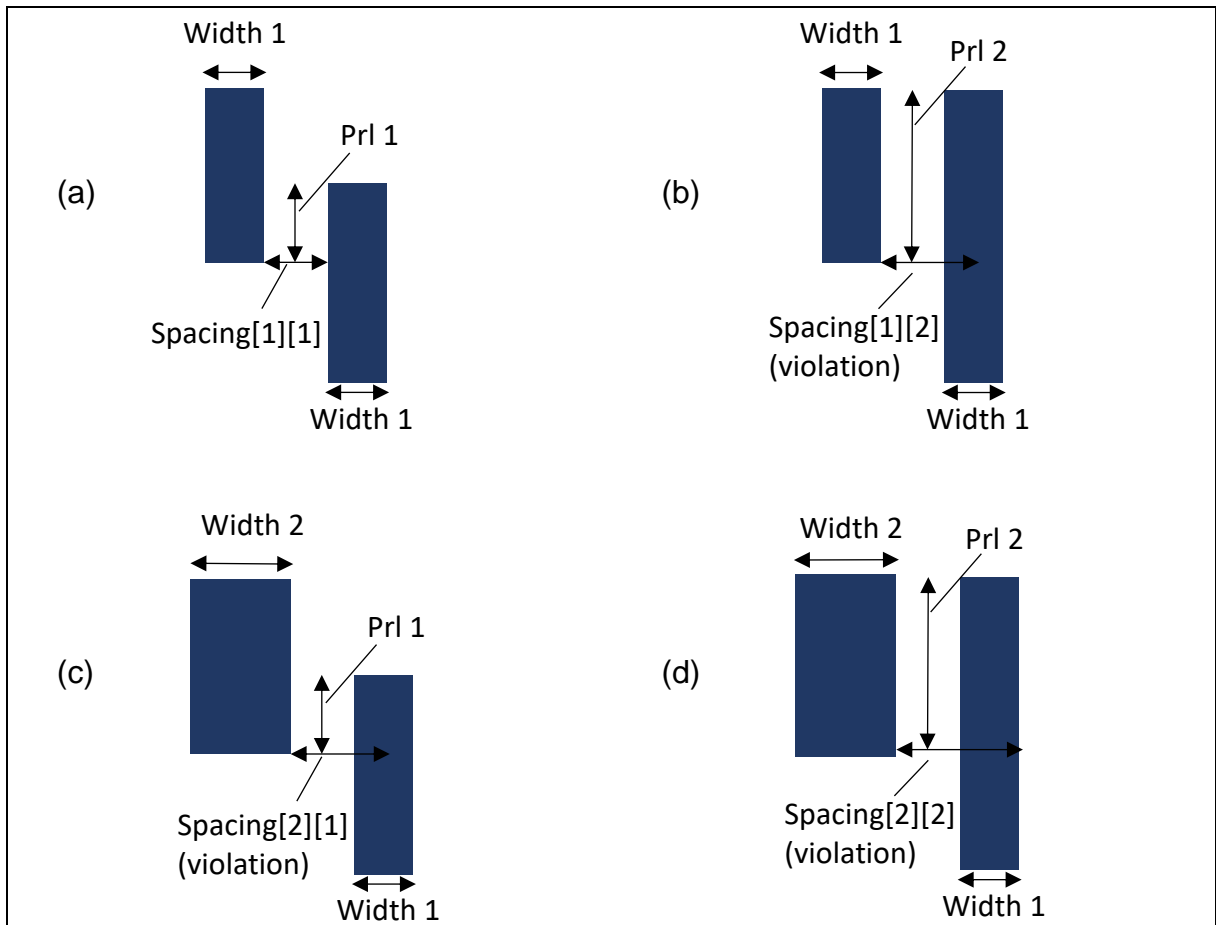


Figure 8- Illustration of the parallel runlength rule.

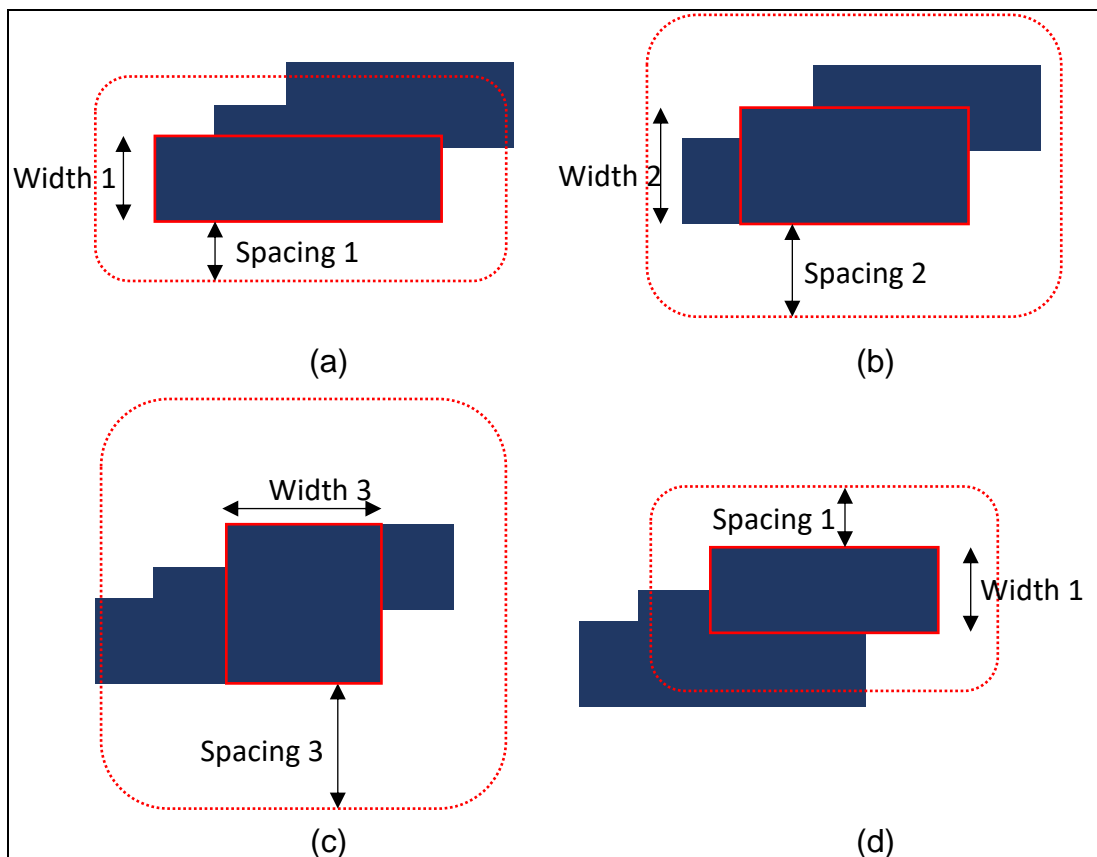


Figure 9- Illustration of the parallel runlength rule with a polygon with multiple rectangles.

pitch can be higher than this sum.

Figure 8 shows examples of the parallel runlength rule. “Width 1” and “Width 2” are widths of the width ranges 1 and 2 of the spacing table. The same way, the “Pr1” represents the parallel runlength between both objects. Spacing[w][p] is the required spacing between two objects; w is the index of the width range that contains the greater width of the two objects and p is the index of the parallel runlength range that contains the parallel runlength of the two objects. The width of an object is the lesser side of the rectangle.

A metal object may be a polygon, defined by many rectangles. In this case, it may present parts that trigger different spacing values on the spacing table, due to their different widths. This is shown in Figure 9. The spacings shown are width-

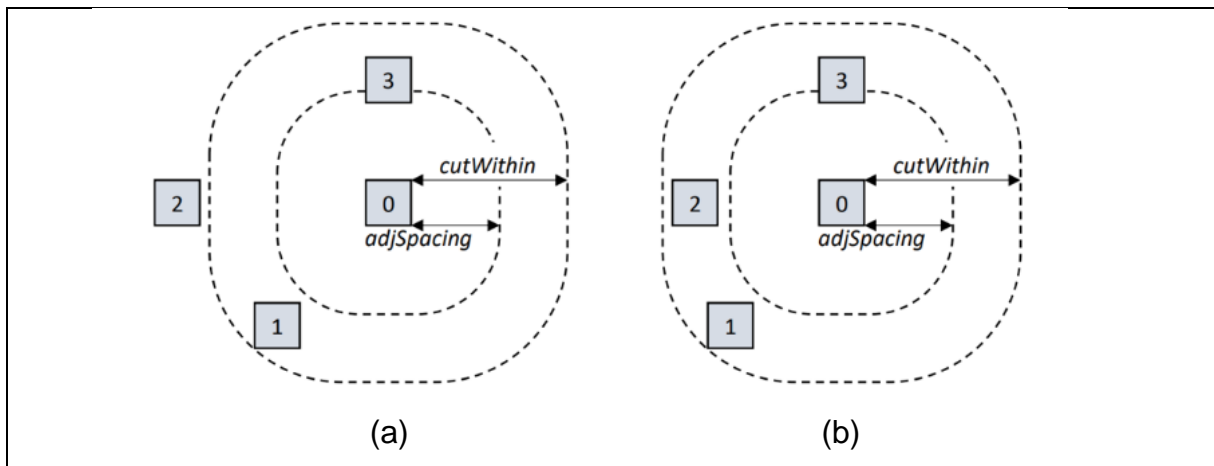


Figure 10- Illustration of the cut spacing rule with adjacent via cuts constraint. The via cut 0 requires spacing of *adjSpacing* if there are 3 or more via cuts touching *cutWithin* radius.

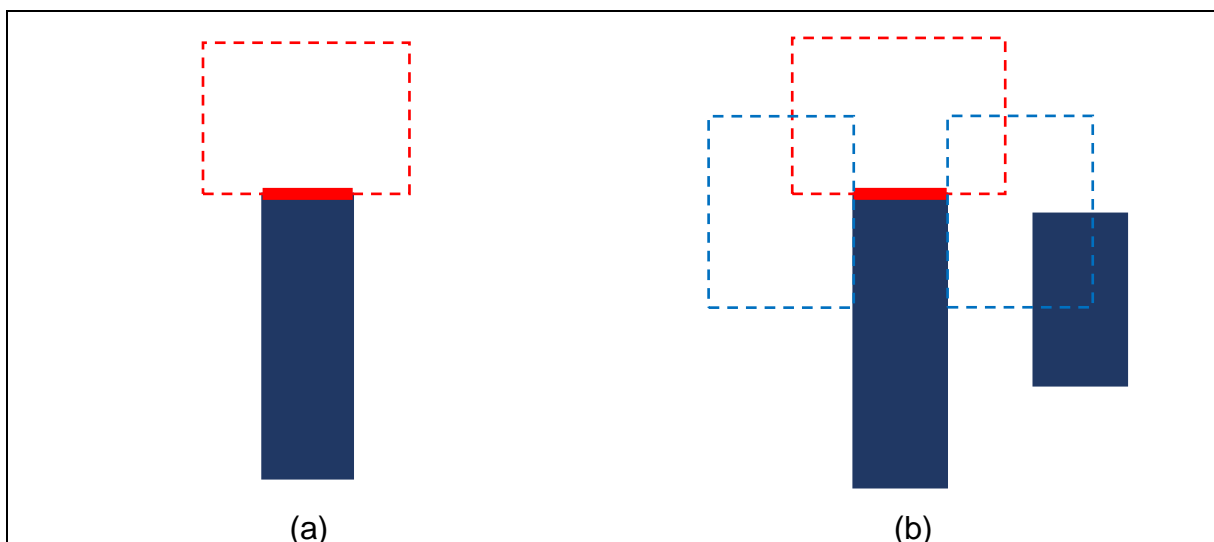


Figure 11- Illustration of end-of-line rules. (a) is the simple case and (b) with parallel edge modifier.

dependent only, as the parallel runlength itself, between two objects, is ignored. An object passing by this polygon must be aware of all of its parts in order to avoid violations. The parallel runlength is measured in each part separately.

The *cut spacing* specifies the minimum spacing between two via cuts. This value is usually higher than the minimum wire spacing. The cut spacing rule may present some modifiers, like the *adjacent cuts*. A cut spacing with, say, 3 adjacent cuts means that this spacing is only applied when there is 3 via cuts in a given radius. Figure 10 illustrates the cut spacing rule with adjacent cuts constraint.

The end-of-line rule requires special spacing from a shape edge that has less than a determined size. Figure 11 shows examples of this rule. The red segment is the end-of-line edge. Any object in the red dashed rectangle causes a violation. The rule may present some modifiers, like the *parallel edges*. In Figure 11b, the parallel edges modifier is used, meaning that it is necessary to exist an object in any of the two blue rectangle areas in order to trigger the spacing requirement. If the modifier is set to *two edges*, then both blue rectangles must be intersected to trigger the spacing.

The pitch of the routing grid ensures that the wires of default width do not present violations with each other. Thus, the violation possibilities are in whatever objects with a width and/or with a spacing requirement greater than the minimum. These objects are pins, obstructions, non-default wires and vias. Thus, the major attention in avoiding DRVs is in the pin access and in the path search, since the paths need vias to connect the wires of different layers.

The ISPD18 benchmarks, adopted in the experiments of this work, use a simplified set of the mentioned design rules. The rules are: (1) minimum area; (2) cut spacing (no adjacent cuts); (3) end-of-line (no parallel edges); (4) parallel runlength, ignoring the parallel runlength itself, varying only the widths. Also, there are no non-default wires.

3 Detailed Routing Literature

This section presents a literature review of detailed routing. Section 3.1 presents detailed routing approaches and techniques. Considering that path search is a fundamental building block of detailed routing, and that part of the contributions of this work is related to path search, section 3.2 presents path search related algorithms. Pin access is also another important step of detailed routing, and since it is one of the main contributions of this work, section 3.3 presents pin access techniques.

3.1 Detailed Routing Approaches and Techniques

Detailed routing is a problem that can be solved in many ways. Over the history of routing in integrated circuits, many approaches and techniques were used. In the old days, when the number of layers available for routing was two or three, routing was solved using the channel routing approach. With the change of paradigm to the current one (over-the-cell routing), the resolution of the detailed routing problem is more flexible, presenting many different approaches.

The detailed routing methodologies may be grouped in some categories, according to the routing *order* of the nets, the use of a predefined *grid*, and the awareness of manufacturing peculiarities that improve *yield*.

Regarding the routing order, there are the sequential (also called net-by-net) and the concurrent approach. In the sequential one, the nets are routed one after another, such that previously routed nets form obstacle to the new ones. Nets that could not have a route found are rerouted in RNR (SHIN, 1987) step, as in global routing, in which nets are routed possibly ripping out other nets. The sequential routing tends to present a favorable runtime, but this can be compensated by the RNR step. The routing order of the nets has a high impact in the number open nets (i.e., nets that could not complete a connection). Thus, in order to properly work, this methodology relies on a good net ordering.

The concurrent approach routes all net pieces in some routing regions simultaneously. It is better at handling high congested areas, but it tends to present higher runtime than the sequential methodology. It is still possible to present open nets and rely on RNR, but much less than in the sequential approach. The concurrent routing is easier to parallelize than the sequential one.

Regarding the routing space modeling, most approaches use a predefined grid. This facilitates the implementation and tends to provide faster runtime than gridless

approaches. The main argument of gridless routing is that it naturally handles design rules better, mainly the less common ones.

Some routing approaches focus on optimizing good practices that may impact on the manufacturing yield. Examples are routers that try to use multi-cut vias and the ones that attend to the multiple patterning constraints (DING; YU; YUAN; GAO; MIRSAEEDI; MA; LIU, 2018, 2018, 2009, 2010, 2011, 2012, 2016).

3.1.1 Channel and Switchbox Routing

In the old routing paradigm, between each row of logic gates there was a space, called *channel*. The pins of the cells were located at the cell border adjacent to the channel. The connections between the cells were performed on the channels. Two metal layers were used, at first.

A channel is a rectangle with pins at two of its opposing borders. As the complexity of the circuits increased, not all connections could be performed on the same channel. The routing space was divided in rectangles, called *switchboxes*, with connections at the four borders. Figure 12 shows an example of a switchbox routed.

The first channel routing algorithm was the Left-Edge (HASHIMOTO, 1971). Later, (DEUTSCH, 1976) proposed the Dog-Leg algorithm, which was an improvement over Left-Edge. The switchbox routers (JOBBANI; LUK; MAREK-SADOWSKA; HAMACHI, 1986, 1985, 1985, 1984) derive from the channel routers. The basic procedure of channel routing is the following. First, a horizontal constraint graph is built

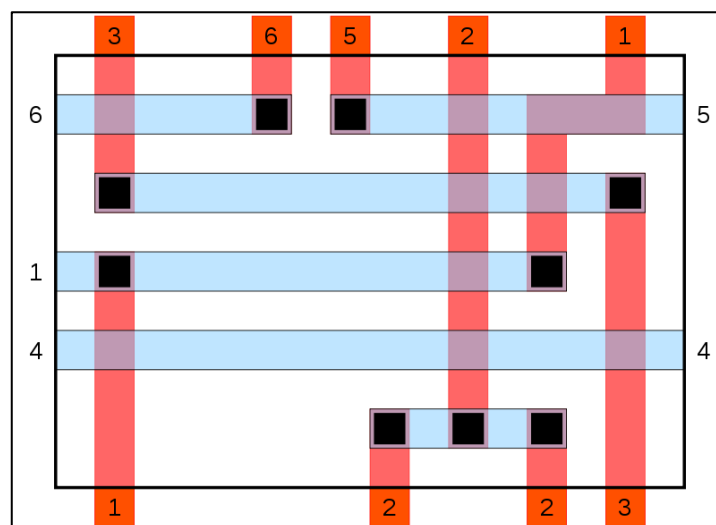


Figure 12 – Illustration of a switchbox routed. Blue and red represent the different metal layers, and the black squares are the vias. The terminals are labeled by numbers representing their respective net.

in order to detect the minimum number of routing tracks needed to make the routing feasible. Second, a vertical constraint graph is built, aiming to provide a topological order of pin connections to be routed. Finally, the connections are implemented following the vertical constraint graph topology and respecting the constraints of the horizontal constraint graph.

With the increase in circuit complexity, the routing channels had to become even larger in height, which would increase the circuit area. Thus, a third metal layer was introduced. In this layer, wires were allowed to be put over the cells. Several over-the-cell channel routers were proposed (CONG, Jason; HOLMES; CONG, J.; NATARAJAN; WU; 1990, 1991, 1990, 1992, 1992). However, this was not enough to handle the wire congestion that was constantly increasing. Thus, more metal layers were introduced, and the routing space assigned to the channels became pointless, causing the end of channel routing.

3.1.2 Over-the-Cell Routing Using Gcells as Switchboxes

In the time of channel routing, global routing worked on irregularly partitioned regions, constituted of channels and switchboxes. With the change of paradigm, global routing partitioned the routing area regularly in gcells, as already shown in Figure 4. A path obtained in global routing is a sequence of gcells, called global routing guide, connecting all net pins. In detailed routing, the routing of a net is restricted to the region denoted by this set of gcells. Considering the sequential paradigm, in order to route a net, it is necessary to use a path search to connect all of its components (pins and wires). Since these components may be far away from each other, this may cause high runtime overheads in some path search algorithms. Thus, one detailed routing approach is to decompose these long connections into smaller connections, one for each gcell (IGUSA; PARNG, 1989, 1989). Each gcell, work as a switchbox, with pseudopins at its boundaries, as shown in Figure 13. Pseudopins are terminals that act like pins. This approach gave rise to the Cross-Point Assignment (CPA) problem (KAO; CHANG, 2001, 1995).

The CPA, also called Pseudopin Assignment, is an intermediate step between global and detailed routing, aiming to determine the positions of the pseudopins at the gcells boundaries. An advantage of this approach is that it allows a good level of parallelism, since it is possible to route each gcell separately. This methodology can

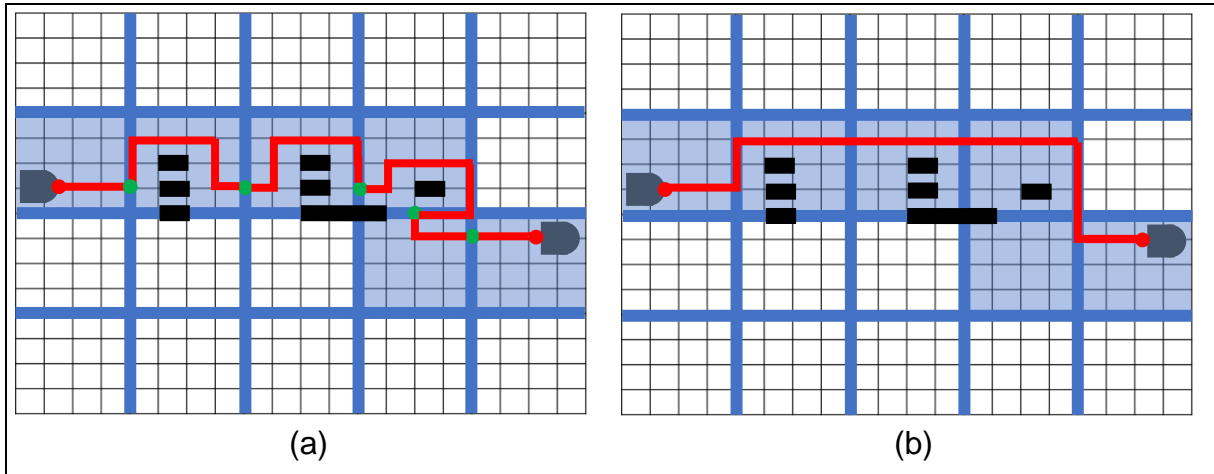


Figure 13- Illustration of routing each gcell as a switchbox and the problem involved in the crosspoint assignment. The gcell grid is denoted by the thick blue lines. The global routing guide is in translucent blue. A path (in red) connects two cells. The green points are the crosspoints (or pseudopins). In (a), it is possible to see the WL increase that may happen in this approach, due to the decomposition of the searches. In (b) it is shown the optimal solution.

be either sequential or concurrent. That is, it is possible to route all connections of all pseudopins of a gcell, each time one is processed (concurrent approach), or it is possible to consider one net at each time (sequential approach) and perform the connections of all gcells in the global routing guide. The mentioned parallelism is independent of the approach.

A disadvantage of this approach is that it provides bad results in terms of WL, as shown in Figure 13a. Since a connection is decomposed, the global view of the full connection is lost and thus, the optimal path cannot be guaranteed. This methodology became less popular with the emergence of more modern approaches (as in the next sections), which performed long connections more efficiently.

3.1.3 Sequential Routing Using Path Search for Long Connections

The justification of the path decomposition of the previously commented approach is that, the path search algorithms available in that time were not satisfactory to handle long connections. The maze search algorithms are known to present a high processing time, which is proportional to the available search space. The line probe algorithms are faster, but don't guarantee the optimal path and cannot handle variable grid point (and graph edge) costs. As will be shown in section 3.2.4, Hetzel's algorithm (HETZEL, 1998) uses line segments and is goal oriented, using A* technique. In the current work, this kind of path search is called A*-interval-based path search. With this algorithm, there is no reason to decompose the pin connections. Another possibility is to preprocess the global routing guide area to create an optimized graph for a Dijkstra-

based (CORMEN, 2001) path search algorithm, as in (CHEN, 2019a, 2019b). However, as the experiments in this work suggest (section 7), the time spent in constructing such graph does not compensate the runtime improvement of using an interval-based path search.

The sequential detailed routing methodology is the following. First, the nets are ordered in a priority queue. As mentioned earlier, a good net order is very important for a good routing. A good criterion for net ordering is related to the routing area of the global routing guides or the estimated WL of the net. Routing nets with less area (or WL) first theoretically tends to present good results, since a lower area net has less freedom to perform the connections. However, in practice, routing nets with more WL first may be better. Since the routing guides of all nets have the same width, the net length factor weights more than the routing area factor. Also, longer nets have most of their routes in higher metal layers, using lower layers only for pin access, while shorter nets tend to use more the lower layers. This makes longer nets present few blockages to shorter nets, if they are routed first, while routing first shorter nets presents more blockages to longer nets.

After the net ordering, the nets are sequentially routed. Since a net can have multiple pins, it is decomposed in two pin nets. This decomposition may be performed at the routing procedure begin, before net ordering, or after a net is selected for routing. Each two-pin net is restricted to a section of the global routing guides, which is called *tunnel* in this work. The two-pin net is routed and the resulting path is inserted into the routing space. Open nets are handled in the RNR step, as mentioned earlier. The path searches in RNR may rip other nets, and the process continues until all nets are routed or some other criterion, like a runtime limit, is met. In order to avoid RNR loops, the path search is penalized when routing through other wires. Another good practice is using a Negotiation-based technique (KAHNG, 2011d) in which the routing space stores a cost history of the wires that were already inserted on it. Thus, congested regions present higher penalty cost for the path search.

In the sequential approach, parallelization is less trivial than in the concurrent one, but it is still feasible. (CHEN, 2019a, 2019b) proposes to route, in parallel, batches of nets that do not present routing guide overlapping with each other. After a batch is routed, all found paths are inserted into the routing space, and the next batch is chosen, and so on.

Examples of sequential routers are (SHIN, 1987), (HETZEL, 1998), BonnRoute (GESTER; AHRENS, 2013, 2015), and (CHEN, 2019a, 2019b).

3.1.4 Track Assignment

Track Assignment (TA) was proposed in (BATTERYWALA, 2002) as an intermediate step between global and detailed routing. It is closer to detailed routing than global routing, though. Consider Figure 14. Each row of gcells of a global routing guide is called *global segment*. An *iroute* (or *ir*) is a wire segment from detailed routing used to implement the global segment. A *panel* is the region denoted by an entire line or column of gcells of the global routing graph, following the preferred routing direction. The track assignment problem consists in assigning all *ir*'s of a panel to routing tracks without overlapping *ir*'s of different nets.

The first step of TA is to group all *ir*'s to their respective panels. Then, each panel is processed separately. This allows a good level of parallelization. For a given

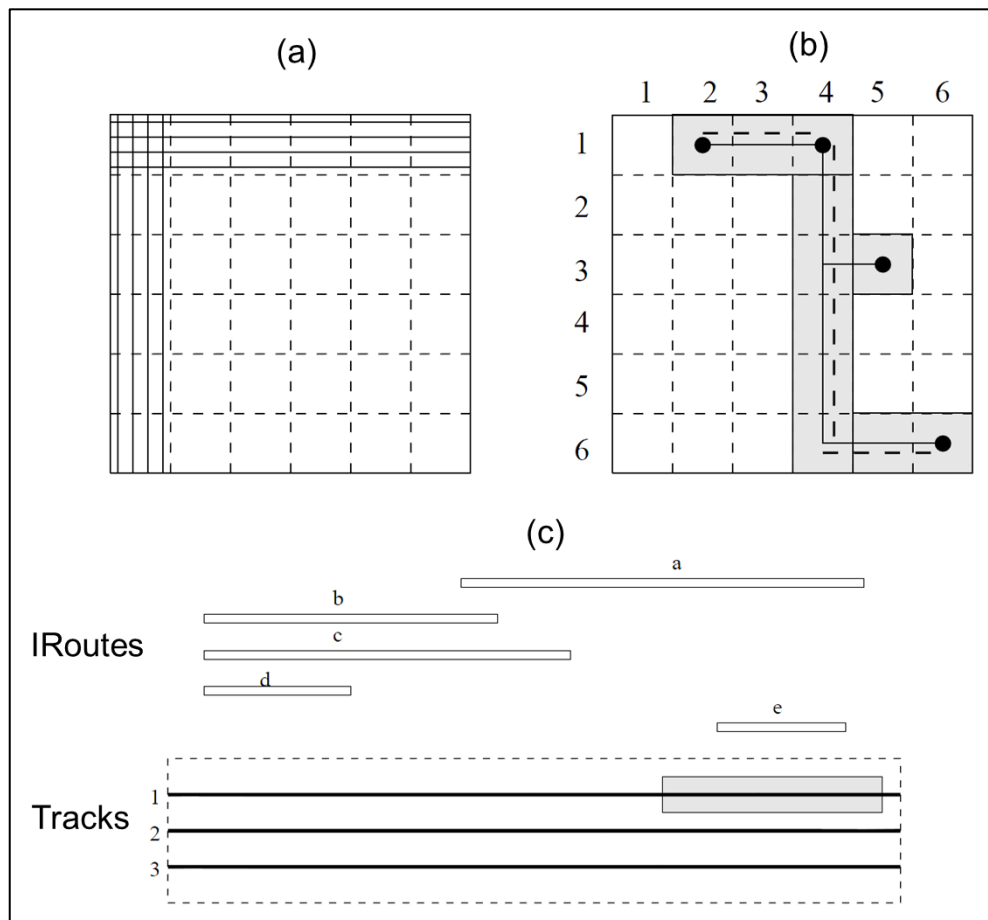


Figure 14 – Illustration of the Track Assignment problem. (a) Vertical and horizontal panels in global routing grid. (b) The global routing guides of a net (in grey), its global segments (dashed lines), its iroutes (lines), and its pins (dots). (c) Track assignment example. The grey rectangle is an obstacle. Source: (BATTERYWALA, 2002).

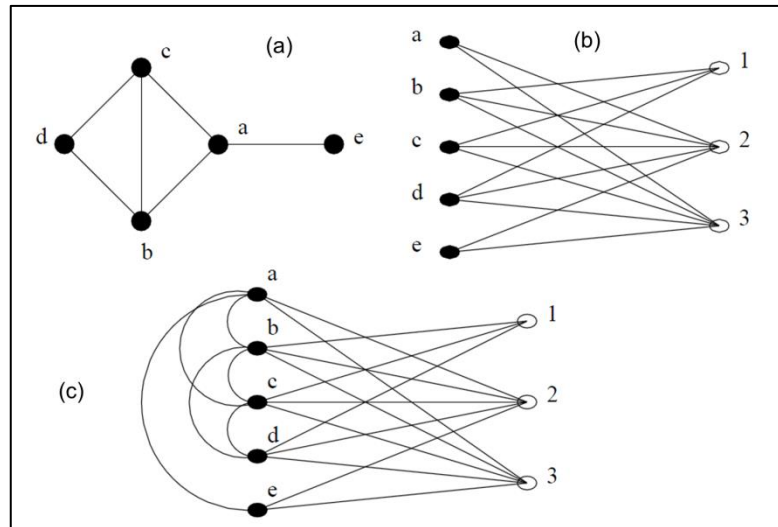


Figure 15 – Illustration of the graphs used in the TA technique. (a) irou-te overlap graph. (b) bipartite assignment graph. (c) combined graph model.

panel, an iroute overlap graph is constructed (Figure 15a). Each vertex of this graph is an *ir* and the edges are their overlaps. Then, this graph is extended to a bipartite assignment graph (Figure 15b), where one set of vertices is composed by the *ir*'s and the other set contains the tracks. The edges connecting both sets represent the availability of assigning an *ir* to a track. The weight of these edges represents the cost of assigning the *ir* to the track. These costs are modeled by many factors, such as obstruction costs, long wire penalties in some cases, and anchor costs (wires and pins of the same net should be as close as possible).

Since the optimal track assignment solution is NP-Complete (BATTERYWALA, 2002), a heuristic is used to solve the problem. In an iterative procedure, the largest cliques¹ of the iroute overlap graph are selected, and their *ir*'s are assigned to different tracks, by modeling this task as a weighted bipartite matching problem and solving it using the shortest augmenting path algorithm of Jonker and Volgenant (1987). The combined graph (Figure 15c) is updated by removing the assigned *ir*'s and their edges linking to the tracks.

After the track assignment step, detailed routing is performed to connect the pieces of the nets. Since the longer connections were already performed by TA, the path search algorithm (even a maze search one) does not encounter great difficulties to find the routes.

In 2013, Zhang (2013) proposed a detailed router, called RegularRoute, based on TA. However, it does not use TA as an intermediate step between global and

¹ A clique is a set of vertices in an undirected graph such that any pair of vertices is connected by an edge.

detailed routing. The detailed router itself uses TA-based technique and provide the full connections of the nets. In RegularRoute, the *ir*'s are not just segments, but also the segment connections to their terminals. These new *ir*'s are called *choices*. There can be many *choices* for the same "*ir* segment". A terminal can be a pin or other segment. The order of assignment of *choices* is determined by modeling the problem as a Maximum Weight Independent Set (MWIS) problem. The MWIS problem aims to find, in a graph, the set of vertices of higher weight, where there is no pair of vertices connected by edges. In the case, the vertices are the *choices* and the edges are their conflicts. A vertex weight is given by a function that returns the *benefit* of the *choice*. The MWIS is solved by iteratively selecting the *choice* of highest *benefit*, inserting it on the routing space and removing all its neighbors from the graph. Then, all the pieces, of the remaining *choices*, that can be assigned to the current panel are assigned. All of these pieces of choices have their terminals assigned to the next upper metal layer, aiming to delegate the incomplete connection to the other layer.

3.1.5 Multicommodity Flow and Integer Linear Programming

The multicommodity flow (MCF) is a graph flow problem. A *commodity* is a flow demand, which has a source and a sink node. The graph $G = (V, E)$ must be a flow network, that is, each edge has an associated flow capacity. Given G and a set of commodities, the goal is to find a path for each commodity such that the total flow in each edge does not exceed its capacity.

Jia (2017) proposed a MCF-based detailed router in which track assignment (BATTERYWALA, 2002) is used, greatly mitigating the hardness for a MCF approach. Then, the routing problem is modeled as a MCF problem, and it is solved by ILP. ILP consists in optimizing an objective function, given a set of constraints. The objective and constraint functions are all linear. The edges in the MCF model are the edges of the grid graph. The edge capacities are binary (0 or 1), meaning that a graph edge can either or not support a wire segment. The flow demand of the commodities is also binary. The objective function to be minimized, in the ILP, is the sum of all edge flows multiplied by the edge costs. The constraints are: 1) All pins of each net are connected by a path; 2) There is no wire segment overlapping; 3) There are no design rule violations. The router presented a good design rule handling, but at the high cost of runtime.

TritonRoute also uses an ILP-based detailed routing framework, but without modeling the problem as a MCF problem. The approach routes, in parallel, sections of each metal layer using ILP. The metal layers are routed from bottom to top sequentially.

3.1.6 Multilevel Routing

At the beginning of the last decade, several multilevel routers were proposed. The motivation for the creation of these routers were that the predicted shrinking size of the transistors would make global and detailed routing much harder, since there would be much more connections per area. The multilevel routing can be either a routing framework with additional steps between global and detailed routing, or an interleaved global and detailed routing framework with many levels of grid coarsening (increasing gcell size) and uncoarsening.

DUNE (CONG, 2001) is a gridless detailed router that has an intermediate step between global and detailed routing. This step is called wire planning, and it is used to plan wire connections in a more detailed view. When a routing connection fails, detailed routing reports back to the wire planning model, which replans the connection. The most popular multilevel framework is the V-shaped model (Figure 16). The V-shaped framework uses a bottom-up followed by a top-down approach. Routers using this model (LIN; HO; CONG, 2002, 2003, 2005) start the routing performing a grid coarsening (bottom-up step). In this step, local connections (i.e. connections within a gcell) are performed, and the routing congestion is estimated for the next coarse level. After some coarsening steps, the top-down phase is executed by uncoarsening the gcells while refining the routing pendencies.

When multilevel routers emerged, Hetzel's algorithm (HETZEL, 1998) was not popular. This path search algorithm certainly weakens the multilevel approach premise. The same is true for track assignment. Still, (HO, 2003) is a multilevel router that uses TA. The use of techniques to handle long connections weakens the multilevel premise, but the expectations about the increasing routing complexity, in that epoch, were high. Whether because global routing evolved to more efficient methods, or detailed routing approaches efficient for long connections emerged or became more popular, the multilevel approach lost emphasis in the academy. It should not be forgotten though, since it presents a more global view of the routing problem.

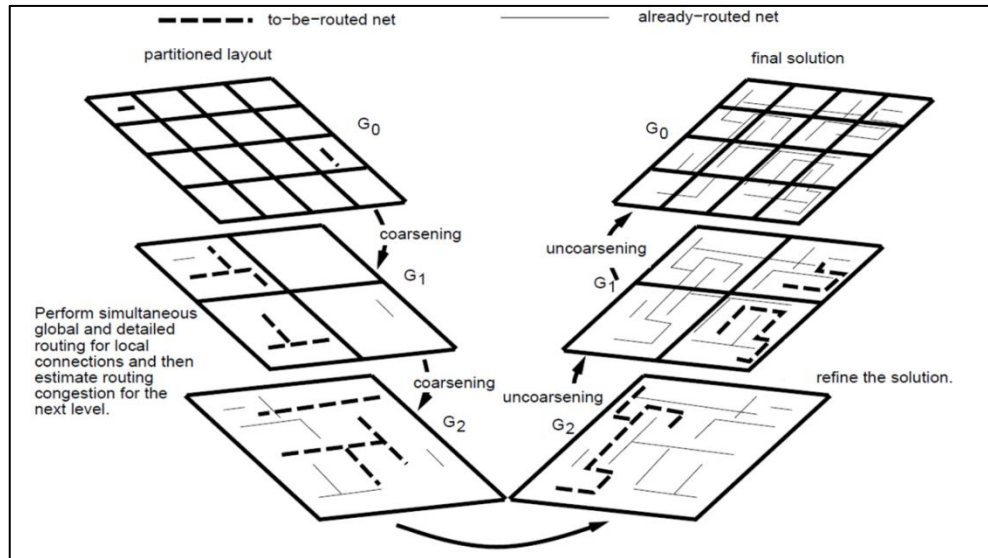


Figure 16 – The V-shaped multilevel routing framework.

3.1.7 Search Space Representation

3.1.7.1 Grid-Based

In the grid-based routing approach, the wires should respect a grid implicit in the search space. The *pitch* is the distance between grid points. This distance should be at least the minimum wire width plus the minimum wire distance. Since upper layers tend to have thicker wires, the pitch of upper layers is larger. Consequently, the layer with a larger pitch is misaligned in relation to the lower pitch layers. Thus, it is not possible to perform a connection between layers with different pitch anywhere, only when the grid points are aligned.

One naive approach to implement the grid is with a matrix for each layer. This implementation, in addition to utilize much memory, is very inefficient to handle grid queries. Since the wires have long segments, the addition of a wire segment in such grid, or a query to verify whether the segment is not blocked, would result in multiple queries for each grid point of the segment. Thereby, a good way to implement the grid is to keep a set of sorted line segments for each routing track.

A more modern approach to implement the grid is used in BonnRoute (GESTER, 2013). In order to handle different wire widths *on the same layer*, as well as the common misalignment of pins to routing tracks, Gester et al. (GESTER, 2013) proposed a grid data structure that allows routing tracks with irregular distances to each other. This data structure, called *shape grid*, shown in Figure 17, is responsible for storing all the shapes (wires, pins, power supplies) in the routing space. This grid partitions the routing space in cells. Each cell is small enough such that shapes of

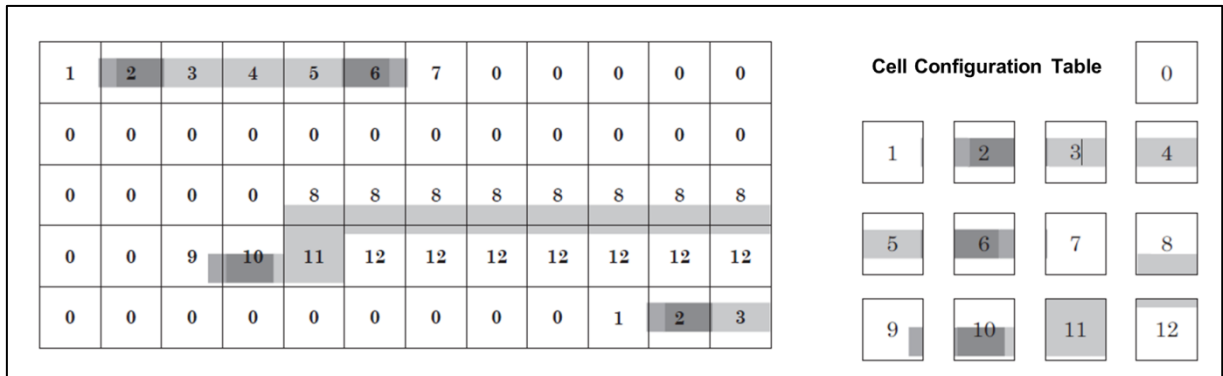


Figure 17 – The shape grid, proposed in BonnRoute. Source: (GESTER, 2013)

different nets cannot be legally present in the same cell. Each cell has a possible configuration. A configuration is a layout of circuit components, as shown in the cell configuration table of Figure 17. These cells are not stored separately. Instead, intervals of cells with repeating configuration are stored. These intervals store the cell id. Since the shape grid includes more detailed information of the search space, queries in this grid are expensive. Thus, a *fast grid* data structure is used in combination with the shape grid. The fast grid stores simplified blockage information obtained from the queries on the shape grid, and is implemented as the standard grid mentioned earlier, with AVL trees storing intervals in the routing tracks. The fast grid is used to perform queries for segments of standard wires (i.e. wires with default width). In (GESTER, 2013) about 98% of the grid queries are performed using the fast grid.

3.1.7.2 Gridless

Considering that wires may have different widths, and thus, different minimum distance rules, the *uniform* grid-based approach does not handle straightforwardly these characteristics. If the highest wire width is used to calculate the grid pitch, this causes much waste of space. Thus, an alternative approach to model the search space is to allow wires to be at any position.

The space representation is more complex than the traditional uniform grid. A common approach is shown in Figure 18. In a given layer, the layout area is sliced by a *slit tree* (KATO; KUH, 1987, 1990), and inside each slice, an *interval tree* (EDELSBRUNNER, 1983) is used to keep all shapes inside the slice. The slit tree slices are performed in the preferred routing direction. There are also orthogonal cut lines that are created for the interval tree. These cut lines are the nodes of the interval tree, while the cells formed by the horizontal and vertical lines are the leaves. The

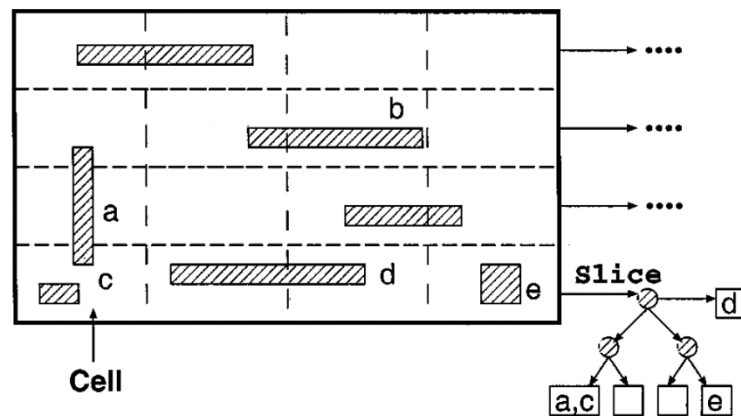


Figure 18 – Illustration of the search space representation using slit tree and interval trees. Source: (CONG, 2001).

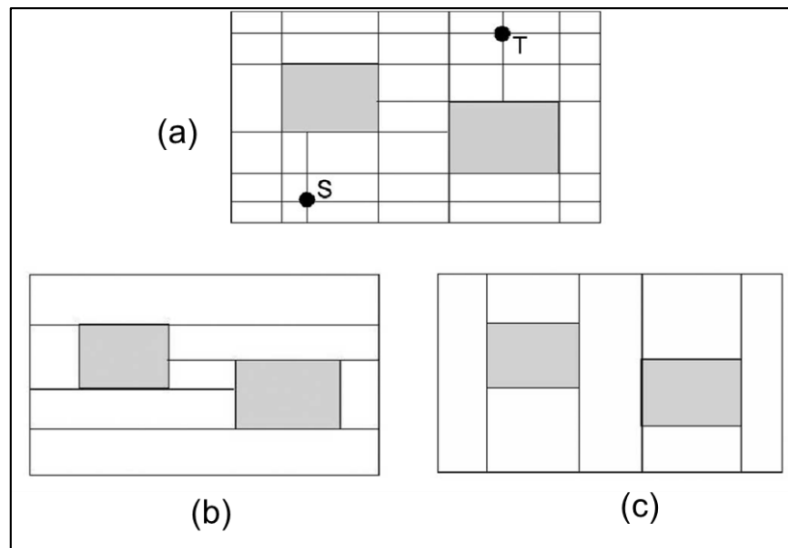


Figure 19 – Illustration of connection graph (a) and tile-based (b) (c) approaches. Source: (LI, 2007).

longer shapes that intersect a cut line are stored at the highest-level node (i.e. cut line) they cut. There are two approaches for gridless routing: *tile-based* (SATO, 1987) (MARGARINO; LIU, 1987, 1998) and by *connection graph* (OHTSUKI, 1985). A connection graph is obtained by extending the boundary lines of all obstacles until reaching the boundary of other obstacles or the routing boundary (Figure 19a). A path is obtained by running a maze path search algorithm in the graph. The graph can be created on-the-fly during the path search. Tile-based approach partitions the search space into two tile types (space tiles and block tiles) and organizes all tiles using a corner-stitching data structure (OUSTERHOUT, 1984). The tile plane of a routing layer is produced by extending the border lines, of the preferred direction, of all obstacles until any obstacle or routing boundary is reached (Figure 19b and c). A maze search algorithm is run in order to obtain the sequence of tiles that connect the source and

target pins (this is called tile propagation), and line probe methods (MIKAMI; HIGHTOWER, 1968, 1969) are used within each tile to define the actual path. Li et al. (LI, 2007) proposed a gridless router combining tile propagation and connection graphs. In (LI, 2011), the approach of (LI, 2007) was combined with track assignment (BATTERYWALA, 2002), which was aimed for grid-based routing.

3.1.8 Multiple Patterning Compliant Detailed Routing

The IC fabrication is performed using photolithography technology. A mask with geometric patterns is exposed to light over a photosensitive material (photoresist), creating the IC features (i.e. wires, pins).

With the advance of technology, and consequently the feature size shrinking, the IC features, represented in the mask, become too close to each other. This causes a precision loss in the printing of the features. The solution is to reduce the wavelength of the light used. However, this wavelength reduction has already reached a limit of 193nm. While the 13.5nm UV technology is not ready, other solutions have emerged, and double patterning technology is one of the most prominent.

In double patterning, the features are defined by two masks. Each mask has a pitch large enough to guarantee the printing precision. The lithography process occurs separately for each mask, assuring the fidelity of the printed features in both masks. Using the same concept, triple and quadruple patterning also emerged. The term “multiple patterning” refers to any of them.

The use of multiple patterning requires an additional step in the design flow of ICs, which is layout decomposition. This procedure assigns a mask to each feature in the layout. Figure 20 shows a layout decomposition example. However, this decomposition is not always possible. A solution is to try to modify the layout, which may be very problematic, or to decompose a feature in two polygons and assign each one to a different mask, creating a *stitch*, as shown in Figure 20. The problem is that a stitch is highly sensitive to distortions in the printed features, possibly causing contact issues, as shown in Figure 20b. Thus, the number of stitches should be minimized.

The layout decomposition is performed after routing. However, considering that the majority of the features are wires, ignoring the stitch creation problem in detailed routing may lead to an increase in the number of stitches. The following works proposed multiple patterning compliant detailed routers: (LIN, 2012), (MA, 2012), (CHO, 2008), (YUAN, 2009), (LIN, 2010), (GAO, 2010), (LEI, 2014) and

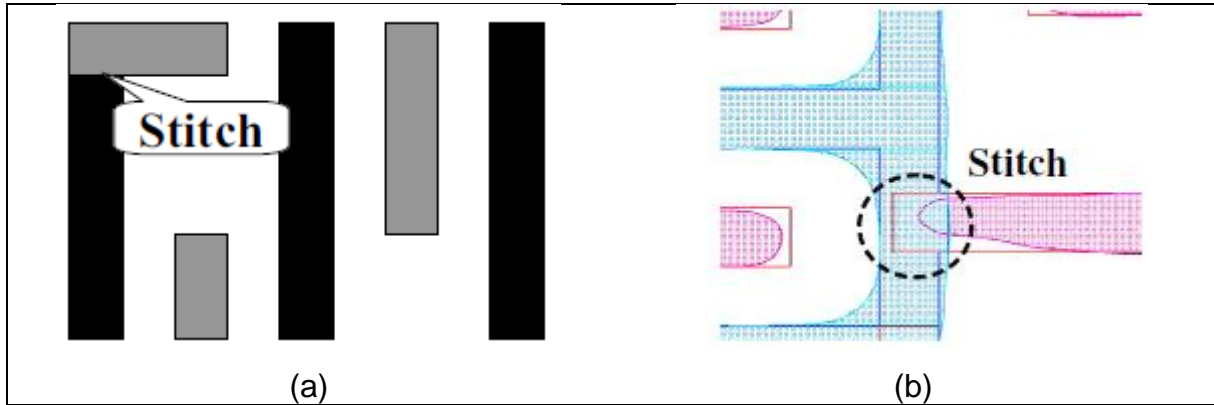


Figure 20 – Illustration of the layout decomposition and stitch generation. Black and grey represent the two masks. In (b), the rounded objects are the resulting printed features. Source: (CHO, 2008).

(AHRENS, 2015). The basic strategy of handling multiple patterning in these works, excluding (AHRENS, 2015) and (LEI, 2014), is to color the metal layers features (one color for each mask) and to penalize stitches in the path search. In (AHRENS, 2015) and (LEI, 2014), this coloring control is simplified by assign colors to routing tracks. Each feature in a routing track is encouraged to have the routing track color. Wires with colors different from the routing track color are penalized in the path search. Considering that the only way to form stitches is with jogs (i.e., non-preferred direction wires) of adjacent tracks, stitches are forbidden in (AHRENS, 2015), since jogs between adjacent tracks are not allowed. Thus, stitch control is simplified.

Another multiple patterning technology is Self-Aligned Multiple Patterning (SAMP). One of the main advantages of SAMP over other multiple patterning approaches is that SAMP presents better overlay control. Overlay is the mask misalignment during manufacturing, which may cause distortions in the printed features and yield loss.

The same way as other multiple patterning approaches, it is ideal to make detailed routing aware of SAMP. However, the SAMP is more complex, presenting other masks (i.e. trim and cut masks), such that the mask features do not represent directly the printed layout features. This makes SAMP compliant detailed routing even more challenging than other multiple patterning techniques. Examples of SAMP compliant detailed routers are (MIRSAEEDI, 2011), (DU, 2013), (LIU, 2016), (DING, 2017) and (DING, 2018).

(MANTIK, 2018) presented a critic to these works claiming that “the problem instances adopted in these works are much smaller than the real industrial designs.”, and this is one of the motivations for the creation of the ISPD18 Contest, which

provides more realistic benchmarks, derived from industrial chips. The ISPD18 benchmarks are not from technology nodes that require multiple patterning though. Thus, the scope of the current work does not consider multiple patterning approaches, since it uses the ISDP18 benchmarks.

3.2 Path Search

Almost all detailed routing approaches rely on path search algorithms. In graph theory domain, the path search is a problem which consists in finding a path between two vertices, or two sets of vertices S and T (source and target sets) in a graph. Some algorithms can handle multiple source-target (s - t) vertices, while others can handle only a single s - t pair. In detailed routing, the graph is a grid-graph, and the path cost is desired to be minimized. This grid graph is only conceptual, and it does not to be explicitly stored, as discussed in section 3.1.7.1. In this work, the term “path search” is also used for line probe methods, such as Hightower’s algorithm (HIGHTOWER, 1969), which can work in a plane of continuous coordinates, meaning that the search is purely geometrical (i.e., it is not a graph search).

3.2.1 Lee's algorithm

The Lee’s algorithm (LEE, 1961), also known as the Maze Router, is a Breadth-First Search (BFS) algorithm specialized for a grid graph context. Figure 21 illustrates how the algorithm works. It starts the search at the source point s . From this point, the adjacent points (with the exception of the diagonals) are visited and labeled with values, representing the known distance from the points to s (Figure 21a). For each of these points the same process repeats, generating successive expansive waves, until target t is found (Figure 21b and c). Then, the algorithm starts the path recovery phase (Figure 21d). Once the target point is reached, it is added in the path, and any adjacent point with a value less than it is also added to the path. This point is selected and the procedure continues until the source point is reached. Then all the labeled points are unlabeled. Lee's algorithm ensures that a path is found, if any exist, in addition to ensuring that the path is optimal.

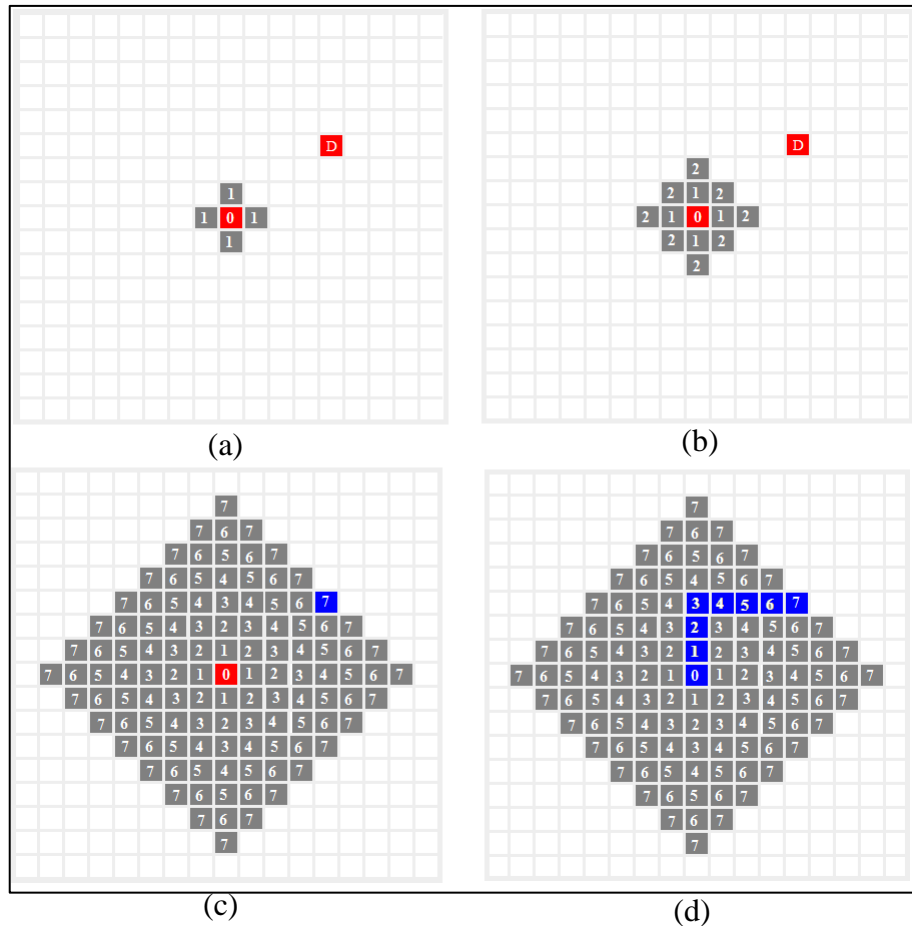


Figure 21 – Illustration of some steps of the execution of Lee's algorithm. In red are the source and target points. (a) The four neighbors of s are labeled. (b) Neighbors of neighbors of s are labeled. (c) After some iterations, the destination is reached. (d) The path is found.

3.2.2 A* Algorithm

Lee's algorithm expands the search in all directions. In situations where the routing region is not full of obstacles, expanding the search in a direction contrary to the target point is pointless. This behavior causes Lee's algorithm to have a high runtime and memory consumption. In order to improve this, Rubin (RUBIN, 1974) applied the A* search (HART, 1968) technique in Lee's algorithm. Although A* is a speedup technique of Dijkstra's algorithm, the term "A* algorithm" is commonly used in detailed routing, and in this work, as Dijkstra's algorithm with A* technique. This algorithm consists in an implementation of Dijkstra's algorithm using a heuristic to bias the search towards the target point. In Dijkstra's algorithm, and consequently Lee's algorithm¹, the cost of a search node n is the cost of the known path between the source point and n . In A*, the cost of the search nodes is given by the function $f(n) =$

¹ The application of Dijkstra's algorithm in a grid graph, with uniform cost edges, provides the same search behavior and results as Lee's algorithm

$g(n) + h(n)$, where $g(n)$ returns the known cost of n to the source point, as in Dijkstra's algorithm, and $h(n)$ is a heuristic function that returns the estimated path cost from n to the target point. Thus, $f(n)$ cost means the least cost, known so far, that a s - n - t path¹ can have. Since the function h estimates the cost of the node to the target point, the total cost, given by f , is called potential cost of n .

In a grid graph, h is usually implemented by the L_1 (i.e Manhattan) distance. The L_1 distance (or L_1 norm) is the shortest possible distance between two points on a grid graph. The L_1 distance of two points (x_1, y_1, z_1) and (x_2, y_2, z_2) in a grid graph is given by

$$L_1((x_1, y_1, z_1), (x_2, y_2, z_2)) = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|.$$

In detailed routing, it is normal that the via usage is discouraged. Thus, vias normally present cost penalties to the path search. Thereby, the value $|z_1 - z_2|$ of the L_1 function may be easily modified by the cost of using a via. Also, if $z_1 = z_2$, the cost of two vias may be used, if the best possible path necessarily must use two vias.

Note that the h function is a heuristic to reduce the search space (improving runtime), not a heuristic to obtain the optimal path. Thus, the optimal path of A^* is still assured, provided h is *admissible*. This is true for the original version of A^* , which admits that the search nodes may be reopened, but in the common applications of A^* , such as in routing (and the one described here), the closed nodes are never reopened. Thus, in this scenario, h must also be *consistent*. A heuristic function is admissible if it never overestimates the real path cost it is estimating. It is consistent if $h(a) \leq \text{cost}(a, b) + h(b)$, that is, the estimated cost from a to the target is not higher than the cost of reaching a neighbor b of a plus the estimated cost from b to the target. These properties ensure that the $f(n)$ values never decrease during the search.

The pseudocode of the algorithm A^* is shown in Figure 22. The pseudocode of A^* is the same of Dijkstra's algorithm, since the only difference is the function used to sort the nodes in the priority queue, i.e., A^* is Dijkstra's algorithm if $h(n)$ is set to 0. However, the presented pseudocode is closer to a more practical implementation than the one defining Dijkstra's algorithm (CORMEN, 2001). For example, in Dijkstra's algorithm, the graph vertices are labeled, which requires all vertices are initially labeled as ∞ . In the presented pseudocode, instead of labeling the vertices of the grid graph,

¹ That is, a path starting at s , passing by n and ending in t

ALGORITHM: A* (Point s , Point t)

```

1  Add node( $s$ , null) in  $O$ 
2  while  $O \neq \emptyset$ 
3     $n \leftarrow \text{pop}(O)$ 
4    if  $n.\text{point} = t$ : return path( $n$ )
5    for each neighbor point  $v$  of  $n.\text{point}$ :
6      if there is a node  $v'$ , with  $v'.\text{point} = v$ , either open or closed:
7        if  $f(n) + \text{cost}(n, v) < f(v')$  :
8          Update  $v'$  in  $O$ 
9        else Add node( $v$ ,  $n$ ) in  $O$ 
10 return null

```

Figure 22 – Pseudocode of A* algorithm.

which is not even explicitly stored in detailed routing, it creates search nodes that reference the grid points, which are the grid graph vertices. These nodes also store some information, as g , optionally h , and a parent node reference. Initially, the source node is created (function *node*) and added in the *open set* O . The second parameter of the function *node* is the parent node. In the case of s , since it has no parent node, it has a null parent. The nodes in the open set are called open nodes, and nodes not in this set are called closed (or visited) nodes. This set represents the current boundaries of the search. The open set is implemented by a priority queue, usually a heap. In each iteration, the algorithm obtains and removes the node of least $f(n)$ cost from O (line 3). If the t is reached (line 4), the path is returned. Next, the node n is said to be *expanded* (lines 5 - 9). For each point v adjacent to the point of n ($n.\text{point}$), ignoring diagonals, the algorithm tries to add nodes in these points in the open set. If there is already a node created in v and it is in the open set and its cost is higher than the cost of the new candidate node (i.e., over v), then the cost and parent reference of the already existing node are updated, as well as the position of this node in the priority queue O . If the already existing node is not in the open set (i.e., it is closed), its cost is guaranteed to be lesser or equals than the candidate node's cost, by the properties of the algorithm. If the candidate node's cost is higher or equal to the already existing node, the candidate node is not added in O . If there is no node either open or closed referencing v , then the candidate node is inserted in O . This is the relaxation step of Dijkstra's algorithm. A good way to control the open and closed nodes is to maintain a "satellite" data structure, as a hash table, of all created nodes so far.

Figure 23 shows a comparison of the search space of A* algorithm (a) and Lee's algorithm (b). Note that if the h function of A* was implemented to always return 0, the algorithm would behave exactly as Lee's algorithm. It is evident the difference of the

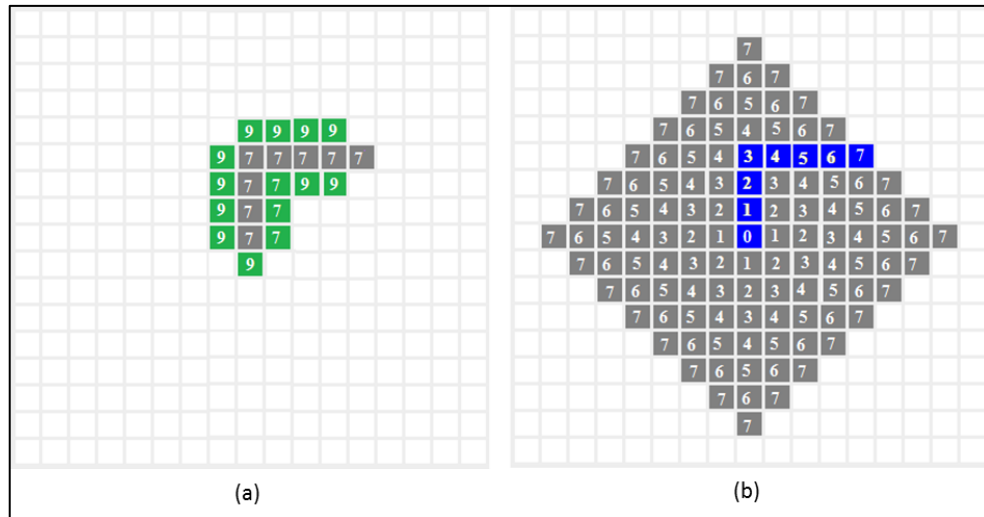


Figure 23 - Comparison of the search space of A* (a) and Lee's algorithm (b). In (a) the gray points are the closed points, and the green points are the open nodes. The numbers represent the costs of the nodes.

search space of both algorithms. By iteratively obtaining the nodes with cost 7, A* quickly converged to t , whereas the Lee's algorithm loses processing time performing waves of expansion in all directions. The use of a heuristic function in A* enables the algorithm to look first where it is most promising, implying a large performance gain.

Another important consideration regarding A* is the tie breaking criterion of nodes with the same cost. The way the algorithm decides the priority of these nodes can drastically affect the performance of the algorithm. Note that in Figure 23a there are nodes with cost 7 that are open. Depending on the tie-breaking criterion, these nodes could be expanded, as well as all their successors, forming a rectangle of visited points, which would be similar to the search space visited by Lee's algorithm. A good tie breaking criterion is to give priority to nodes with the highest g value, or nodes with lowest h value. This way, nodes that are further from the source point, and consequently closer to the target point, will be chosen first. This gives the algorithm a Depth-First Search (DFS) behavior.

In detailed routing, it is desirable that the path search algorithm handles multiple source and target points. The A* algorithm naturally supports this without major modifications in the original algorithm. The algorithm receives two sets of points S and T , representing the source and target points, respectively. Each node now has a reference to its *current* target point. In the node creation, and during the execution of the algorithm, this reference is updated. The criterion for choosing a target is the estimated cost of the node to the target, using the h function. If the cost of a newly created node, using its parent's target, is equals to its parent's cost, it is not necessary

routing, since the path searches are constrained by the global routing guides, which may present many detours. One of the contributions of the current work is related to making h aware of the shapes of the global routing guides. This will be discussed in section 6.3.

3.2.3 Line Probe Algorithms

The previous algorithms may be denominated as maze search algorithms. They expand the search point by point, guarantee the optimal path, but suffer from runtime. Even the A* algorithm, with its large reduction of expanded nodes, faces the same problem, since its complexity depends on the distance between the source and target points. Thus, maze search algorithms may be unfeasible for finding long connections. To overcome this problem, Mikami and Tabuchi (MIKAMI, 1968) developed the first path search algorithm that uses line segments instead of points. Rather than traversing all points in a row, the algorithm creates a line that extends itself until reaching an obstacle or the end of the search space. Thus, the complexity of this class of algorithms does not depend on the distance between the points of origin and destination, but on the number of lines created. In general, algorithms of this class do not guarantee the optimal path, but are faster than the maze search algorithms. Mikami and Tabuchi's algorithm guarantee that a path is found, if one exists. Another drawback of the line probe algorithms is that they don't work with variable grid costs, which is easily handled by the maze search algorithms.

The basic procedure of the Mikami and Tabuchi's algorithm is the following. The source and target points are called *base points*. Initially, two perpendicular lines are created over the base points. These lines are called level 0 lines. The lines are extended until reaching an obstacle the search space limit. If a line from the target point intersects a line from the source point, the path is found. If this does not occur, new base points are created on the generated lines. At each iteration, for each base point, a line of level i (perpendicular to the lines of level $i-1$) is created over the point. When all points on the four leading lines are occupied by base points, a line is selected so that new base points can be created over it. Figure 25 shows an example of the Mikami and Tabuchi's algorithm execution.

Another algorithm that works very similarly to the Mikami and Tabuchi's algorithm is Hightower's algorithm (HIGHTOWER, 1969). The difference of this

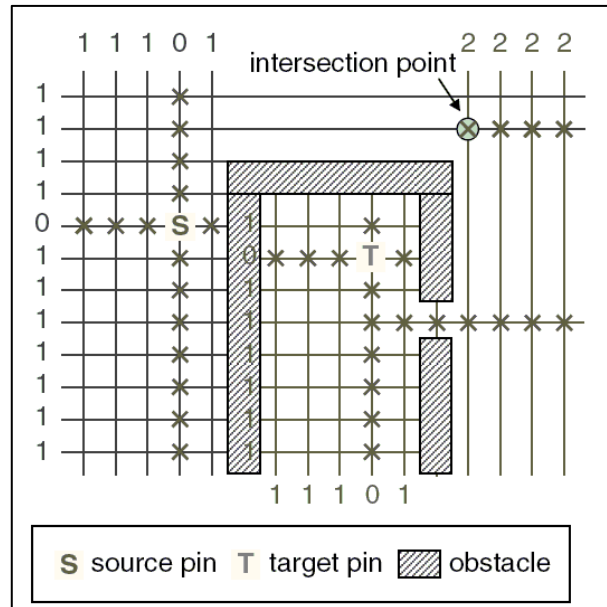


Figure 25 – Illustration of Mikami and Tabuchi's algorithm. The points marked with \times represent the base points. The numbers refer to the line levels.

algorithm is that it considers only lines that detour the obstacles. In addition, each line may have a maximum of two base points. Because of this restriction, the Hightower algorithm does not guarantee that any path is found. To handle this, it is possible to use backtracking techniques to be able to choose the appropriate base points for the path to be found. However, this leads to an increase in processing time, which can be almost as high as the time of Lee's algorithm (CHEN, 2009).

3.2.4 A*-interval-based Path Search

Consider Figure 26. In (a), the s - t path resulting from the application of the algorithm A* is shown. The resulting path is made of two segments, and was constructed by several expansions of nodes with cost 9. Note that the cost of nodes in each segment has never changed. This is a common behavior in A*, which was exploited by Hetzel (HETZEL, 1998). In such cases, it is not necessary to label each node of a path segment. If a sequence of nodes belonging to the same segment has the same cost (Figure 27b), these nodes are redundant for the search, and can be merged to a set containing them, which is labeled with the nodes value (Figure 27c). In Hetzel's algorithm, this set is represented by an interval. Thus, Hetzel's algorithm can be viewed as an A* algorithm such that its search nodes are intervals of nodes of the traditional A*. Since the search space is a grid graph, there is implicit information on the graph geometry that enables to calculate such intervals in constant time. Using

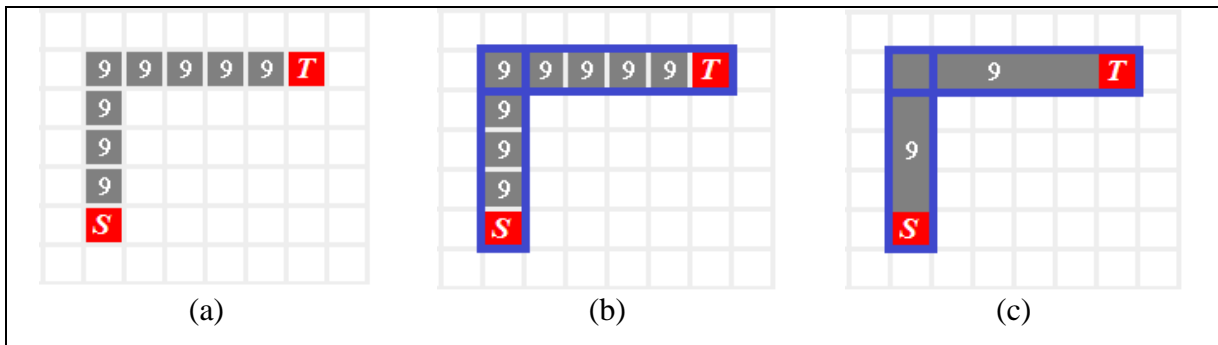


Figure 26 – Illustration of the principle of merging a group of redundant nodes into intervals. Costs are given by $f(n)$. (a) Path resulting from the application A* algorithm. Two groups of redundant nodes (b) are merged in two intervals (c).

a function h that is based on the L_1 distance, the intervals align with the target point coordinates. Note that Figure 26 has the objective of presenting the idea of merging sets of nodes in intervals, and for this it was used an example in a 2-dimensional scenario, without major compromises with the mechanics of the algorithm.

When proposed in (HETZEL, 1998), the algorithm had some restrictions, working only with the L_1 distance as the h function. It considered that the routing tracks of a layer matched tracks of different layers with the same preferred direction, that is, that the pitch was constant in all layers. Later, Peyer et al. (PEYER, 2009) made the algorithm more generalized, allowing variable pitches. The BonnRoute tool (GESTER, 2013) uses the updated version of this algorithm, and this is the version presented here.

The main loop of Hetzel's algorithm is almost the same of A*: the best interval is chosen, it is removed from the open set, and it is expanded, generating all adjacent intervals, which are put into the open set, and so on, until an interval chosen cover a *target* point. The $f(n)$ ties are decided by giving priority to the interval which covers that *furthest* point from *target*. Figure 27 shows an example of an interval expansion. Considering that the h function is using L_1 distance, the interval containing the source point would be $[source.x, target.x]$. However, since there is an obstacle, the interval ends at the obstacle adjacency. When an interval is expanded, it generates intervals at points neighboring the expanded interval. As in A*, it is necessary to check whether the newly created intervals can actually be opened. If a newly created search-interval intersects an already existing search-interval, interval subtractions are performed in order to determine which parts of which intervals will remain. The criterion to determine this is the same used in the relaxation step of A* (lines 7-10).

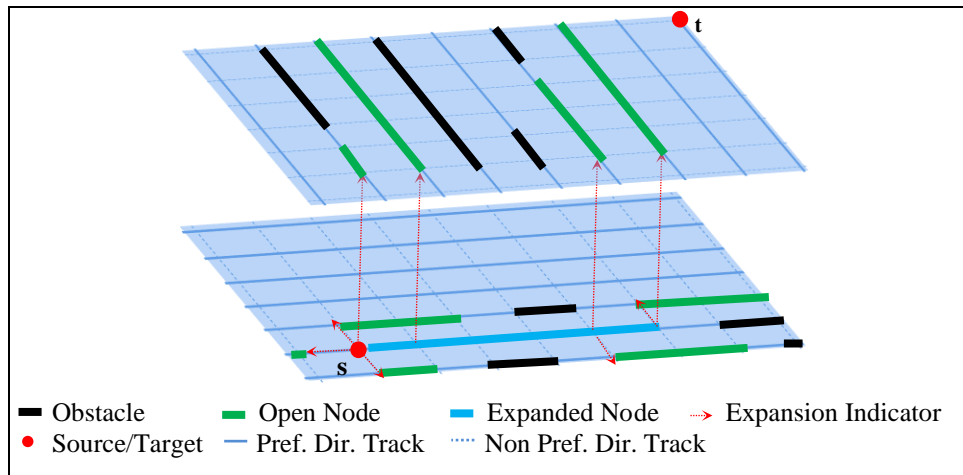


Figure 27 – Illustration of the interval expansion of Hetzel's algorithm

In the best and average case, the approach proposed by Hetzel shows a speedup w.r.t. A^* of at least 6 (GESTER, 2013). In (GONÇALVES, 2017), a speedup of 20 was achieved. This is due to the fact that Hetzel's algorithm complexity is affected by the number of intervals and an interval usually contains many points of the traditional A^* search. In the worst case, when most intervals have a single point, the algorithm behaves like A^* . The algorithm guarantees the optimal path, as in the A^* search. Hetzel's algorithm can be seen as a hybrid of maze search and line probe approaches.

3.2.5 Improved Heuristic Function for A^* -based Path Search in Detailed Routing

A^* -based path search is widely used in detailed routing. As mentioned earlier, the speedup of A^* depends on its heuristic function h . The more realistic is the estimation, the more the algorithm takes a Depth-First-Search (DFS) behavior towards the target node and, consequently, the faster it ends.

In detailed routing, the L_1 distance is the most straightforward way to implement the h function. When $h(n)$ uses L_1 , it implicitly predicts that the path from n to the target node is of L/Z shape. However, the tunnels (i.e. global routing guide sections) have shapes that force the paths to perform detours, such as an U shape path, as in Figure 28. In these cases, using L_1 distance makes the path search to take a Breadth-First search (BFS) behavior, which is slow. On the other hand, if h is implemented such that it is aware of the tunnel shape, it will provide a more realistic lower bound, enabling the search to perform a DFS.

In order to handle this problem, Peyer et al. (2009), proposed a method to preprocess the tunnels, before the path search, computing more realistic lowerbounds

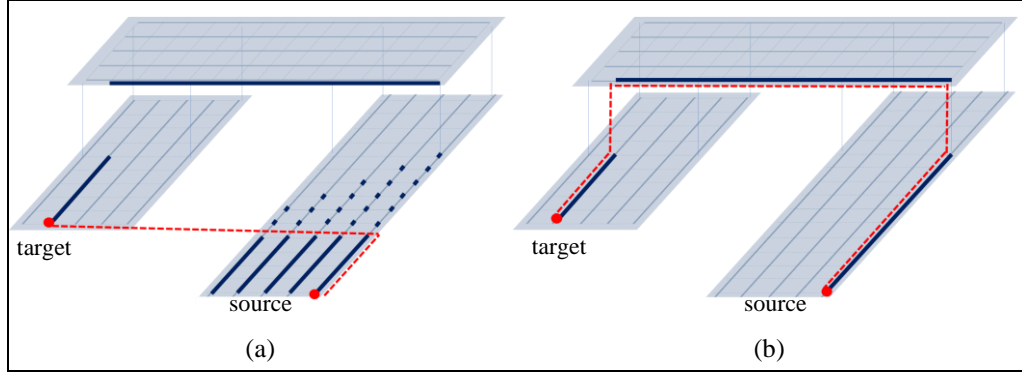


Figure 28: Illustration of the expanded search nodes using L_1 distance (a) and TL (b), using an A*-interval-based path search (as Hetzel's algorithm), restricted by a tunnel of 3 rectangles. Each dark blue line segment represents a search node (interval). Red dashed lines are the cost estimations. The cost of a non-preferred graph edge is 3 times the cost of a preferred edge.

for the h function. During the path search, the algorithm used the precomputed information. The key idea in the preprocessing step is to partition the tunnel rectangles in sub rectangles such that it is possible to express relatively simple lowerbound functions for these rectangles. The lowerbound of a point in a rectangle is given by the function

$$d(x, y) = c_1(x - x_1) + c_2(y - y_1) + \delta(c_1, c_2).$$

c_1 and c_2 are the edge costs for stepping in x and y directions, respectively, and x_1 and y_1 are the lower left corner coordinates of the rectangle. The edge costs consider penalties for jogs and also negative values of the default edge costs, or 0. For example, considering a preferred direction edge has cost 1 and a jog has cost 4 the possible values of these coefficients are 1, 4, -1, -4 and 0. $\delta(c_1, c_2)$ is an offset cost to the target component of the path search.

Figure 29 shows an example of how manipulating the coefficients c_1 and c_2 obtains offsets to different positions within the rectangle. This may be performed by changing the signal of these coefficients or by making them 0. The expressions are omitting the offset value $\delta(c_1, c_2)$. However, some additional clauses are needed to enable offsets to x_2 and y_2 . The values of these clauses are added in the offset function $\delta(c_1, c_2)$. This offset function represents the offset from the corresponding offset point to the target component of the path search.

The problem with this method for calculating lowerbounds is that it calculates a limited number of offsets (more exactly 8). Given the rectangle of Figure 29, it is possible that the best cost to the target component uses a point different from any of

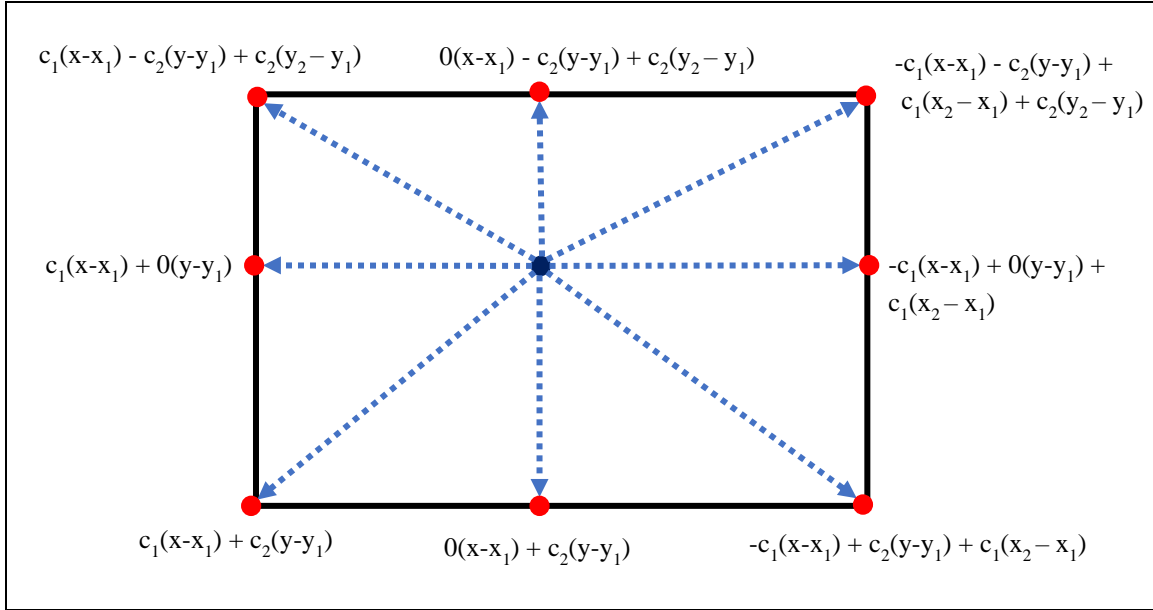


Figure 29- Illustration of the manipulation of the coefficients and the corresponding offset points (in red). In dark blue is the point where the lowerbound will be measured. x and y are the point coordinates and x_1, x_2, y_1 and y_2 are the rectangle bounds ($x_1 \leq x_2$ and $y_1 \leq y_2$)

the shown points. Thus, in order for this method to work properly, it is necessary to partition the tunnel rectangles into smaller rectangles such that it is guaranteed, that there is no other offset point with a better offset cost. Figure 30 shows an example of this rectangle partitioning. The partitioning depends on the structure of the target component. The exact criterion to determine the rectangles was not mentioned in (PEYER, 2009), but it is clear that it allows different sets of target points to be present in the same rectangle, such that it is necessary to use at least two functions to determine the lowerbound of a given point. For example, using the same coefficient values mentioned earlier, and considering the rectangle R in Figure 30 has 4×4 pitches, the lowerbound of a point (x, y) in R is given by the minimum of the two functions $d(x, y) = 4(x - x_1)$ and $d(x, y) = -4(x - x_1) + (y - y_1) + 16$. Note that horizontal edges receive jog penalty (cost 4). Each function is targeting one set of target points. The first function targets the left border and takes the form of the function $c_1(x - x_1) + 0(y - y_1)$ from Figure 29. The second function targets the lower-right corner of the rectangle and takes the form of the function $-c_1(x - x_1) + c_2(y - y_1) + c_1(x_2 - x_1)$ from Figure 29.

Using a modified version of Dijkstra's algorithm, considering as graph vertices the partitioned rectangles, the distance functions d are propagated into the neighboring rectangles, creating the offset values $\delta(c_1', c_2')$ on them. The rectangles containing the target points have $\delta(c_1, c_2) = 0$ in the start. These offset values are given by the

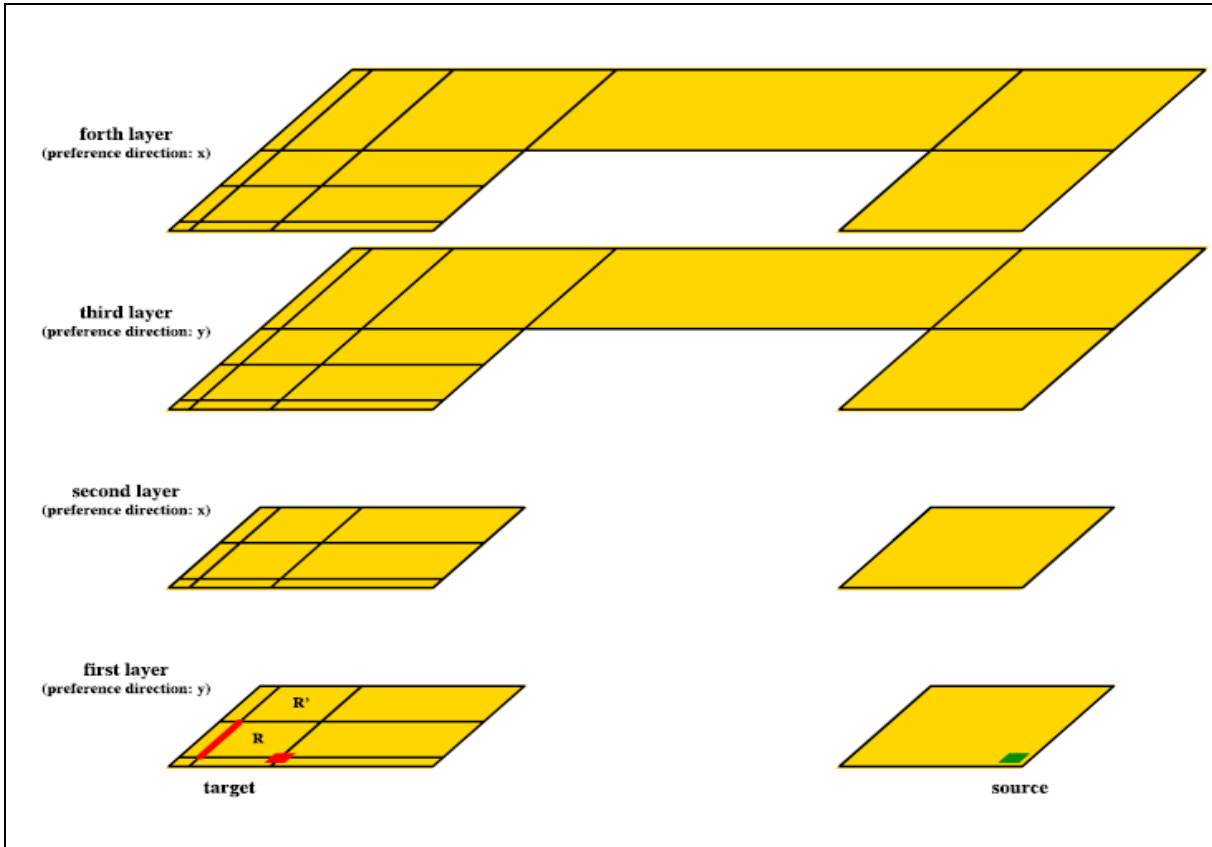


Figure 30- Illustration of the rectangle partitioning in a tunnel. Source: (PEYER, 2009)

minimum cost to reach the projected rectangle from a red point in Figure 29 corresponded to the type of the distance function that is being propagated. For each of these newly created offset values $\delta(c_1', c_2')$ there is also a corresponding function d , with new coefficients. Thus, considering R and R' of Figure 30, the function $d(x, y) = 4(x - x_1)$, which is targeting left target border of R , induces an offset $\delta(4, 1) = 0$ in R' and the other function, targeting the lower-right corner of R , induces an offset $\delta(-4, 1) = 4$. Their corresponding functions in R' are $d(x, y) = 4(x - x_1) + (y - y_1)$ and $d(x, y) = -4(x - x_1) + (y - y_1) + 20$, respectively. Note that these functions are of the type of the lower-left and lower-right ones shown in Figure 29. At the end of the execution of the algorithm, each rectangle has a set of distance functions that are used to calculate the lowerbound during the path search.

3.2.6 Design Rule Aware Path Search

One of the biggest challenges of detailed routing is the design rule handling. If the router tries to solve DRVs in routing postprocessing, many DRVs will not be solved. Thus, it is necessary to adopt a correct-by-construction approach. Making the path

search algorithm aware of some design rules is essential for a good design rule handling.

In (CHANG, 2013), it is proposed MANA, an A*-based maze algorithm handling the minimum area (or min area) design rule. The min area rule requires a metal shape, of a given routing layer, to meet a minimum area. A straightforward approach to solve min area violations is by a post-processing, after routing: pieces of metal which violated the min area requirement are extended. However, it is possible that there is no space available for such extensions, and thus, new violations are created. This can be avoided if the path search algorithm is aware of this design rule.

The approach in (CHANG, 2013) is to make each search node consider path segment extensions to meet the min area, when necessary. When a node is created, it is not known whether it is necessary to do a min area extension or not, since a path arising from this node may still meet the min area. Actually, a single node may originate many paths, and some of them may meet the min area, while others do not. Considering this, each node has a minimum and a maximum cost. The maximum cost is considering the min area extension, and the minimum cost is not considering it. Nodes are ordered in the open set by the minimum cost. In practice, the algorithm only knows that an extension will be actually needed when a node expands creating a descending node in an adjacent layer. In these cases, the maximum cost of the node with the extension is used to calculate the cost of the descending node.

This separation of minimum and maximum costs is to make the algorithm guarantee the optimal path, considering the extensions. If the maximum cost is ignored, and if a node with a better (i.e., lower) cost excludes other node with a higher cost, then this may cause the algorithm to not find the optimal path, because this better node may lead to a path that requires this node to present an extension, but this is only known *after* the node expands. This double cost implies that different nodes may share the same point, but it is possible to prune some nodes. If the minimum cost of a node is equal or higher than the maximum cost of other node sharing the same point, then the first node is pruned. The MANA algorithm guarantees the optimal path, considering the min area extensions, but since it is a maze search algorithm, the runtime is not favorable, as its application in practice. The algorithm proposed in (CHEN, 2019a, 2019b) is basically the same as MANA, except that it allows min area extensions with violations to exist, with a penalty in the node cost.

Another approach to handle the min area rule in the path search is proposed in (AHRENS, 2015). This work proposes a multi-label path search to handle min area rule and wire coloring for multiple patterning approaches. The strategy is to handle the min area rule by constructing a grid whose pitch already guarantees that any wire segment will meet the rule. This is problematic since it can lead to an increase in WL and present routability problems. The proposed algorithm was shown to present unfavorable runtime, being worthwhile only in some circumstances.

3.3 Pin Access

Due to the irregular shapes and locations of the pins, the grid points are often outside the pin shapes. In these cases, since the path search runs in a grid, the available valid connection points to the pins are those at the pin's surroundings. Thus, it is necessary to create off-track (i.e. out of grid) pin access paths (PAPs) to connect the access points to the pin. The challenge is to create these PAPs without (or minimizing) DRVs. Figure 31 shows some pin access situations with DRVs. Even when accessing grid points inside the pin it is still possible to occur violations, as shown in Figure 31b.

The pin access problem can be classified in intra-cell and inter-cell pin access. The intra-cell pin access aims to provide short connections between the pins and the neighboring access points, handling the DRVs that commonly emerge from such connection and with nearby pins and possibly with other PAPs. The inter-cell pin access also avoids conflicts between intra-cell PAPs, but focuses on extending pin access paths beyond the cell, avoiding conflicts with PAPs of other cells. The PAPs

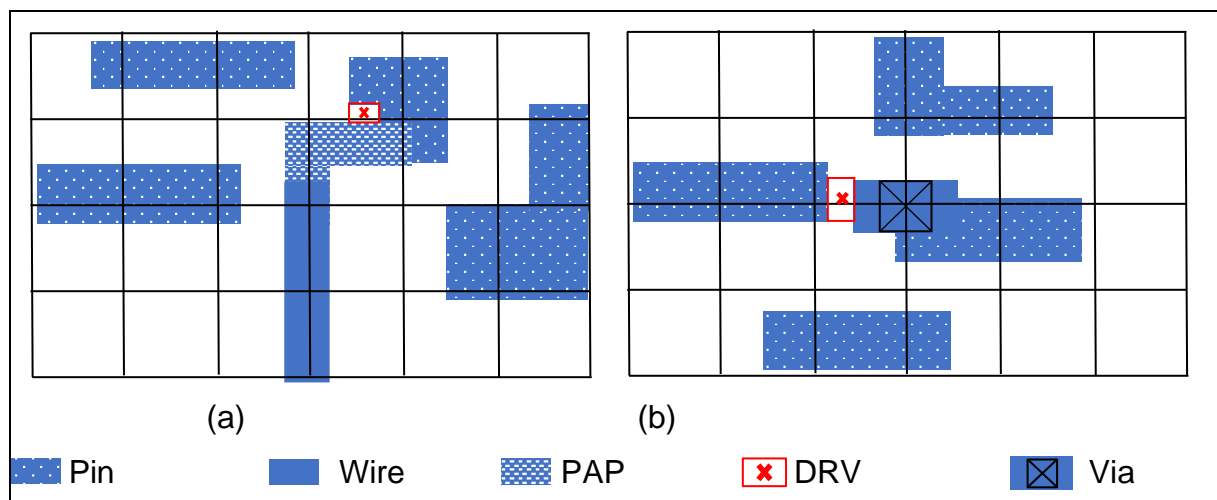


Figure 31- Illustration of the pin access problem.

are possibly directed towards other pins of the same net, or may be connected to pins from other nearby cells from the same row.

3.3.1 Escape Routing

In detailed routing literature there are relatively few works addressing the pin access problem. The work in (OZDAL, 2009) proposed an inter-cell pin access approach, called escape routing. Escape routing aims to alleviate the routability problem caused by dense pin clusters. The pin access solution consists in assign each pin with at least one path that escape the congested area. These paths are drawn towards the other pins of the same net. Some routes are even completed, when the pins are in the same standard cell row. The method selects clusters of pins, that may contemplate more than one cell, and finds a conflict-free solution. The problem is solved using ILP. Figure 32 shows an example of escape routing solution in a cluster of three cells. All escape routes are statically implemented in the routing space before routing.

The work in (XU, 2017) also proposes an escape routing technique, but handling SAMP constraints. Ignoring the SAMP-related considerations, the main difference from

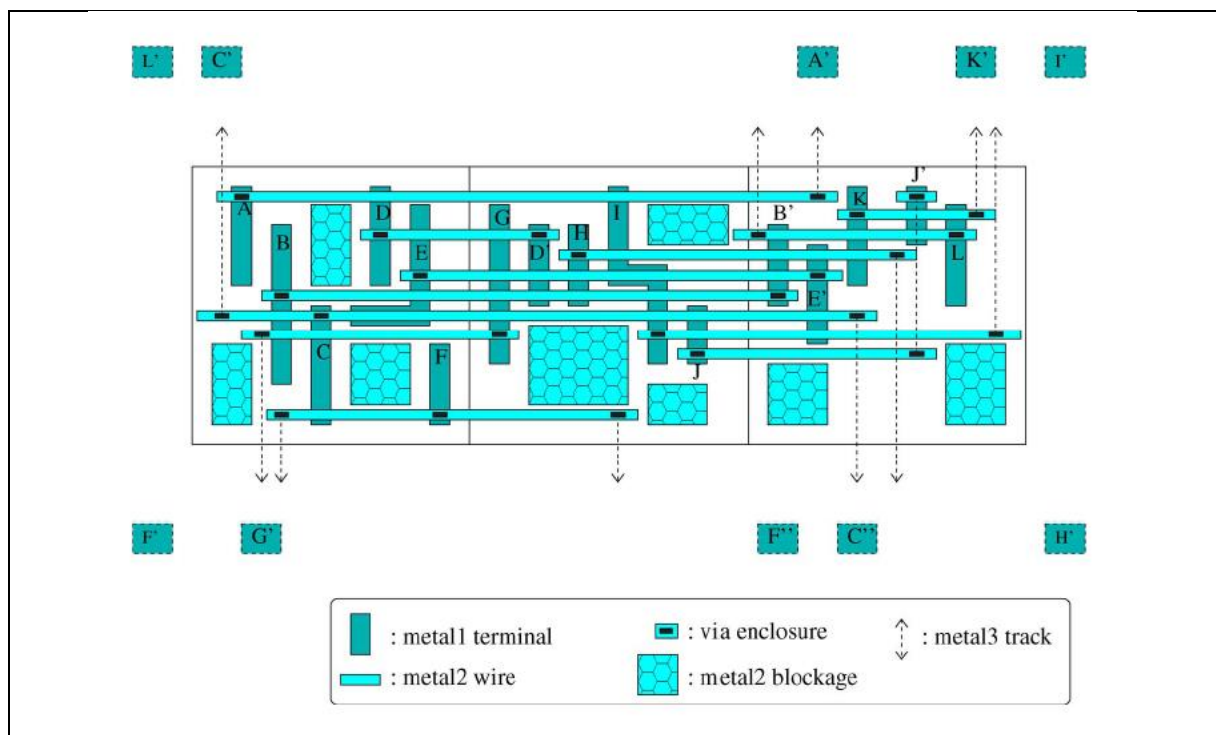


Figure 32- Illustration of an escape routing solution. The items outside the pin cluster are the corresponding connecting pins (i.e, pin X connects with X' or X'') Source: (OZDAL, 2009).

(OZDAL, 2009) is that the problem is modeled by a weighted interval assignment problem. These intervals represent the wire segments that will connect to each pin using metal 2. The intervals may be extended up to the boundaries of the bounding box of the net of the connecting pin. The problem consists in assigning one interval to each pin, such that there is not conflict between intervals of different nets and the total interval length is maximized. Entire standard cell rows are processed and the solution is obtained by using ILP.

3.3.2 Intra-cell with Conflict-Free Solution and Static Implementation

Nieberg (NIEBERG, 2011) proposed an intra-cell pin access method. The approach calculates a conflict-free pin access solution for each cell and implements all PAPs in the routing space before the routing itself (static PAP implementation). A conflict-free solution is a solution where all PAPs may coexist without causing any DRV. The PAPs are not calculated for every cell instance in the layout though. Each cell has many instances placed over the chip floorplan. The key idea is to process the cells, not their instances. However, a cell itself has not a grid alignment and, therefore, it is impossible to create pin access paths and pin blockage information, since they are related to the routing grid. Thus, it is necessary to look for all the grid alignments that a cell may assume when instantiated. Each cell may have many grid alignments, but their number is still far inferior than the number of cell instances. Also, a cell may be placed mirrored and with different rotations. However, taking into consideration only these patterns is not enough, since there are other routing objects placed in the chip area, such as power and ground nets. Thus, by considering the cell, the grid alignment, the geometric configuration and the nearby shapes, it is possible to find repeating patterns, which are called *circuitclasses* (examples in Figure 33). Thus, the pin access is performed on the circuitclasses.

Regarding the PAP calculation, the major effort is to create pin access paths without DRVs, either with any pin, with other PAPs, or with the path itself. Figure 34 shows an example of a PAP with and without violations. The PAPs are obtained by a gridless path search. Thus, an implicit Hannan (HANAN, 1966) grid is used, considering the neighboring pins and their spacing requirements. The source of each

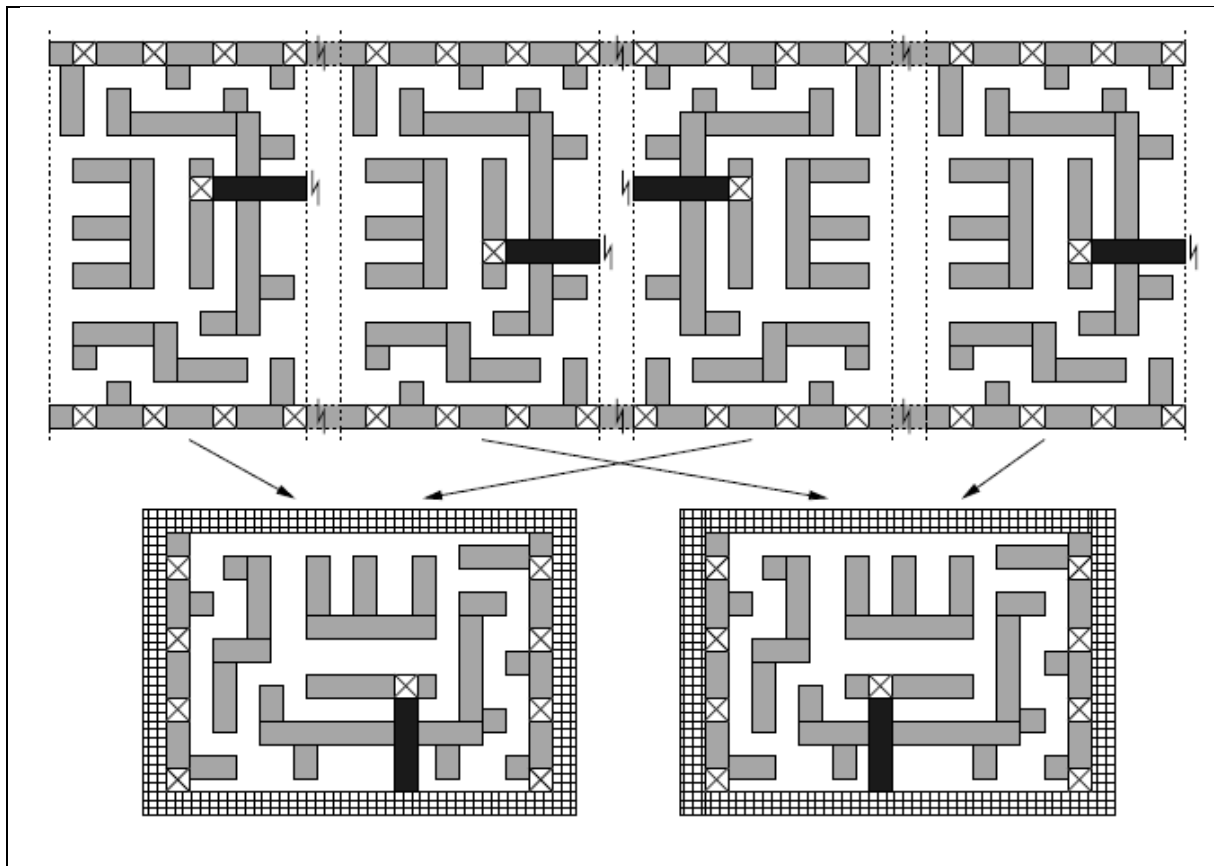


Figure 33- Illustration of *circuitclasses* (bottom) and their corresponding instances (top). Source: (NIEBERG, 2011).

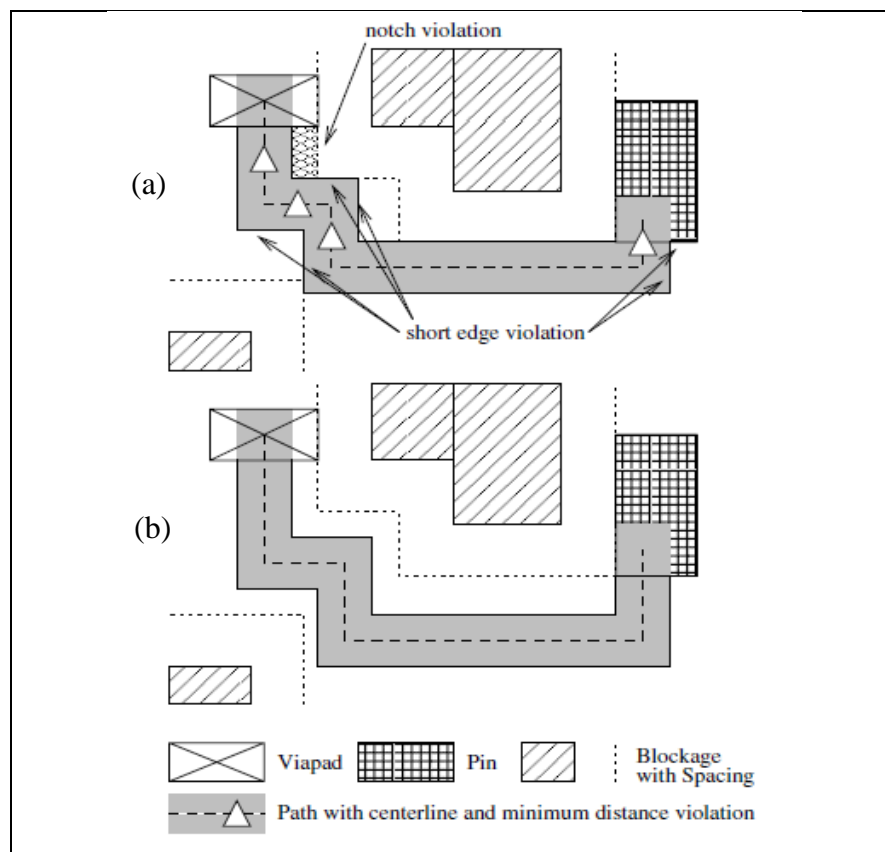


Figure 34- Illustration of PAP creation with (a) and without (b) violations. Source: (NIEBERG, 2011)

After all PAPs are calculated for all circuitclasses, they are implemented in the routing space, to avoid that wires, during routing, block the pin access points. However, neighboring cells may present conflicting PAPs with each other. In these cases, only one PAP is implemented. In routing, if a pin does not have any available PAP, then a PAP is created on-the-fly, but there is not guarantee that the PAP will be DRV-free.

3.3.3 Inter-cell with Conflict-Free Solution and Dynamic Implementation

(XU, 2016) proposed a SAMP compliant pin access technique. The approach consists in calculating all possible conflict-free pin access solutions of each cell. No solution is implemented in the routing space, before routing, like in (NIEBERG, 2011). The solution of a cell is determined and implemented dynamically, during routing.

A pin access solution considers wire extensions on the access points, in metal 2, to handle the min area rule and some SAMP-related rules. This wire extensions may cause conflicts with neighboring cells, as shown in Figure 35. Thus, an inter-cell pin access technique is used. The goal is to find a valid pin access solution for all standard cells in a row. This is accomplished by creating a graph, where each vertex represents a solution, and by finding a path in this graph, as illustrated in Figure 36. Each cell has many solutions (vertices) and each of them is linked (by a graph edge) with the solutions of the neighboring cells. The source and target vertices of this path search are virtual, and represent the beginning and the end of a row, respectively.

The source of the need to find a valid solution for an entire row is the fact that

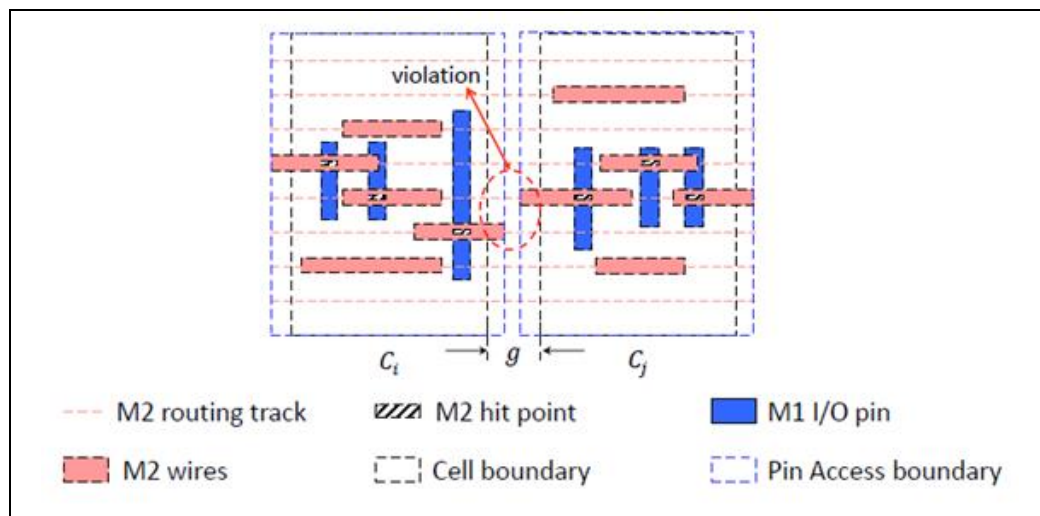


Figure 35- Illustration of inter-cell conflicts. A “hit point” is a pin access point. Source: (XU, 2016).

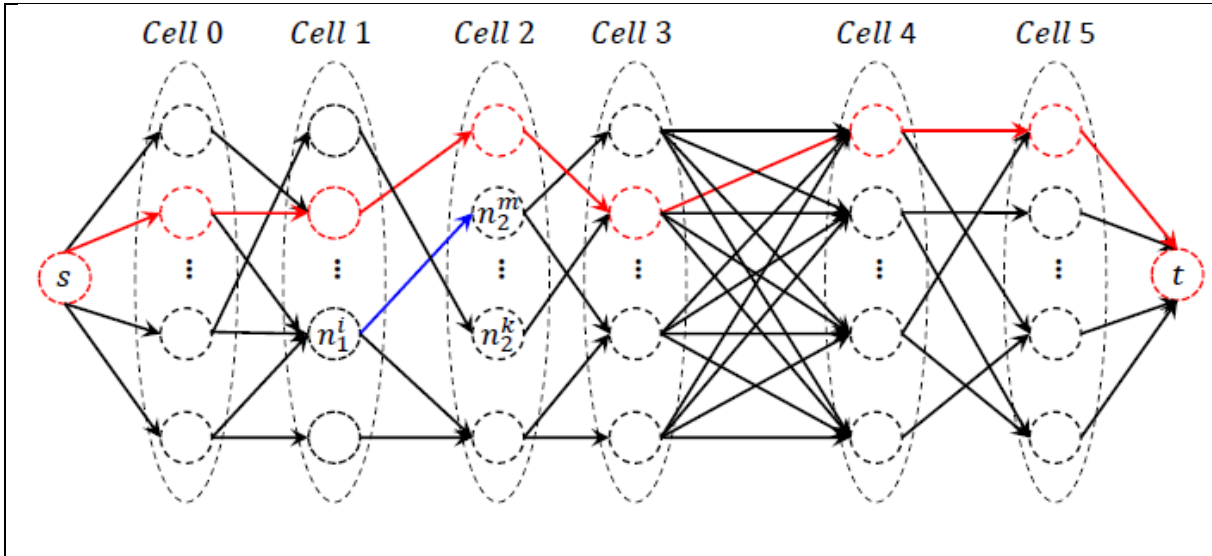


Figure 36- Illustration of finding a non-conflicting pin access solution for the entire standard cell row.

wire extensions are performed in the pin access calculation. However, this is not necessary, since they can be performed during routing. Thus, the problem of finding a pin access solution for a standard cell row can be avoided by delegating the wire extensions to the routing step.

Another important thing to note, about this approach and the one in (XU, 2017), is that they are aimed for SAMP technologies. In these technologies, the pin access is simpler due to the more regular shapes of the pins, but most importantly, due to the fact that the routing is one-dimensional on each metal layer, meaning that jogs are not allowed. Thus, off-track pin access, like in (NIEBERG, 2011), is not possible. In SAMP technologies, the pin access consists in just reaching a pin through metal 2 with a standard on-track wire and placing a via over the pin. Thus, this pin access context is not much realistic for the context of the current work, since the benchmarks used here are not of technology nodes that require multiple patterning approaches. Still, it is valid to mention these techniques, since they could be used, with restrictions and possibly drawbacks, in any technology. Although not a SAMP approach, (OZDAL, 2009) also does not report about considering the problem of finding DRV-free PAPs (considering the via connections with the pins), and the examples of pin access solutions are very similar to the ones in (XU, 2016) and (XU, 2017) regarding the simplicity for a pin access connection.

4 SmartDR Overview

This section presents an overview of the proposed detailed router, SmartDR. Section 4.1 presents the modeling of some routing information, such as how wires and vias are stored. Section 4.2 presents the routing flow.

4.1 Modeling Routing Information

The routing grid is implemented by a data structure called *grid* in this work. It stores occupancy information of all routing-related objects, such as wires, vias, pins and obstructions. Each metal layer in the grid is an array where each position corresponds to a routing track. Each routing track is implemented by an AVL tree storing intervals that represent occupied space, by any mentioned routing object, in the grid. These intervals may also be called blockages, since they form blockages for objects of different nets. Each different routing object has a specific interval type that is related to the object that induced it, but there are only two general interval types worth to mention here: *solid* and *spacing*.

Figure 37 shows the criteria to determine the occupancy of a grid point and its type. A grid point is considered occupied if a shape would cause a violation in a default wire in this point. Thus, each grid point has an occupancy zone (Figure 37a), which has the wire width. If a shape touches this zone, either with its spacing radius or with the shape itself, the grid point is marked as occupied. Let us emphasize again that this marking is done using intervals of grid points in the grid data structure. In Figure 37b a

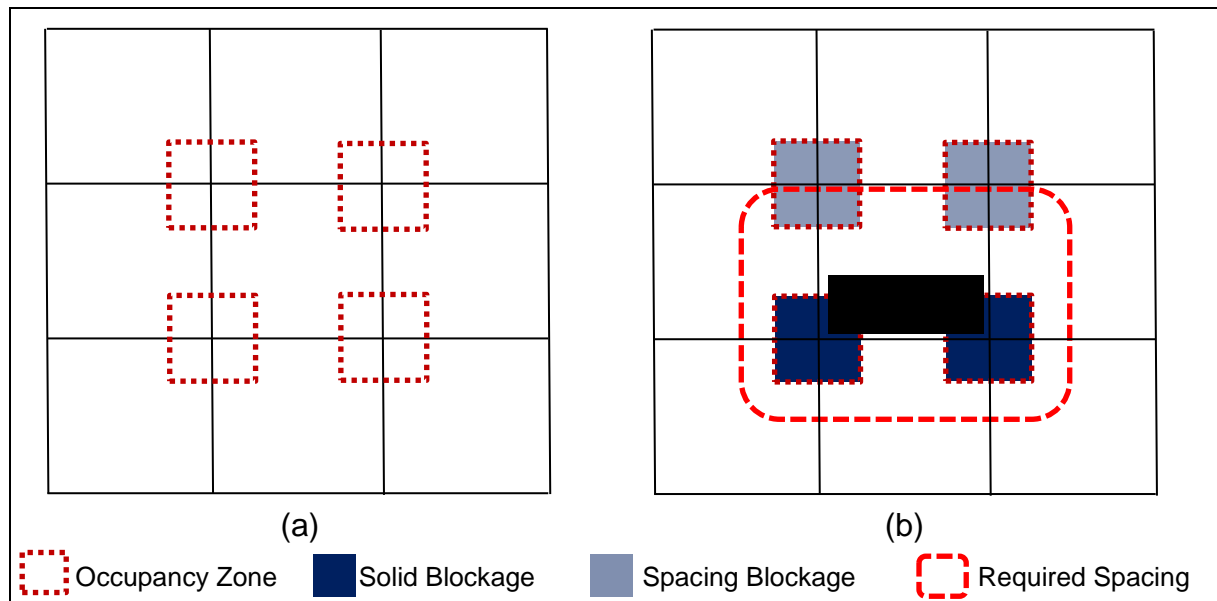


Figure 37- Illustration of grid point occupancy criteria.

shape (in black) touches 2 grid points and they are marked as solid blockages, and the spacing radius touches the upper grid points, which are marked as spacing blockages. The difference between these two types of blockages is that spacing blockages of different nets may coexist in the same grid point without causing any DRV. Thus, this makes routing less restrictive, which improves routability. The solid blockage intervals have also the information whether their tips are spacing blockages or not, to avoid the necessity of creating spacing intervals.

SmartDR presents a separate control for cut layers. The minimum distance required between two via cuts is higher than the minimum wire spacing. Without loss of generality, let us consider the vias of the lower layers (via1-via5), which cannot be placed in adjacent grid points due to the spacing requirement. A trivial solution is to store via cut blockage information is the same adopted in the metal layers, as shown in Figure 38a. However, this is not efficient, since up to 3 queries are needed to check whether the via cut can be used or not. Thus, SmartDR uses a more efficient strategy. The routing space is sliced horizontally, grouping a number of horizontal tracks, equal to the radius of the cut blockage information (3, in this case). Figure 38b illustrates this. Each slice holds an AVL tree in which the keys are the “x” coordinate of the via, and the associated data is the via itself. A single key may store more than one data (up to 2), but this is rare in practice. Each AVL tree is stored in a vector. Regarding the slicing orientation, it is irrelevant whether they are horizontal or vertical. A query in this data structure consists in an interval search in the tree, where the interval is the spacing radius of the via cut. All via cuts whose “x” coordinate intersects the interval are

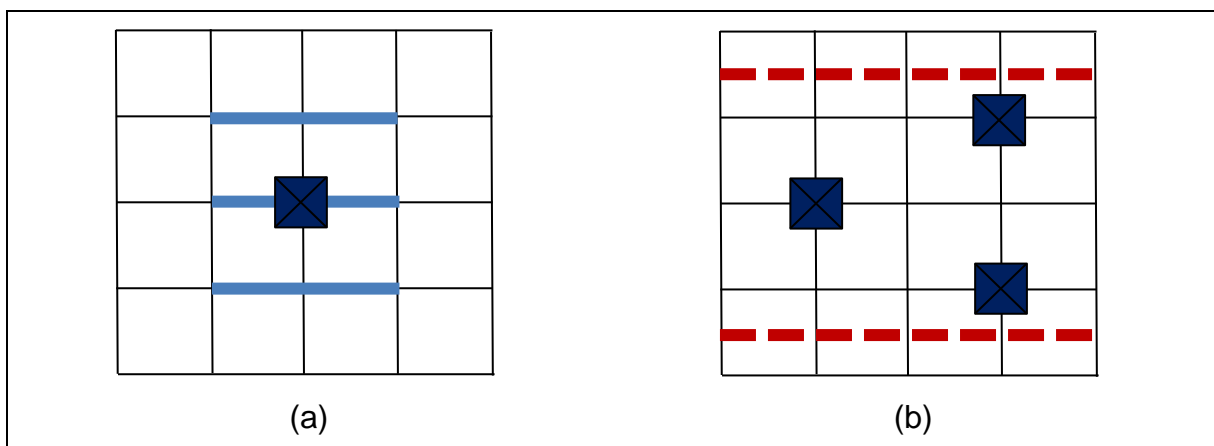


Figure 38- Illustration of trivial via cut handling (a), and the one adopted in SmartDR (b). The blue segments in (a) are the blockage intervals of the via cut. In (b), we have a slice (red dashed lines) of three tracks with three via cuts.

checked whether they cause a violation with the querying via or not.

The size of the slices is justified by the fact that, if a querying via is on the center of the slice, then one query is enough to know whether it can be used or not. If it is in other tracks, then it is necessary a maximum of two queries, one in the current slice and the other in the adjacent one. In the given example, since the middle track represents one case where it is necessary only one query, and the adjacent tracks are two cases where two queries are needed, the average number of queries is 1.66 (i.e., $5/3$ queries), against 3 queries of the trivial method mentioned earlier.

This data structure is called *via map* and is exploited by the via check feature (section 6.4.1) of the proposed path search algorithm, to provide more efficient via queries during the path search. The via storage has some other peculiarities, but they will be discussed in section 6.4.1.

4.2 Routing Flow

Figure 39 shows the routing flow of SmartDR. The router takes as input a .lef and a .def file describing the circuit and a .guide file describing the global routing guides. After parsing the files and performing initialization procedures, it processes the cells, creates pin access paths and places pin blockage information on the routing space. Not all cell instances are preprocessed though. Here, the same strategy of (NIEBERG, 2011) is used. Each cell has many instances placed over the chip floorplan. The key idea is to process the cells, not their instances. However, a cell itself has not a grid alignment and, therefore, it is impossible to create pin access paths and pin blockage information, since they are related to the routing grid. Thus, it is necessary to look for all the grid alignments that a cell may assume when instantiated. Each cell may have many grid alignments, but their number is still far inferior than the number of cell instances. A cell with a grid alignment is called cell configuration (or cellconfig, for simplification) in this work, and it represents the same concept of circuitclasses of (NIEBERG, 2011). After each cellconfig is processed, each cell instance is iterated and all blockage information related to its cellconfig is transcribed to the grid (Populate Grid step, in Figure 39).

The pin access step generates pin access paths (PAPs) without DRVs created by the geometry of the path itself (path-path DRVs), and also without DRVs between the path and the pin (path-pin DRVs). These PAPs, with some exceptions, are not statically implemented in the routing space, as usually done by pin access approaches

(OZDAL; NIEBERG; XU, 2009, 2011, 2017). During the routing phase, they are checked for legality. A PAP is legal (or available) if it can be implemented in the routing space without causing any DRV. If a PAP is available and it is chosen by the path search to be a path terminal, it is dynamically implemented in the routing space. The proposed pin access approach is explained in the next section.

Before the Standard Routing step, each multipin net is decomposed in sets of two-pin nets, using Prim's algorithm (CORMEN, 2002). The Prim's algorithm considers that each pin is a vertex and the graph is complete, i.e., all vertices are connected to each other. The cost of the edges is given by the minimum distance, aware of the global routing guide shapes, between the vertices (pins). In this step the mentioned routing guides of each net are sliced into many guides, which are called *tunnels*, one for each two-pin net. The tunnels are refined, by adding guide rectangles (GRs) in the upper layer of the GRs that use long jogs. The GRs may also be enlarged to contain all access points of a pin.

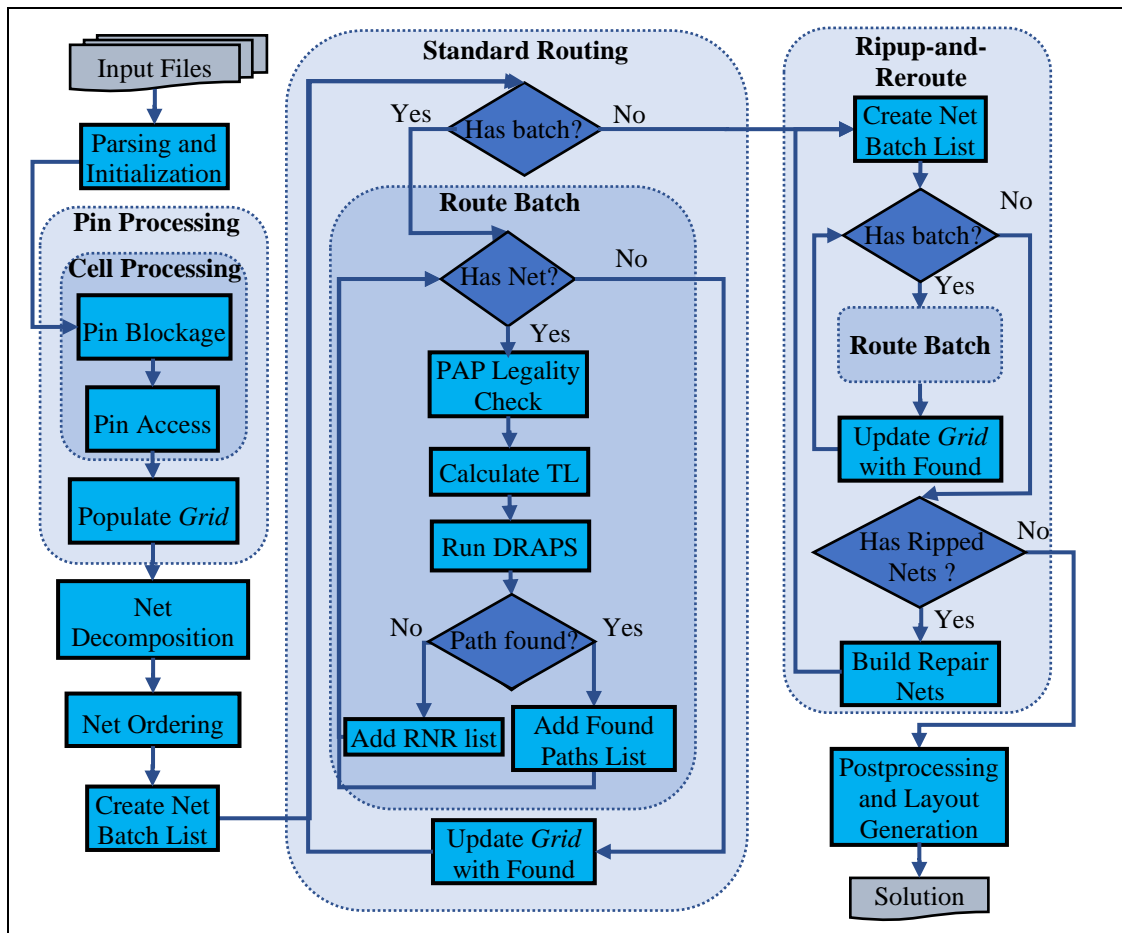


Figure 39- Flowchart of SmartDR detailed routing flow.

After the Net Decomposition and Guides Slicing step, the nets are sorted by the decreasing order of their half-perimeter wire-length (HPWL). The HPWL of a net is the half of the perimeter of the bounding box of the net pins. Routing longer nets first is better since they have more likelihood of being blocked than short nets. Since the tunnels of all nets have the same width, the net length factor weights more than the routing area factor. Also, longer nets have most of their routes in higher metal layers, using lower layers only for pin access, while shorter nets tend to use more the lower layers. This makes longer nets present few blockages to shorter nets, if they are routed first, while routing first shorter nets presents more blockages to longer nets.

SmartDR uses a parallel scheme, based on (CHEN, 2019a, 2019b), to route nets in batches. Before the Standard Routing step, batches of nets that do not present tunnel overlapping with each other are created. Thus, all nets within a batch can be routed in parallel later.

In Standard Routing step, each net batch is routed. The first step of the net routing is to perform the PAP legality check. The access points of the legal PAPs are passed as source and target points for the path search algorithm later. But before the path search is run, the tunnel of the current net is preprocessed in order to obtain a more realistic lowerbound for the heuristic function of the path search. This is addressed in section 6.3. Then, the path search is executed. If a path is found, it is added in a list of found paths. If a path could not be found, it is added in the Ripup-and-Reroute (RNR) list. After a net batch is routed, all paths found are added in the routing space, including the PAPs that were used by the path search to access the pins.

In the RNR step, the basic structure of the net batch routing methodology is the same as the Standard Routing step. Thus, these details are omitted in the flow chart, as they are inside the “Route Batch” box. In RNR, the path search algorithm is run in a mode where it is allowed to find a path that overlaps existing wire segments of different nets. These segments are removed from the routing space and are added in a list of ripped segments of their respective net.

After all batches are routed, if there are nets that were ripped, a net repairing procedure is started. For each net that has rips, a repair net is created, based on the connectors (i.e. wires and pins) of the ripped wire segments. These nets are decomposed in two pin nets, and their tunnels are obtained. The RNR algorithm may

also inflate the tunnels (i.e. increase and/or enlarge GRs) based on the ripping history. After this, the repair nets are added in the RNR list, and another RNR iteration begins.

When all nets are successfully routed, a postprocessing is applied in the solution, solving some DRVs. Then, the layout of the routing solution is created and saved in a def file, which is the output of SmartDR.

5 Pin Access

This section presents the proposed pin access approach. The scope of the pin access problem addressed in this work is regarding “hard pin access instances”, where pins have very irregular shapes and are often not aligned to the grid, requiring off-track pin access paths to perform some connections, as in (NIEBERG, 2011). Also, the proposed approach is intra-cell pin access, since it aims to provide short connections between the pins and the neighboring access points, handling the DRVs that commonly emerge from such connection.

The common approach to solve the pin access problem is to calculate a conflict-free solution (CFS) for all pin access paths of each pin of a cell. A CFS is a set of PAPs that may be simultaneously coexistent without causing any DRV. Figure 40a shows an illustration of a CFS. This procedure is performed before the routing step, and the resulting PAPs are implemented in the routing space, as in (NIEBERG, 2011). This implementation acts as a routing resource reservation that prevents wires to occupy the PAP locations. However, this strategy brings some issues. First, it is not feasible to implement a large amount of PAPs, mostly the ones that use vias, because they will form many routing blockages for other nets, and only one PAPs is actually needed. On the other hand, with a restricted number of PAPs, there are not many routing blockages, but there are few options for the path search to connect to the pin. This tends to make the path search harder, since it may spend more time finding an access point, or even prevent it to find a route at all, as in Figure 40b. Thus, in both approaches, routability is affected. A third approach is to implement few PAPs but let the path search chose other access points. The PAPs of these alternative access points are calculated on-the-fly or at the end of routing, in a postprocessing step. However, it may be impossible to find such PAPs without DRVs. Thus, the ideal scenario is to keep many DRV-free PAPs available to be chosen by the path search but without blocking many routing resources. To attend this, the current work proposes to use resource sharing ghost pin access paths with dynamic legalization and implementation.

A ghost PAP is a PAP that was created but was not implemented in the routing space. During the routing phase, they are checked for legality (i.e., whether they can be implemented without causing any DRV). If a PAP is legal and it is chosen by the path search to be a path terminal, it is dynamically implemented in the routing space. With this approach, a pin may have many DRV-free ghost PAPs that does not cause

any routing blockage. This gives the path search high flexibility to find an access point to the pins, increasing routability, and consequently, reducing RNR effort, which is a routing step with unfavorable runtime that may also degrade the routing quality. Another important feature of our method, is that the routing resources are shared for all PAPs from all pins. This means that, unlike the CFS methodology, the pin access solution accepts that a single grid point may be used by different PAPs of different pins. This is shown in Figure 40c. The decision of which PAP will utilize the grid point is performed on-the-fly. This also favors the flexibility of PAP selection, and, consequently, the routability. Thus, both features (ghost PAPs and resource sharing) are important. For example, in Figure 40b, even if we used a CFS with ghost PAPs it would be still impossible to connect to pin A, since both grid points on the right of A are statically assigned to PAPs of B (as seen in Figure 40a). With the proposed method, these points represent valid PAP candidates for both pins, which allows the connection to pin A. Thus, in this scenario, the path is found only thanks to the resource sharing feature. Since this approach requires dynamic PAP legality check, which has a runtime overhead, techniques that mitigate this overhead are also proposed in the next sections.

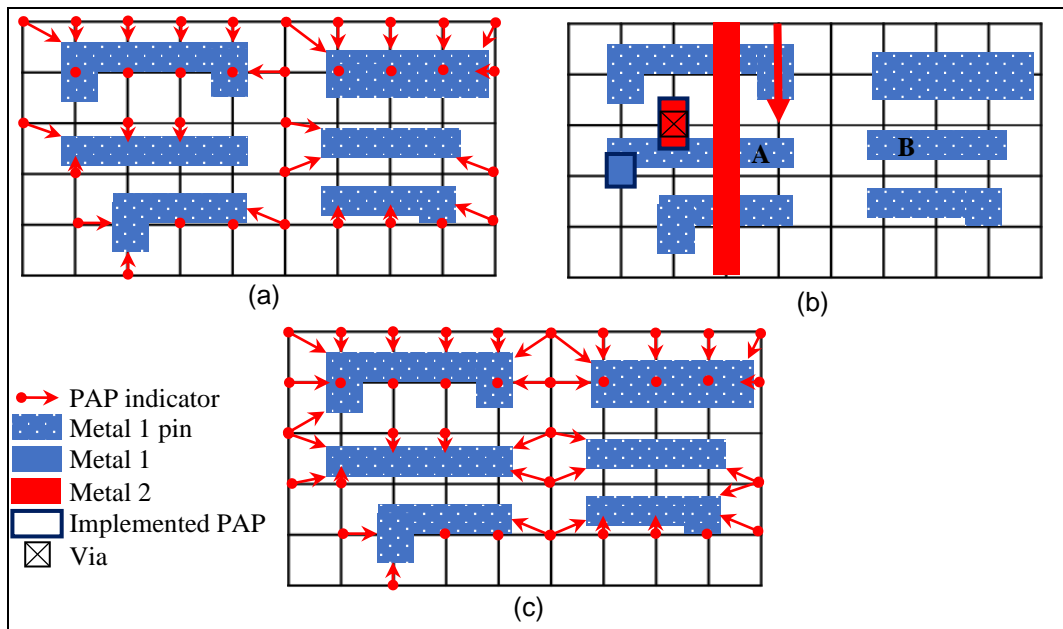


Figure 40: Illustration of pin access situations. (a) CFS. Some grid points, adjacent to pins, without PAPs are forbidden since using them will cause DRVs. (b) A path (red arrow) in metal 2 tries to connect to pin A, but cannot reach the implemented PAPs shown in (a). We are assuming metal 3 is unreachable either by wire blockages or absence of routing guide. (c) Pin access solution with resource sharing.

5.1 Calculating Pin Access Paths

5.1.1 Overall Procedure

The PAP creation procedure begins calculating root PAPs. These are the PAPs whose access points are in the vicinity of the pin, in the same layer. Thus, they are directly connected to the pin. Then, all root PAPs are expanded (as in maze routing), creating new PAPs, including in the upper metal layer, allowing PAPs to use access vias (these are called via-PAPs). There may be more than one expansion iteration, but it is commonly useless, since the paths of two or more expansions almost never present DRVs, and the main goal of the pin access calculation is to handle such DRVs.

A PAP that access the pin from an adjacent layer may do it by using many different vias from the via library. If we statically assign a specific via for a PAP, it is possible that, while in routing, this via causes a violation where other vias do not. This makes the PAP unavailable when it could be available if it used other via. Thus, a single PAP may keep many candidate access vias. The algorithm tries to create on-track vias. However, if it was not possible to create any PAPs with on-track vias for a given pin, an off-track via-PAP is created.

Although the proposed approach is to not implement the PAPs, it is pertinent to open exceptions in some cases. There are pins that present high potential to have their access routes easily blocked by wires of other nets (usually small pins). Since most of the pin access is performed by metal2, a via-PAP is implemented in such cases. After the calculation of the PAPs of all pins of a cellconfig, a procedure to identify which PAPs will be implemented is started. The procedure identifies these potentially blockable pins and tries to implement a via-PAP, in the cellconfig, avoiding DRVs with other via-PAPs already implemented. Via-PAPs that are more distant to the other pins receive high priority to be selected. For the ISPD18 testcases, this heuristic was enough to guarantee no conflicts between implemented via-PAPs. Anyway, in the case of conflicts, this can be solved by branch-and-bound techniques. The following heuristic to determine which pins are potentially blockable is used (any of these conditions must hold):

- If the pin has two or less via-PAPs
- If metal 2 preferred direction is vertical (horizontal) and the width (height) of the PAP bounding box is 0

- If metal 2 preferred direction is vertical (horizontal) and the pin have 4 or less PAPs and the PAP bounding box width (height) is one metal 1 pitch (in preferred direction)

The PAP bounding box is the bounding box that encloses all (on-grid) access points of all PAPs of the pin. The last steps of the pin access procedure are the conflict check of *same-pin* PAPs and the creation of LUTs with blockage information, as will be described ahead.

5.1.2 Design Rule Handling

Using the proposed approach, it is not necessary to control the violations between PAPs of different pins. When a PAP is calculated it also ignores possible violations with other PAPs of the same pin. However, these violations (of same-pin PAPs) are eventually detected, as will be explained later.

When calculating a PAP, it is allowed the occurrence of violations between the PAP and the pin, provided it may be solved using patch metals without causing any violation with other pins. The same-pin DRVs caused by the patch metal insertion can also be solved using patch metals the same way. Figure 41a shows an example of DRVs solved by patch metal insertion in Figure 41b. The patch metal usage gives more

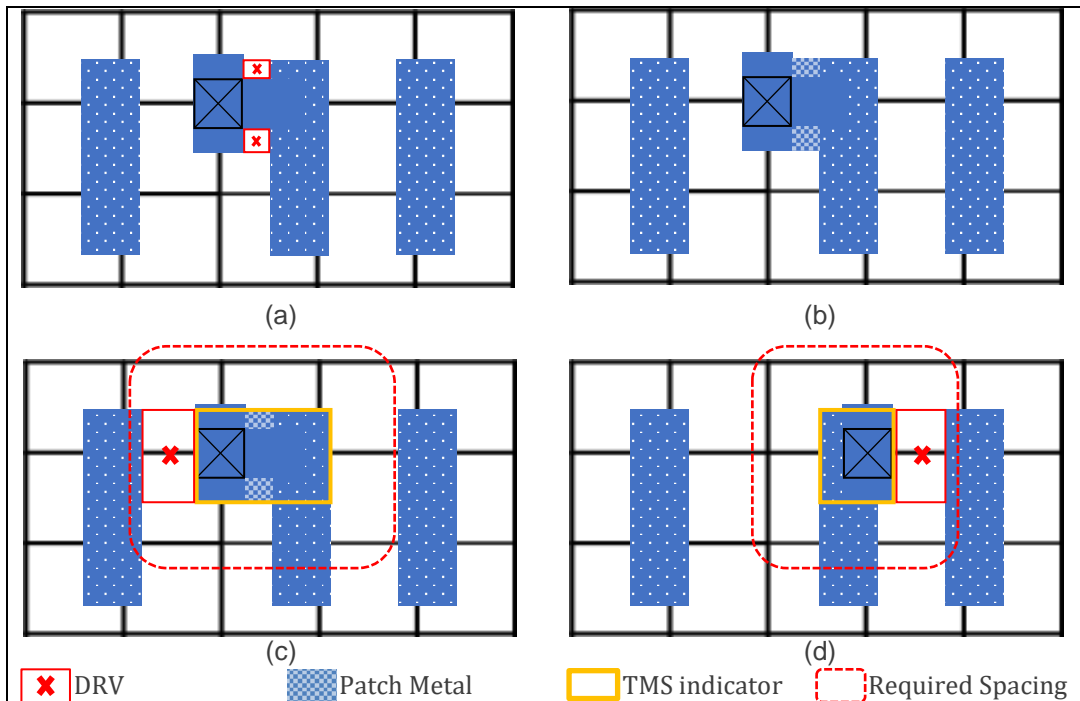


Figure 41: Illustration of patch metal insertion and TMS creation.

flexibility in the PAP creation, but may create thick metal shapes (TMSs), as shown in Figure 41c and d, that require a spacing higher than the minimum. In the figure's situation, the patch metal insertion solved two violations but created an unsolvable one. The proposed pin access algorithm is aware of TMS creation. In section 5.3, it is presented the proposed TMS detection algorithm. Regarding via-PAPs, the design rule checking with possible patch metal correction is performed with all possible vias in the via library. So, a via-PAP may have many configurations, one for each via. Only vias that cause unsolvable (by patch metals) violations are not present in the PAP.

During routing, when two or more PAPs of a same pin are implemented simultaneously, they may present DRVs between them, as shown in Figure 42a. In this case, we say they are conflicting PAPs. To allow high PAP diversity, it is allowed conflicting PAPs to exist if their violations can be solved using patch metals (Figure 42b). In this case, the conflict is said to be solvable. If the simultaneous implementation of two PAPs create DRVs that cannot be solvable using patch metals, as in Figure 42c, then the conflict is unsolvable and the PAPs cannot be simultaneously implemented.

Thus, the PAP legality check must be aware of DRVs of same-pin PAPs. However, it is expensive (in runtime terms) to dynamically detect same-pin PAP conflicts. This is mostly due to the fact that TMS detection is heavy. Therefore, during the pin access step and after all PAPs are created, same-pin PAP conflicts are precomputed and stored in conflict graphs, one for each pin. The graph vertices are the PAPs and the edges are their conflicts. Solvable conflicts store the patch metals and their related blockage information, as will be explained in the next section.

This procedure iterates over all pair combinations of PAPs of a given pin, checking: (1) if their simultaneous implementation requires patch metals; (2) if the

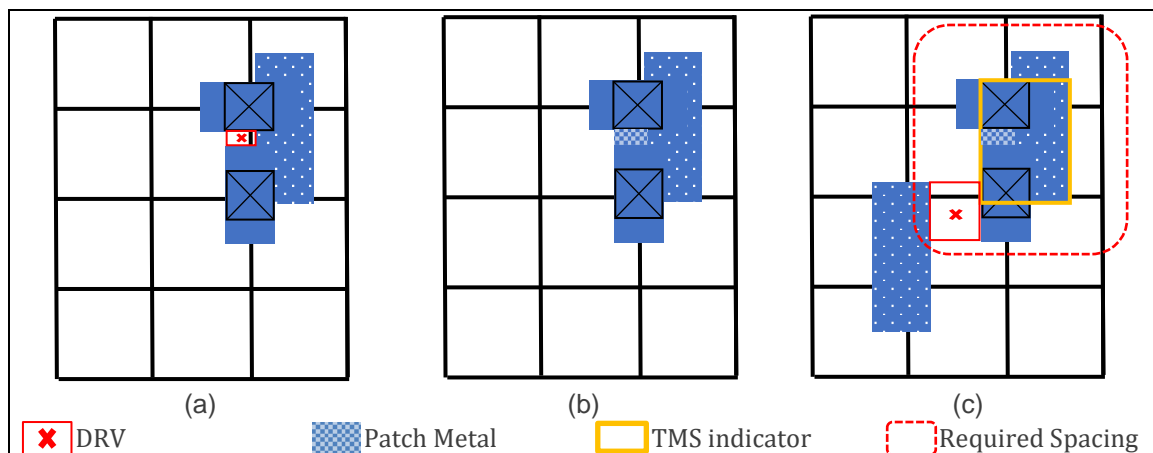


Figure 42: Illustration of same-pin PAP conflicts.

patch metals directly cause a violation with nearby pins; and (3) if the patch metals create TMSs and cause violations with other pins. For this verification, PAPs with multiple vias are decomposed into many PAPs each one with a single via. To optimize runtime, this conflict verification is trimmed by considering only the PAPs that are close enough to create and DRVs.

5.2 Dynamic Manipulation

In the pin access procedure, after the creation of the LUTs containing the PAP blockage information, LUTs are created containing queries (i.e. line segments) in the regions where the obstacles should be verified and brought to the cache. These are the regions that will be queried by the PAP blockage information in order to check PAP legality. These queries should also be optimized to minimize the accesses in the *grid*. Figure 43b shows the cache queries of a pin. Although Figure 43a shows only one PAP, the cache queries in Figure 43b are considering all PAPs of the pin. Figure 43c illustrates metal 1 blockages around the pin, during routing step. Figure 43d shows the resulting contents of the cache after using the queries of (b) in the scenario of (c).

The first step in the PAP legality check of a pin is to create the cache. In order to know whether a PAP is legal or not, it is necessary to look at the same-pin PAP conflict graph. If there is an implemented PAP that has an unsolvable conflict with the candidate PAP, then the candidate PAP is illegal. Otherwise, the cache is queried using the LUT holding the blockage information of the PAP and their same-pin conflicts

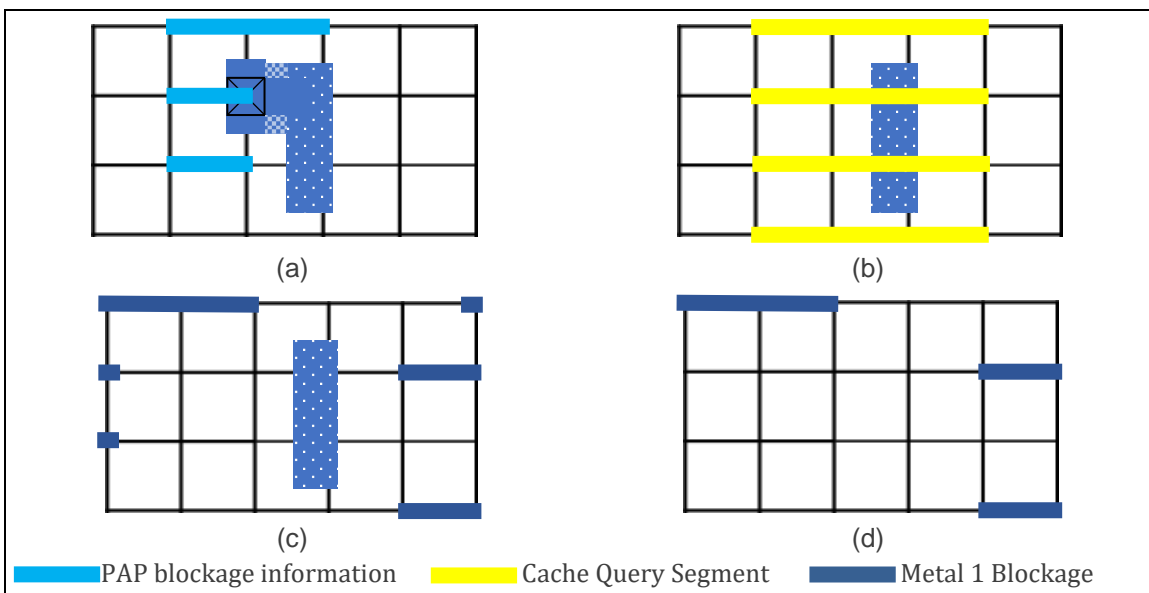


Figure 43: Illustration of the PAP blockage information and cache usage in metal 1.

(if existent). If the blockage information is not intersected by routing obstacles in the cache, then the PAP is legal. Otherwise, it is illegal. For via-PAPs, the legality check is performed iteratively, for each via, until the PAP is considered legal. When the path search successfully finds a path, the PAPs used as the terminals of the path are implemented. This implementation consists in adding in the routing space all PAP blockage information, which avoids other routing objects (wires, vias and PAPs) to use the same routing resources.

In RNR, PAPs may be directly ripped or may be ripped when its wire connector is ripped. In such cases, they are deactivated, i.e., all blockage information is removed from the routing space. When performing PAP legality check in RNR, a PAP is also considered legal when it intersects obstacles that are susceptible to be ripped, such as wires, vias and other PAPs (except those that had its implementation fixed in cell processing phase). However, the path search penalizes the search nodes of these PAPs the same way it penalizes nodes that overlap obstacles.

One last advantage of the proposed approach is that it naturally handles the problem of overlapping PAPs of neighboring cells. Pins very close to the cell boundary may be too close to the pins of a neighbor cell, such that its PAPs overlap each other. This is a problem if we use the static PAP implementation methodology with CFS, as in (NIEBERG, 2011). Since it is not always possible to implement these PAPs in a safe region, due to neighboring pins of the same cell, they cannot be statically implemented, or they are implemented but disallow the implementation of conflicting PAPs of the neighbor cell. This introduces the possibility of a pin not having available PAPs. In these cases, (NIEBERG, 2011) calculates these PAPs on-the-fly, but there is no guarantee there will be a DRV-free PAP. In the proposed approach, we can afford to allow the creation of PAPs in the risk zone, or even out of the cell borders, since they will be eventually verified for availability during routing. Also, the high PAP diversity and the possibility to deactivate them guarantee that there will be always DRV-free available PAPs.

5.3 Thick Metal Shape Detection

The TMS detection of two or less rectangles is trivial, but in the current problem, there may be many rectangles to analyze. Thus, the TMS detection method must be robust. This section proposes a fast algorithm to perform this procedure. Note that this algorithm may be used regardless of the pin access approach, but it is most important

in the proposed one, since it is widely used. This high usage is due to the fact that the method always tries to solve PAP-pin DRVs with patch metals, rather than forbidding the PAP creation, and using patch metals increases the likelihood of TMS creation.

A TMS is a rectangle whose lesser side is large enough to require a spacing higher than the minimum spacing of the spacing table. We define the TMS Detection Problem as follows:

Definition 1: *given a polygon R , defined by a set of rectangles, the TMS Detection Problem aims to find all possible TMSs, with maximum dimensions, that are completely covered by R .*

The proposed method uses a border projection approach to solve the problem. Consider Figure 44. The first step is to obtain the external borders of the shape resulted from all rectangles in R . Then, each border is maximized (c) and projected towards the shape interior (d). The projection ends when the border touches an opposite border. Then, the rectangle resulted from both borders (projector and projected), called TMS candidate, is checked whether it is a TMS or not. The projected border is subtracted from all touched borders, and the remaining pieces of the projected border continue the projection, and so on.

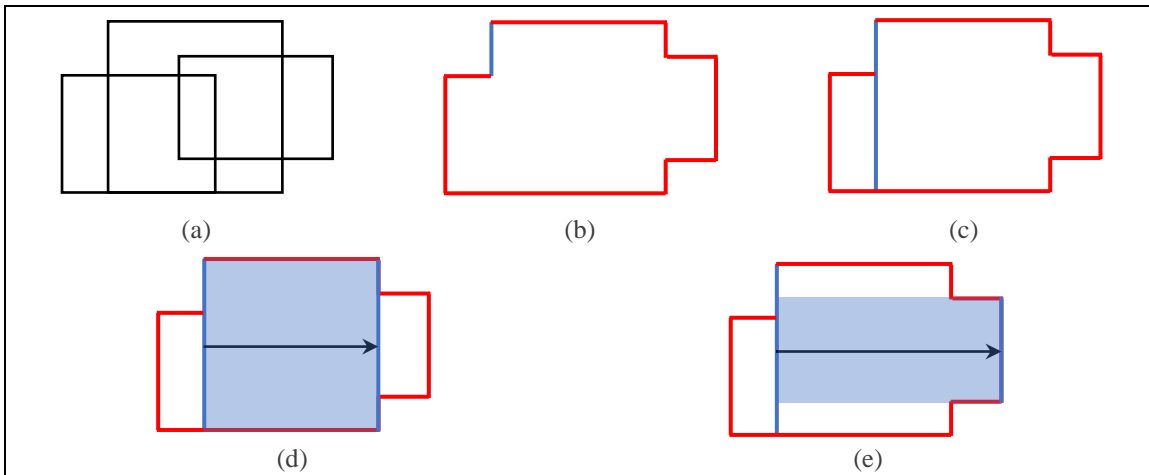


Figure 44: Illustration of our TMS detection algorithm. (a) R set. (b) external borders of R ; the blue border will be projected. (c) the border is maximized. (d) the border is projected and a TMS candidate is obtained (transparent rectangle). (e) the remaining piece of the projected border continues the projection.

The TMS detection algorithm is used in three cases: 1) On the pin shapes; 2) On PAP creation; 3) On same-pin PAP conflict detection. The case 1 occurs in the initialization procedures of the detailed router. In case 2, it is not necessary to reconstruct the pin borders, since they were already calculated. Only the PAP borders need to be created and added in the pin borders set. The PAP borders are removed from this set, after the creation of the PAP. Also, the pin borders do not need to be projected again, since they would possibly find the same TMSs again. In case 3, since the PAP borders were already calculated, they are also reused. Therefore, in cases 2 and 3, the TMS detection problem is slightly changed. We now want to find only the TMSs that are created by adding a set of rectangles A on R . In our application, R is the pin shape and A is the PAP shapes, in cases 2 and 3. In case 3, A also holds the patch metals used to solve the violation between the PAPs. This variation of the first problem is defined as the Differential TMS Detection problem, as follows:

Definition 2: *considering Definition 1, find the TMSs of $A \cup R$ that are not in R .*

A trivial method to solve this problem is to calculate the TMSs of $A \cup R$ and obtain the difference of the TMS solution of R , but this is not efficient. Thus, an algorithm to efficiently solve this problem is proposed. The key idea is to project only the borders of A . The algorithm is presented in Figure 45. If set $A = \emptyset$, then the algorithm solves the first version of the problem. The *BorderMap* B is a data structure that holds the borders of the polygon R . It is composed of two maps, one for vertical and other for horizontal borders. Each map maps lines to sets of borders of their respective line. These lines are grouped in a sorted list, so that the border projection algorithm can use a sweep line approach. The *createBorders* function obtains all borders of the rectangles of the first parameter and subtracts them from the rectangles of the second parameter to avoid the existence of borders inside the resulting shape (i.e. $A \cup R$). Obviously, the borders of a given rectangle are not subtracted from itself. The *updateBorderMap* function adds a set of borders in the border map. The *getBorders* function returns a copy of the borders of the border map. The *copy* function returns a copy of the border in the parameter. If a border is being projected for the first time (lines 11 and 12), it is extended, as shown in Figure 44c. The *TMS-Candidate* function checks whether the border meets the minimum length that the lesser side of a rectangle must meet for it to be considered a TMS. Thus, line 13 is an optimization

step that avoids the projection of borders that are guaranteed to not create TMSs. Line 14 performs a border projection illustrated in Figure 44d, using a sweep line approach. The function *rect* creates a minimum rectangle enclosing the border *b*, and extending itself to the line of *o*. It is actually creating the TMS candidate. The algorithm may find redundant TMSs. When a TMS is found, the list of TMSs found so far (*TMS-List*) is checked to avoid redundant TMS insertion (line 15). It is possible to solve the redundant projection issue by removing the intersections of the projected border with its intersecting borders, from the borders in *P*. However, this was not worth the effort in our implementation. In line 17, the *erase* function subtracts *b* from the borders intersected by *b*. In lines 18-20, the resulting borders continue the projection, as illustrated in Figure 44e. The *intervalIntersectionLength* function performs an interval intersection between the two borders (i.e. it considers only the border tips) and returns the length of the intersection interval. If the intersection length is greater than 0, this means that, if *b₂* creates a TMS *t* in a future projection, then *t* depends on *o* to exist. Otherwise, all TMSs created by *b₂* have their existence independent of *o*. Since *o* is a border of *A*, these TMSs already belonged to *R*, and do not need to be considered.

ALGORITHM: TMS-Detector(Set *A*, Set *R*, BorderMap *B*)

```

1  if A ≠ ∅:
2      P ← createBorders(A, A ∪ R)
3      updateBorderMap(B, P)
4  else:
5      P ← getBorders(B)
6  for each border b in P:
7      projectBorder(b, copy(b))
8  return TMS-List
9
10 function projectBorder(Border b, Border o):
11     if b = o:
12         Extend b until it reaches the shape limits
13     if not TMS-Candidate(b): return
14     Project b towards its inner direction until it intersects a parallel
        opposite border
15     if rect(b, o) is a TMS and TMS-List do not contain rect(b, o):
16         TMS-List.add(rect(b, o))
17     pieces ← erase(b, intersectedBorders(b))
18     for each border b2 in pieces:
19         if A = ∅ or intervalIntersectionLength(b2, o) > 0:
20             projectBorder(b2, o)

```

Figure 45- Pseudocode of the proposed TMS detection algorithm.

5.4 Comparison with Related Work

The main goal of section is to clarify the differences between the proposed pin access method and the work in (NIEBERG, 2011), since it is the only work that addresses pin access within the scope of the proposed method, which is intra cell pin access allowing off-track paths. The other methods presented in section 3.3 are inter-cell, and their intra-cell handling is more straightforward, since off-track paths are not allowed (at least in the SAMP methods), and it is guaranteed that the connection to the pins can be done without DRVs by simply putting a via over the pin. Table 1 shows the differences between the proposed method and (NIEBERG, 2011).

Table 1: Differences Between the Proposed Method and (NIEBERG, 2011).

Method	CFS	RS	PAP implementation	Patch metals
(NIEBERG, 2011)	yes	no	static, with exceptions	no
Proposed	no	yes	dynamic, with exceptions	yes

CFS vs Resource Sharing (RS): in (NIEBERG, 2011), each *cellconfig* (referred as *circuitclassess* in (NIEBERG, 2011)) has a conflict-free solution for all pins, meaning that all PAPs may coexist without causing any DRVs. In the proposed approach, the *cellconfigs* present a resource sharing solution, meaning that the PAPs of different pins would present DRVs between them, if implemented simultaneously, and may use the same routing resources (grid points).

PAP Implementation: in (NIEBERG, 2011), before routing, all PAPs are implemented in the routing space. There may be conflicts between PAPs of neighboring cells, and in these cases some PAPs may not be implemented. If this makes a pin to not have any implemented PAP, then a PAP is constructed on-the-fly during the routing step. In the proposed approach, the PAPs are not implemented before routing, except for the via-PAPs of small pins (at most, one via-PAP per pin is implemented). During routing, the other PAPs are dynamically checked for legality and are dynamically implemented, if chosen by the path search. LUTs and caching strategies are used to mitigate runtime overhead of legality check. Also, an implemented PAP may be deactivated in RNR.

Patch Metal Usage: in (NIEBERG, 2011), no patch metals are used. The PAPs may perform many detours to avoid DRVs with the pin and within the path itself, and the path may not be the shortest. The proposed method heavily relies on patch metals

to allow PAP diversity. The PAPs are always the shortest possible. The patch metals greatly contribute to TMS creation, and a fast TMS detection algorithm is used.

Although the proposed method is intra-cell, it is not incompatible to an inter-cell point of view. An inter-cell extension of the method would require one more step, to create wire segments connecting the via-PAPs and to direct them to other net components. As for the compatibility of the proposed approach with other pin access contexts, with simplified intra-cell handling, the use of patch metals and TMS detection would not fit, but the resource sharing idea can be used independent of the pin access context.

5.5 Experiments

This section evaluates the effectiveness of the proposed pin access approach. For this, the ISPD 2018 Contest (MANTIK, 2018) benchmark suite is used, which is derived from industrial test cases. Table 2 presents the benchmark information. The columns represent the number of standard cells, number of macro blocks, number of nets, number of I/O pins, number of metal layers, die size (in mm²) and technology, respectively. In order to obtain design rule violation results and other routing metrics, Cadence Innovus 17.1 and the ISPD18 Evaluation Script are used. The experiments were run in a Linux machine with 132Gb RAM, CPU AMD Opteron 2.3 Ghz. The proposed detailed routing system was implemented in Java. This experimental setting was the same used in all other experiments in this work (sections 6.6 and 7).

First, we will evaluate the effectiveness of the proposed RS approach over the CFS one. Note that the goal is not to compare the proposed pin access algorithm to any specific algorithm, such as (NIEBERG, 2011). The goal here is to compare *approaches*. In theory, the CFS approach either blocks many routing resources, if implementing many PAPs, or presents few PAPs per pin. In either case, routability and runtime is affected. In theory, the RS approach solves these issues, but there is a runtime overhead for PAP legality check. Thus, the objective of this experiment is to evaluate and measure these predictions in practice.

In order to do this, a pin access solution resulted from the proposed pin access method is used to create a CFS, and two versions of SmartDR, with CFS and RS, are compared. The algorithm to extract a CFS out of the RS solution is presented in Figure 46. All PAPs in metal 1 that do not present conflicts with other PAPs, including within the same pin, are implemented, except by those in adjacent grid points to other PAPs

Table 2: Benchmark Information

Bench	#std	#blk	#net	#pin	#layer	Die Size	Tech	Description
test1	8879	0	3153	0	9	0.20x0.19	45nm	Standard cell netlist only.
test2	35913	0	36834	1211	9	0.65x0.57	45nm	Standard cell netlist with IO pins.
test3	35973	4	36700	1211	9	0.99x0.70	45nm	Standard cell netlist with IO pins and block macros.
test4	72090	4	72410	1211	9	0.89x0.61	32nm	Design has Metal2 OBS in some of its standard cells.
test5	71946	8	72394	1211	9	0.93x0.92	32nm	Design has Metal2 OBS, Metal2 Power/Ground pins, and routing direction is reversed.
test6	107919	0	107701	1211	9	0.86x0.53	32nm	Design has Metal2 OBS, Metal2 Power/Ground pins, and reversed routing direction, but without any block macro.
test7	179865	16	179863	1211	9	1.36x1.33	32nm	Quad-core design with Metal2 OBS and Metal2 Power/Ground pins as blockage.
test8	191987	16	179863	1211	9	1.36x1.33	32nm	Quad-core design with Metal2 to Metal3 OBS and Metal2 to Metal4 Power/Ground pins as blockage.
test9	192911	0	178858	1211	9	0.91x0.78	32nm	Quad-core design with Metal2 to Metal3 OBS and Metal2 to Metal4 Power/Ground pins as blockage, no block macro, higher utilization.
test10	290386	0	182000	1211	9	0.91x0.87	32nm	Quad-core design with Metal2 to Metal3 OBS and Metal2 to Metal4 Power/Ground pins as blockage, no block macro, extra congested area.

of the same pin. In metal 2, only one via-PAP was implemented by pin, also without conflicts with any other PAP of any pin. The best number of implemented PAPs was also evaluated, as discussed ahead, and the mentioned configuration was found to be the best. The algorithm iteratively implements one PAP per pin, until it has no PAPs to implement. It gives priority to pins with less PAPs, and to PAPs that are further from the other pins. In all testcases, all pins had some implemented PAP. In the pseudocode of Figure 46, A is an array of priority queues of PAPs. Each queue holds the PAPs, of a given pin, that are available to be implemented. I is the set of implemented PAPs. P

ALGORITHM: CFS(Resource Sharing Solution S)

```

1  Build PAP queue array  $A$  from  $S$ 
2   $I \leftarrow \emptyset$ 
3  do:
4    Build pin queue  $P$  considering the available PAPs in  $A$ 
5    if  $P$  is empty: break
6    while  $P$  is not empty:
7       $pin \leftarrow pop(P)$ 
8      while  $A[pin]$  is not empty:
9         $pap \leftarrow pop(A[pin])$ 
10       if  $canBeImplemented(pap, I)$ : break
11        $pap \leftarrow null$ 
12       if  $pap \neq null$ :
13          $I \leftarrow I \cup pap$ 
14   while true
15   return  $I$ 

```

Figure 46- Pseudocode of the method to create a CFS from the RS solution of SmartDR.

is the priority queue that stores all pins. In line 4, if a pin has 0 available PAPs, it is not added in P . The function *canBeImplemented* verifies the implementing restrictions mentioned earlier.

Table 3 presents the results. In order to provide a more accurate measure of the runtimes, they are presented considering a single-thread execution. However, in order to present a realistic total runtime difference of the compared approaches, the column “Total Real Time” considers 8 threads, as it is the default in ISPD18 Contest, and means the total time spent on the entire routing flow. “LC” is the total time spent on PAP legality check. “Total Time RS 1t” is the total time of the routing flow using 1 thread. “Path Search Time” is the time spent only in the path search. “Fails” is the number of failed path searches in the Standard Routing step. “Grid Init. Time” is the time spent on transcribing all cellconfig information to the grid, before routing. “RS” refers to the proposed Resource Sharing approach. All times are measured in seconds. “Red. Total” is the total runtime reduction by using RS (i.e. $(CFS-RS)/CFS\%$). “Red. PS” is the path search runtime reduction of RS. “Fail Inc.” is the increase in “Fails” of CFS.

The execution using CFS is implementing one via-PAP per pin. Implementing up to 2 via-PAPs has no significant impact on runtime, as it slightly increases or decreases, but it increases the number of fails. However, implementing up to 3 via-PAPs has considerable impact in runtime (17% in average, w.r.t. 1 via-PAP). This is shown in Figure 47. Also, implementing all possible via-PAPs has caused near 3000

Table 3: Comparison Between RS and CFS Approaches

Benc.	LC	Total Time		Total Real Time		Path Search Time		Fails		Grid Init. Time		Red. Total	Red. PS	Fail Inc.	Red. WL	Red. Vias
		RS	1t	RS	CFS	RS	CFS	RS	CFS	RS	CFS	(%)	(%)	(%)	(%)	(%)
test1	2.7	24		16	21	9	14	38	341	0.4	1.6	24	36	797	6.4	5.2
test2	25	164		71	92	82	122	163	1101	1.9	7	23	33	575	3.3	6.0
test3	27	255		157	217	166	269	525	1865	2.1	7.4	28	38	255	3.1	6.9
test4	75	609		314	355	256	463	1214	5263	9.8	27	12	45	334	3.3	-2.9
test5	64	562		208	303	232	446	10167	14288	16	32	31	48	41	3.0	-2.1
test6	98	802		286	329	340	649	17148	24289	27	51	13	48	42	3.1	-2.0
test7	164	1199		429	502	571	990	24590	36145	47	88	15	42	47	2.9	-2.1
test8	171	1206		435	664	565	998	24255	36263	45	89	34	43	50	3.2	-2.9
test9	180	1160		405	647	557	987	28439	40473	44	91	37	44	42	3.8	-2.9
test10	189	2135		717	1041	1482	2183	31763	46450	52	94	31	32	46	3.3	-2.3
Avg												24.8	40.8	222.9	3.5	0.1

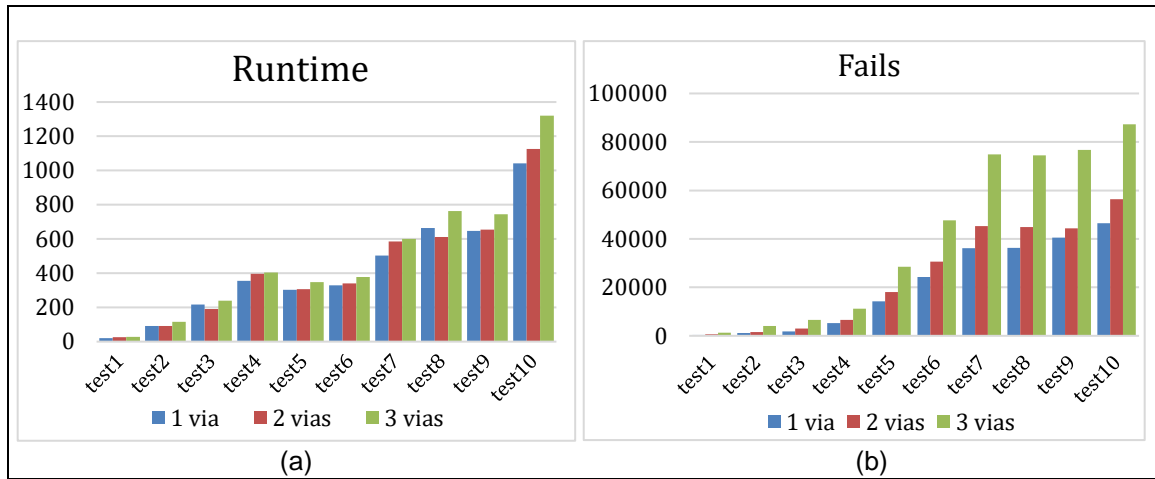


Figure 47: Comparison of the CFS implementing 1, 2 and 3 via-PAPs, regarding total routing runtime (a) and number of failed searches (b) in Standard Routing step.

fails in test1, which is the easiest test case, and an unfeasible runtime. This only shows what was already expected: implementing too many via-PAPs causes more routing blockages than choices to the path search access the pins. Regarding the implemented PAPs in metal 1, there is not much space to implement many such PAPs, due to the neighboring pins and their PAPs. Thus, this does not have considerable impact in routability. Still, two criteria were tested: in the first, all PAPs without conflicts with other PAPs of the same pin were implemented; the second criterion added another restriction to the first, which was to forbid the implementation of PAPs in adjacent points. Thus, the second criterion provides a sparser solution. Both criteria were evaluated and it was observed a very slight runtime and fails improvement with the second criterion. In average, the runtime reduction was 0.3% and the fails reduction was of 3.4%.

The PAP legality check takes, in average, takes 12% of the total runtime of SmartDR. The CFS approach presents higher Grid Init. Time, since it has to implement the PAPs. This time increase compensates part of the PAP legality check overhead. As expected, CFS presents more fails than RS. The average fail increase is 321% (589%, for testcases 1-4, and 144% for the others). The runtime overhead resulted from this fail increase is reflected mainly in the path search, since it spends more time to find a PAP or to realize that there is no route to any PAPs. The average runtime increase in the path search is 40%. This shows the high runtime impact in the path search that the lack of pin access flexibility may cause. The average total real time increase is 24.8%.

The WL and via count of the solutions of CFS and RS were also measured. In Table 3, “Red. WL” is the WL reduction when using the proposed method, and “Red. Vias” is the reduction in the via count. The large increase in the number of fails contributes to the routing quality degradation. With more fails, we have more ripups in RNR and this makes WL to increase. The average WL reduction of using RS is of 3.5%. Regarding via count, it is possible to occur an increase and a decrease. With more available via-PAPs, this contributes to an increase, but with less fails, and consequently less ripups and detours, this contributes to a decrease. For the 45nm testcases, the RS approach brought a via reduction of 6% in average. For the other benchmarks, there was an average increase of 2.4%. Also, the CFS using 1 via-PAP presented better results, regarding WL and via count, than using 2 or more via-PAPs.

Regarding the via count oscillation in different technologies, this can be explained by the following. DRAPS tries to minimize the number of via-PAPs used, as will be discussed in section 6.2. This is done by assigning costs to via-PAPs, and to assign cost 0 to a via-PAP already implemented. As will be also shown in the experiments of section 6.6.1, DRAPS manages to reduce the number of vias more in the 45nm benchmarks, with 11.1% in average, than in the 32nm ones, with 6.3% in average. The difference is probably due to the fact that in the 45nm benchmarks the routing over the pins is sparser, and thus, due to the lack of obstacles, the algorithm is freer to reuse the same vias, in different path searches. A possible solution to decrease the use of via-PAPs in the 32nm benchmarks is to increase the via cost in the path search, so that it makes more effort the reuse the via-PAPs already implemented.

In addition, an evaluation of the impact of implementing one via-PAP, in the proposed approach, was also performed. Figure 48 shows results in three scenarios:

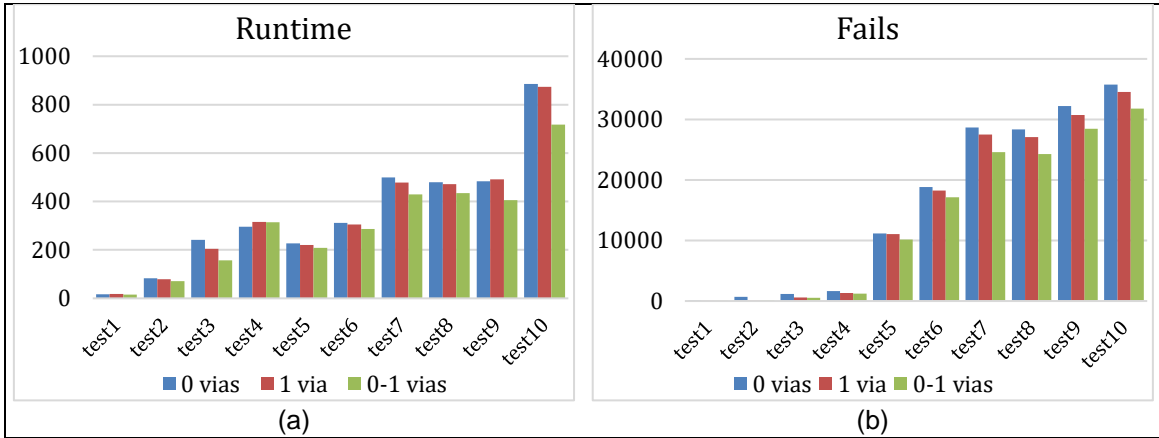


Figure 48: Comparison of different strategies of via-PAP implementation, and results in total routing runtime (a) and failed searches (b) in Standard Routing step.

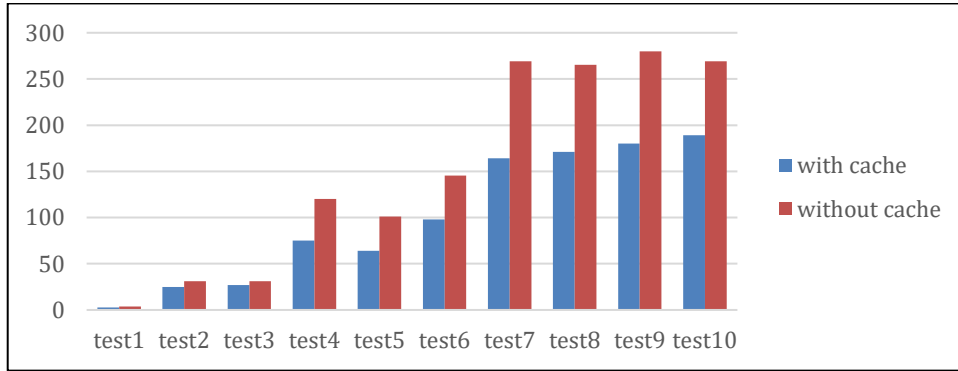


Figure 49: Runtime results of PAP legality check with and without cache.

(1) implementing no via-PAP; (2) implementing 1 via-PAP per pin; (3) using the heuristic described in section 5.1.1 to identify potentially blockable pins and implementing one via-PAP in these pins. Implementing no via-PAPs causes considerable increase in fails and runtime, w.r.t. the default approach (0-1 vias). In average, it presents 70% more fails and 16% more runtime. Implementing 1 via-PAP per pin brings, in average, 9.6% more fails and 13% more runtime.

Figure 49 shows the runtime results of the PAP legality check with and without using a cache to store the grid blockages before the legality check itself. The runtime with cache includes the time spent on building the cache. The average runtime reduction of using a cache is 30%.

Regarding TMS detection, it was evaluated the runtime of the procedure and the impact in spacing violations when not using TMS detection. Also, it was measured the gain in runtime of using the proposed TMS algorithm considering the Differential TMS Problem (Definition 2) against the same algorithm but solving the first version of the problem (Definition 1). Basically, in Definition 2, all PAP borders that were already constructed are reused and the algorithm only tries to detect TMSs created by the

Table 4: TMS Usage Results

bench	Cellcon	PA2 Time	TMS2 Time	TMS1 Time	PA1 Time	TMS2/PA2 (%)	TMS1/PA1 (%)	TMS2 Red. (%)	#spacing	
									TMS	No TMS
test1	196	4.9	0.8	2.4	6.8	16.3	35	67	1	40
test2	222	6.2	1.15	3.2	8.2	18.5	39.4	64	6	936
test3	247	6.2	1.05	2.8	7.5	16.9	38.0	63	33	996
test4	2764	111	21.6	87.1	167	19.5	51.9	75	204	6120
test5	2768	83	19.4	66.6	123	23.4	53.8	71	75	4566
test6	2886	93	19.4	72.6	143	20.9	50.5	73	228	6992
test7	228	5.1	0.74	1.6	5.6	14.6	28.3	53	300	13143
test8	230	5.1	0.80	2.1	6.0	15.7	35	62	351	13141
test9	136	4.5	0.84	2.1	5.5	18.6	37.7	60	238	13383
test10	144	4.7	0.67	1.3	5.1	14.3	26.4	50	1398	14793
Avg						17.8	39.6	63.8		

addition of the PAPs in the pin shape, while in Definition 1 it always constructs the borders of the resulting shape and tries to detect all TMSs.

Table 4 presents the experimental results regarding TMS detection. The “Cellcon” column represents the number of cellconfigs. “PA2 Time” is the time spent on the router Cell Processing step (basically the pin access calculation time), when using the TMS algorithm to solve the problem of Definition 2 (here we refer to it as TMS2). “TMS2 Time” is the time spent on TMS2 algorithm. “TMS1 Time” is the time of the TMS procedure of solving the TMS problem of Definition 1. “PA1 Time” is the Cell Processing time using TMS1. “TMS2/PA2” is the proportion of “TMS2 Time” w.r.t. “PA2 Time”. “TMS1/PA1” is the proportion of “TMS1 Time” w.r.t. “PA1 Time”. “TMS2 Red.” is the runtime reduction of “TMS2 Time” w.r.t. “TMS1 Time”. “#spacing” is the number of spacing violations using TMS detection (“TMS” column) and not using it (“No TMS” column). Note that the results, in DRVs, of both approaches (TMS1 and TMS2) are the same.

The TMS2 procedure takes, in average, 17% of the pin access process time (“PA2 Time2”). Considering only tests 4-6, which present higher number of cellconfigs, the procedure takes, in average, 21% of the total process. The TMS1 procedure presents a high runtime overhead w.r.t. TMS2. Using TMS1, the procedure time takes 39% of the total cell processing time, in average. This arises to 52%, when considering tests 4-6. In average, using TMS2 reduces the runtime in 63% w.r.t. TMS1.

The non-use of TMS detection causes a huge increase in the number of spacing violations (4958%, in average). This shows the importance of TMS detection in the proposed pin access method. Since patch metals are frequently used to solve DRVs, TMS creation also becomes more frequent.

5.6 Conclusions and Future Works

Section 5 presented the proposed pin access method, which is a novel approach to solve the pin access problem. Unlike any other pin access method in literature, the proposed approach allows pin access paths of different pins to share the same routing resources. The proposed approach is also unique regarding the patch metal usage and the TMS detection, allowing high PAP diversity without suffering from DRVs caused by patch metal usage and TMS creation, and this is thanks to the TMS detection algorithm proposed.

The impact of the high flexibility in PAP selection brought by the proposed method was measured by comparing it to a CFS approach. The experiments showed that it considerably reduces runtime and number of failed path searches. It also provides a slight improvement on WL and may also do the same with the via count, although the via count may be increased. The TMS detection was shown to be essential in design rule handling in the proposed pin access method, as the proposed TMS detection algorithm was shown to be efficient.

As future works, it is intended to do the following researches. The PAP legality check may be exchanged by making each insertion of routing objects in the search space to be aware of the blockage information of the PAPs, such that it disables any intersecting PAP. This implies in the necessity of an efficient control of the available PAPs. It is also pertinent to evaluate the impact of varying the number of the created PAPs, since it is possible that a little less PAPs may present a better tradeoff, regarding PAP selection flexibility (and, consequently routability and runtime) and the dynamic PAP control overhead. The patch metal usage should be evaluated in order to identify the extension of its beneficial effects. It is also necessary to evaluate to what extent it is worth to allow TMS creation. Regarding reserving routing resources to PAPs, it is intended to evaluate the effectiveness of applying cost penalties in the grid, in the PAPs, to discourage the path search of using these points but without disallowing it.

6 Path Search

This section presents the proposed techniques related to the path search. More specifically, it is presented the proposed design rule aware path search algorithm (DRAPS), which is an A*-interval-based path search, and a method to improve the lowerbounds of the h function of A* in the detailed routing scenario. The path search mechanics of DRAPS is based on the path search algorithm in BonnRoute (GESTER, 2013), which is based on Hetzel's algorithm (HETZEL, 1998). The contributions of the proposed techniques are (1) adaptations in the path search to properly handle PAP costs in the source/target points, presented in section 6.2, and (2) the integration of design rule checking (DRC) into the path search core, presented in section 6.4. The proposed method to improve the h function is presented in section 6.3. Section 6.1 presents the basic path search mechanics and an overview of the algorithm.

6.1 Path Search Mechanics

The routing space may be defined by a grid graph. This graph does not need to be explicitly stored in a graph data structure though, as it may be implicitly represented by the layer pitch information and the routing obstacles. The proposed algorithm is based on the A* search (HART, 1968). Each node n has an associated cost, $f(n) = g(n) + h(n)$, where $g(n)$ is the best-known cost from a source point to n , and $h(n)$ is an estimated cost from n to a target point. In the classical A* approach, each search node is associated with a grid graph vertex. In DRAPS, each search node is composed of a set (interval) of grid graph vertices on the same track. Rows of vertices with the same $f(n)$ cost (calculated as if they were search nodes in a maze search) are merged in one interval of vertices, which is labeled by $f(n)$ and consists in a search node. Due to the knowledge of the implicit geometric information of the grid graph and the h function behavior, it is possible to efficiently calculate the intervals in constant time. This interval labeling technique is the same proposed by Hetzel, discussed in section 3.2.4. If the h function is implemented by the manhattan distance, then the intervals are directed to the target node (as in Figure 27). DRAPS uses a more complex h function though, that is aware of the global routing guides, and is explained in section 6.3.

Figure 50 presents the pseudocode of the proposed algorithm. The algorithm handles multi source-target points. As in multi source-target A* search, each node n seeks a target point $t \in T$ that minimizes $f(n)$. Each search node has many associated information: an interval of grid points, a reference point for the interval calculation, g

ALGORITHM: DRAPS(Source S , Target T)

```

1  Initialize open set  $Q$  with source set  $S$ 
2  while  $Q \neq \emptyset$ :
3       $n \leftarrow pop(Q)$ 
4      if  $f(n) \geq f(B)$ : return  $B$ 
5      if  $n.i$  intersects  $n.target$ : return  $n$ 
6      Find a target point  $t$  intersecting  $n.i$  with lowest  $f((n, t))$ 
7      if  $t \neq \text{null}$  AND ( $B = \text{null}$  OR  $f((n, t)) < f(B)$ ):
8           $B \leftarrow (n, t)$ 
9      for each interval  $i$  adjacent to  $n.i$ :
10          $n_2 \leftarrow newNode(i, n, T)$ 
11         if  $n_2.z \neq n.z$  AND NOT  $DRC(n_2)$ : continue
12         for each interval  $i_2$  in  $STL$  that intersects  $i$ :
13             if  $f(i_2.node) > f(n_2)$ :
14                  $i_2 \leftarrow i_2 - i$ 
15                 if  $i_2 = \emptyset$ :
16                     remove  $i_2$  from  $STL$  and  $i_2.node$  from  $Q$ 
17                 else:
18                      $i \leftarrow i - i_2$ 
19                     if  $i = \emptyset$ : break
20         if  $i \neq \emptyset$ :
21              $Q \leftarrow Q \cup \{n_2\}$ 
22              $STL \leftarrow STL \cup \{i\}$ 
23     return null

```

Figure 50- Pseudocode of DRAPS.

and h values, and a parent node. In line 3, the minimum $f(n)$ cost node is extracted from the priority queue Q , which is the A* search *Open Set*. In the case of a tie, the algorithm gives priority to the last added node, which provides a depth-first behavior in the search, unlike in (GESTER, 2013), which chooses the furthest node from the target point, presenting a breadth-first behavior, which tends to be slower. Lines 4 and 6-8 are referent to the PAP cost in the target points, which will be addressed in the next section. Line 5 checks whether n has found its target ($n.target$ is the target point sought by n). In line 9, two intervals are adjacent if they are conceptually connected by a graph edge. The $n.i$ is the interval of node n . The $newNode(i, n, T)$ function creates a search node with interval i , antecessor node n and seeks a target $\in T$. In line 11, if the newly created node belongs to another metal layer, the algorithm performs a design rule check (DRC function) that returns true if there is no violation. The DRC is explained in section 6.4. STL refers to the “satellite” data structure that stores all the open and closed nodes. The relaxation step of A* algorithm (lines 12-22) is performed on intervals. In the case of intersection of intervals, the interval whose node has the lowest

cost (or the interval in STL, in case of a tie) cuts the intersection piece of the other interval ($i - i_2$ and $i_2 - i$ represent interval subtractions).

The open set Q is implemented by a combination of data structures: a heap and a hash table. The heap stores the costs of the search nodes, with no repeated costs. The keys of the hash table are the node costs and the data of a key is a stack of search nodes with the key's cost. When a node is removed from Q , the top of the stack with the best cost is removed, giving the algorithm a depth-first-search (DFS) behavior in the case of a tie of node costs. Separating nodes with the same cost in stacks decreases the time overhead of the heap queries, since it has less elements. The little overhead of querying the hash compensates the overhead of heaving all nodes in a heap. Also, in practice, the hash is queried only once to obtain the stack.

The algorithm holds sets of line segments that represent available routing space free of obstacles. They will be referred as empty-space intervals. All intervals are created over them, except in RNR step, in which it is allowed for intervals to be created inside the routing obstacles, with a high cost penalty. The use of these empty-space intervals is justified by the fact that it is cheaper to query the data structure that stores them, than to query the grid. This may be interpreted as a cache of available routing space. These intervals are created on-the-fly, during the path search.

In order to an interval expand creating nodes in adjacent layers, it is necessary to iterate the interval while trying to create the nodes. However, there are few routing guide sections that present guides in two adjacent metal layers. Only in these sections it is pertinent for the algorithm to try to create nodes in adjacent layers. DRAPS is optimized to iterate the intervals only in these cases.

6.2 Handling PAP Costs in the Source and Target Points

The PAPs have associated costs that are considered by the path search. The cost of a PAP is the WL it uses, considering cost penalty for jogs, plus the via cost, if it is a via-PAP. These costs aim to improve these routing metrics, especially the number of vias. The high PAP selection flexibility provided by the proposed pin access method may increase the number of via-PAPs used. Thus, via-PAPs that were already implemented have cost 0 to encourage the path search to reuse them. The pin access experiments in section 5.5 were already considering this feature in DRAPS.

Handling costs in the source points is trivial: the $g(n)$, which is usually 0 for the source points, is set to the cost of their respective PAP. The problem arises when

algorithm returns the saved target node. This is performed in line 4 of DRAPS algorithm (B is the best target node found). In lines 6-8, B is updated. If $n.i$ intersects a target other than $n.target$, then this target must be saved. Since many targets may intersect $n.i$, the one that minimizes the total cost is chosen. In the pseudocode, the tuple (n, t) represents a search node that is a copy of n , except that it seeks target t . Due to the proposed technique to improve the h function, it is not always necessary to iterate over all target points to select t . This will be explained in the next section.

6.3 Tunnel Lowerbound

This section presents the proposed Tunnel Lowerbound method, published in (GONÇALVES, 2019a). The A^* is a generic path search algorithm that uses a heuristic function (h) to guide the search to the target node. This function calculates the estimated path cost from a node n to a target node. The more realistic is the estimation, the more the algorithm takes a Depth-First-Search (DFS) behavior towards the target node, and, consequently, the faster it ends.

In detailed routing, the L_1 distance is the most straightforward way to implement the h function. When $h(n)$ uses L_1 , it implicitly predicts that the path from n to the target node is of L/Z shape. However, the global routing guides have shapes that force the paths to perform detours, such as an U shape path. In these cases, using L_1 distance makes the path search to take a Breadth-First search (BFS) behavior, which is slow. On the other hand, if h is implemented such that it is aware of the global routing guide shape, it will provide a more realistic lower bound, enabling the search to perform a DFS. This was already discussed in sections 3.2.2 and 3.2.5.

6.3.1 The Proposed Technique

Each net has a global routing guide associated with. This guide consists in a set of rectangles which limit the path search space. We call *tunnel* a section of the global routing guide which connects two net components (terminals) that act as source and target of the path search (i.e., pins and wires). Each path search aims to find a path connecting two terminals, and this search is constrained by a tunnel. The function $rect(n)$ denotes the tunnel rectangle that contains the point n . We call Tunnel Lowerbound (TL) the minimum tunnel-aware-cost of a path connecting a point p to a target point which minimizes such cost. Note that there may be more than one target point since a pin usually spans more than one track. The function $TL(n)$ denotes the

TL measured at point n . The function $lb(n_1, n_2)$ returns the L_1 distance modified by the via cost, if the points do not belong to the same metal layer.

The basic idea of the proposed technique is to: 1) precompute the TL for each tunnel rectangle, before the path search; 2) during the path search, calculate the heuristic function $h(n)$ by obtaining the precomputed cost of $rect(n)$ and adding it to the offset cost between n and the reference point for the TL calculation.

A reference point (or just ref point, for simplification) is associated with a target point, and with an offset cost to such point. There may be as many ref points as target points in each tunnel rectangle, although this usually does not happen in practice, as will be seen ahead. It is necessary to maintain this relation of ref points to target points due to the following. If the target pin has multiple target points and a rectangle has only one ref point, $h(n)$ may overestimate the cost, depending on the location of both n and the target points. For example, in Figure 52, the minimum cost from n_1 to the target pin is the cost from n_1 to t_4 . If the only ref point of $rect_2$ was r_1 , then $h(n_1)$ would return the offset cost from n_1 to r_1 plus the offset cost from r_1 to t_1 ($TL(r_1)$), which is higher than the cost from n_1 to t_4 . Thus, sometimes it is necessary for a rectangle to have more than one ref point. The number and the location of the ref points of a rectangle r are determined by the projections of the ref points of the previous rectangle into r , such that the offset cost of the projection is minimized. The source rectangles of this chain of projections are the ones containing the target points. Usually there is only one such rectangle. The target points are also ref points of these rectangles. In Figure 52, the arrows denote these

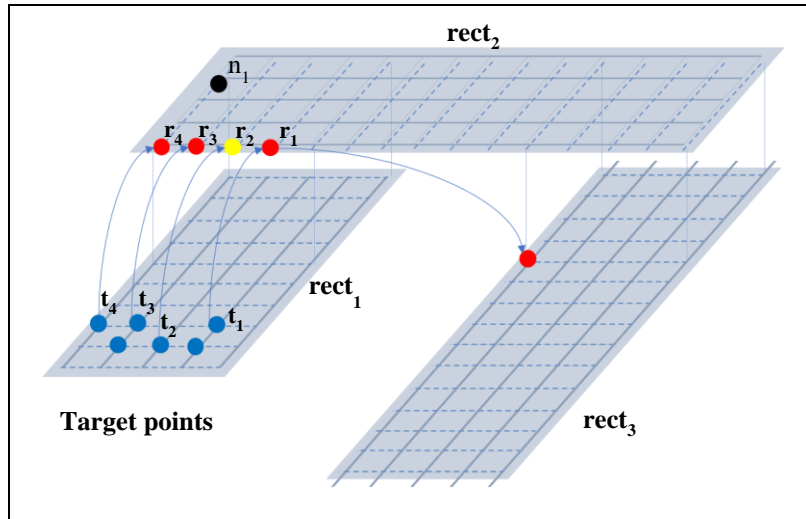


Figure 52: Illustration of the TL technique. Ref points are red and yellow.

projections. If a ref point is the result of the projection of more than one ref point, it stores the offset cost (and is related to the target point) of the lowest cost projection.

The heuristic function h is defined as

$$h(n) = \min\{lb(n, r) + TL(r) \mid r \text{ is a ref point of } rect(n)\},$$

implying that each time h is called, it iterates over the ref points of the rectangle in order to obtain the lowest total cost. Thus, this operation is performed in $O(k)$ time, where k is the number of ref points of the rectangle. However, this number is very small in practice. As the projections propagate over the rectangles, they quickly reduce the number of ref points to just one. This happens at the second rectangle bend (see $rect_3$ of Figure 52). Thus, for a tunnel with at least two bends and many rectangles, the majority of the rectangles have only one ref point. In the experiments evaluating this technique, most rectangles have only 1 ref point, and the average number of ref points by rectangle is only 3. The iteration over ref points can be even improved by storing adjacent ref points as intervals or even rectangles. Therefore, in practice and in the average case, $h(n)$ performs in constant time. Also, regarding the awareness of PAP costs in the path search, it is not necessary to iterate over all target points, in line 6 of DRAPS algorithm. The search can be performed in the target points related to the ref points of the tunnel rectangle containing the expanding node.

The projections of ref points may generate redundant ref points. A ref point is redundant if, in the case it is omitted, it will not cause any cost overestimation by h . In Figure 52, r_2 is a redundant ref point because, even if the TL is measured over it, the h function will return a value that is no less than the value returned if it chooses r_1 or r_3 as ref points. Thus, redundant points should be avoided, since they are useless.

6.3.2 Algorithm and Implementation

The preprocessing before the path search consists in calculating the ref points and their TL , for all tunnel rectangles. The procedure begins in the rectangles containing the targets points. The ref points of these rectangles are projected into the intersecting rectangles of both neighboring and same layers, and so on, until all rectangles have all ref points set. This problem is modeled as a shortest path problem on a graph with non-negative edges, and is solved using Dijkstra's algorithm

(CORMEN, 2001) with some modifications. Each graph vertex is a tuple (R, p) , where R is a rectangle, which can be defined as a set of grid points, and $p \in R$ a ref point of R . The edges of a vertex are the intersections of R with other rectangles of both the same and neighboring layers (i.e., rectangles that can be directly accessed from R). The cost of an edge $((R, p), (R', p'))$ is $lb(p, p')$.

The preprocessing algorithm is described in Figure 53. Lines 1-3 perform the initialization of priority queue Q . Line 5 retrieves and removes from Q the lowest cost node. Lines 6-16 represent the expansion of the chosen node. Lines 8 and 9 check whether p' is redundant. Lines 10-16 are the relaxation step of Dijkstra's algorithm. The priority queue Q stores search nodes (R, p, δ) , where (R, p) is a vertex and δ is the offset value of p to the target pin, i.e, $TL(p)$. Q is sorted by δ . Each rectangle is a data structure that stores all its ref points and its respective offset values δ . The function $addRefPoint(R, p, \delta)$ stores the ref point p in rectangle R with offset value δ . The function $refPoints(R)$ denotes the list of ref points stored in R . The $TL(x)$ function of the pseudocode is actually accessing the offset value δ of x stored in R' .

An A*-based path search algorithm handling multi target points must choose which target point it is seeking. However, using the proposed technique, the ref point selection replaces this operation. This is due each ref point is indirectly related to a target point. Target points may also be stored in the rectangles, for each ref point, in the $addRefPoint$ function, and updated with the target point of p , together with $TL(p')$,

ALGORITHM: *DijkstraTL*(Tunnel tn)

```

16 for each  $n \in \{(R, t, 0) \mid t \in R \text{ is a target point}\}$ :
17    $Q \leftarrow Q \cup n$ 
18    $addRefPoint(R, t, 0)$ 
19 while  $Q \neq \emptyset$ :
20    $(R, p, \delta) \leftarrow pop(Q)$ 
21   for each neighbor  $R'$  of  $R$ :
22     Calculate  $p' \in R'$  such that  $lb(p, p')$  is minimized
23     for each ref point  $r$  of  $R'$ :
24       if  $lb(p', r) + TL(r) \leq \delta + lb(p, p')$ : go to 4
25     if  $p' \in refPoints(R')$ :
26       if  $\delta + lb(p, p') < TL(p')$ :
27          $TL(p') \leftarrow \delta + lb(p, p')$ 
28         Update  $(R', p', TL(p'))$  in  $Q$ 
29     else:
30        $addRefPoint(R', p, \delta + lb(p, p'))$ 
31    $Q \leftarrow Q \cup \{(R', p', TL(p'))\}$ 

```

Figure 53- Pseudocode of the proposed algorithm to precompute the tunnel lowerbounds.

in line 12. The target point of a ref point will always be the same of its parent ref point. The complexity of Dijkstra's algorithm with ideal implementation is $O(|V| \times \log |V| + |E|)$, where $|E|$ is the number of edges and $|V|$ is the number of vertices. In our problem $|E| \cong |V|$, since, in most of cases, each rectangle has only one successor rectangle in the expansion chain. For each graph edge traversed, there is also the redundant point verification overhead, which iterates over all ref points of the rectangle in question (R' , in the pseudocode). Considering p is the average number of ref points by rectangle, $|E|$ can be substituted by $|V| * p$. $|V|$ is equivalent to $r * p$, where r is the number of rectangles in the tunnel. In the worst case, each rectangle stores a number of ref points equals (or similar) to the number of target points t . Thus, the worst case complexity of *DijkstraTL* is bounded by $O(rt \log rt + rt^2)$. However, this happens only in tunnels with few rectangles and less than one bend, which is a minority, and consists in the simplest routing cases. Thus, this worst case bound is far from the average case. In average, the majority of the tunnel rectangles have only one ref point. Therefore, $p = 1$, and the complexity of *DijkstraTL* in the average case is bounded by $O(r \log r)$.

The proposed method is robust in the sense that it does not require any specific conditions, regarding the tunnel properties, to work. The tunnel may be any set of rectangles of arbitrary sizes. Also, the target points may be present in any rectangles. Regarding optimality, it is evident that $TL(r)$, where r is a ref point, is the minimum path cost from r to its target (i.e. it is a lowerbound). This is guaranteed by the fact that each ref point projection generates ref points in positions that minimize their distance and, in the case of a projection over an existing ref point, the lowest cost (TL) projection receives priority. Regarding $h(n)$, since it chooses a ref point that minimizes the total cost, it returns the minimum cost between n and the target point set. Thus, $h(n)$ is a proper lowerbound, making it *admissible* and *consistent*, which ensures the A* optimality.

6.4 Design Rule Handling

One of the biggest challenges of detailed routing is the design rule handling. If the router tries to solve DRVs in routing postprocessing, many DRVs will not be solved. Thus, it is necessary to adopt a correct-by-construction approach. Making the path search algorithm aware of some design rules is essential for a good design rule handling. DRAPS is aware of the via library, the min area rule and the cut-cut spacing

violations that occur inside the same path. In DRAPS, the DRC function (in DRAPS algorithm, Figure 50), performs design rule checking, possibly disallowing the candidate new node to be added in the open set. The same function (DRC) handles the three cases of design rule violations that DRAPS is aware, since the circumstances that enable the creation of DRVs by these cases are basically the same, as seen ahead. The pseudocode of the DRC function is presented in Figure 54, and will be addressed in the next sections. DRAPS was published in (GONÇALVES, 2019b)

ALGORITHM: DRC(Node n)

```

1   $p \leftarrow$  first predecessor node  $n'$  of  $n.parent$  with  $n'.z \neq n.parent.z$ , or null, if
    $n'$  does not exist
2  if  $p \neq \text{null}$  and  $n'.z = n.z$  and  $cutSpacingViolation(n', n)$ :
3    return false
4  if the area of the traversed path  $< minArea(n.parent.z)$ :
5    if there is not enough space around  $n.parentSegment()$  to extend it such
      that  $area(n.parentSegment()) \geq minArea(n.parent.z)$ :
6      return false
7   $viasInRange \leftarrow viaMap.getViasInRange(n.point)$ 
8  if  $viasInRange \neq \text{null}$ :
9    for each via  $v$  in  $viaList$ :
10     if  $hasViaViol(v, viasInRange)$ : continue
11     break
12  if all vias in  $viaList$  present violation: return false
13 for each via  $v$  in  $viaList$ :
14   if  $hasViol(v, n, grid)$ : continue
15   break
16 if all vias in  $viaList$  present violation: return false
17 return true

```

Figure 54- Pseudocode of the DRC function.

6.4.1 Via Checking

The detailed router is restricted to use a set of vias defined in a via library. If the vias used to connect the wires are chosen after the path is found, it may be impossible to find a via that does not create any DRV, such as end-of-line and short violations (a short violation happens when two shapes of different nets make contact).

DRAPS is aware of the via library. When an expansion implies in a via, it chooses the most suitable via, and queries the *via map* (section 4.1) and possibly the *grid* to know whether the via can be used or not. If the via cannot be used, it chooses the second most suitable via and so on. The algorithm stores a sorted list of vias for each cut layer. The via ordering in these lists is given by the criterion that defines a via as “most suitable” than others, discussed ahead. Lines 7-17 of the DRC function

describe this procedure. Lines 7-12 will be better explained in the next subsection. These lists are created before the routing step, by preprocessing the via library.

A via is considered more suitable than other via if the metal widths, in non-preferred routing direction, of the top and/or bottom shapes are shorter. If the bottom width is shorter, the via is more suitable. In case of a tie, the top is checked. A via with larger metal width, in non-preferred routing direction, tends to block adjacent tracks. The bottom shape is checked first since lower metal layers are more disputed, and thus, it is preferable to block adjacent tracks of the upper layers. Figure 55 illustrates the via selection order.

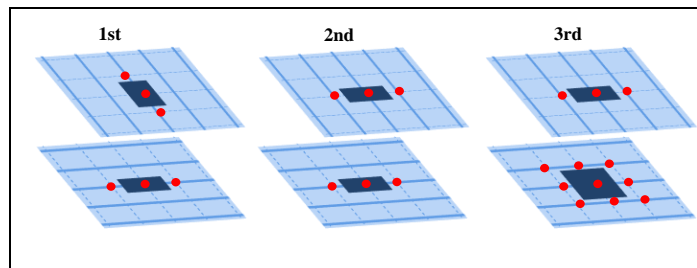


Figure 55: Via selection order and via blockage information (red dots). Dark blue rectangles are the via metal shapes.

6.4.1.1 Efficient Via Queries

In order to query the grid to check whether the via can be used or not, it is necessary to know the tracks and the track extension the via occupies, taking into consideration both its shape and its minimum spacing requirements. Due to performance matters, this via blockage information is calculated for each via definition before routing, together with the sorted via lists (see Figure 55), and stored in a LUT. In a grid query of a via during routing, the predefined via blockage in the LUT is used. The via blockage information is stored on intervals.

If the via queries are performed using only these intervals, it is still possible to exist violations with other vias, as shown in Figure 56. One solution is to use more querying intervals in the via surroundings, but this is more costly in runtime terms. A via pad aligned with the preferred direction track usually requires only one grid query, and, in this solution, it would require three queries. Another solution is to keep the queries as they are but making the vias occupy more space on its surroundings. However, this creates more blockages than necessary, which is unfavorable for routability.

The solution used by DRAPS is to make use of the via map, proposed in section 4.1. The via map was designed for storing cut layer information of vias, but it is also

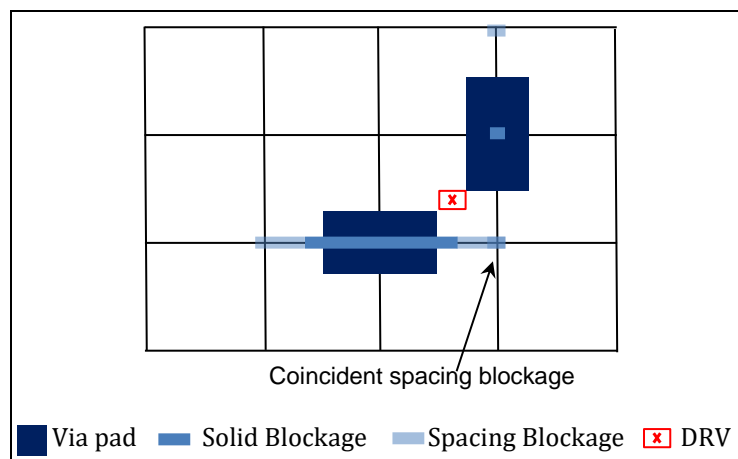


Figure 56- Illustration of a situation where querying the grid with the via blockage information is not enough to avoid DRVs. The two vias induce a blockage on the same point, but since both are spacing blockages, they may coexist. The preferred routing direction is horizontal.

used in DRAPS to check via pad violations, meaning that the via map does not store only the via cuts, but the entire vias. Thus, with a single query in this data structure, it is possible to know whether the candidate via presents a cut spacing violation or a violation in the via pads with other vias. However, a query in a given cut layer of the via map is blind to the via pads of the vias of adjacent layers. Thus, it is necessary that each cut layer in the via map store the vias from the adjacent cut layers. For example, the cut layer via2 stores the vias of via1, via2 and via3. Considering that n is the number of vias in a slice of a cut layer in the via map without this merging technique, a query in a slice is $O(\log n)$ to find the first via (this is added to $O(k)$ to find all k vias in a range). With the proposed approach n is multiplied by 3 on average, but this is still better than to perform 3 queries in the via map, since $O(\log 3n) < O(3\log n)$.

The via check in the via map is done in lines 7-12 of the DRC function. All vias in a range around the point of the expanding node are obtained with a single query. Then, if there are any such vias, they are geometrically compared against the candidate via for design rule checking.

Even if the via check in the via map does not accuse any violation, it is still possible to exist violations with other routing objects, such as wires, pins or cell obstructions (PAPs are being considered as wires here). Thus, it is necessary to perform queries in the grid. Although not implemented in DRAPS, it is possible to identify the vias that presented violation in the via map check to avoid checking them again. Anyway, the via checking in the grid tends not to be too expensive, due to the following. When a via check in the grid occurs, DRAPS has already obtained the empty-space intervals, in which the search nodes are stored, of the top and bottom

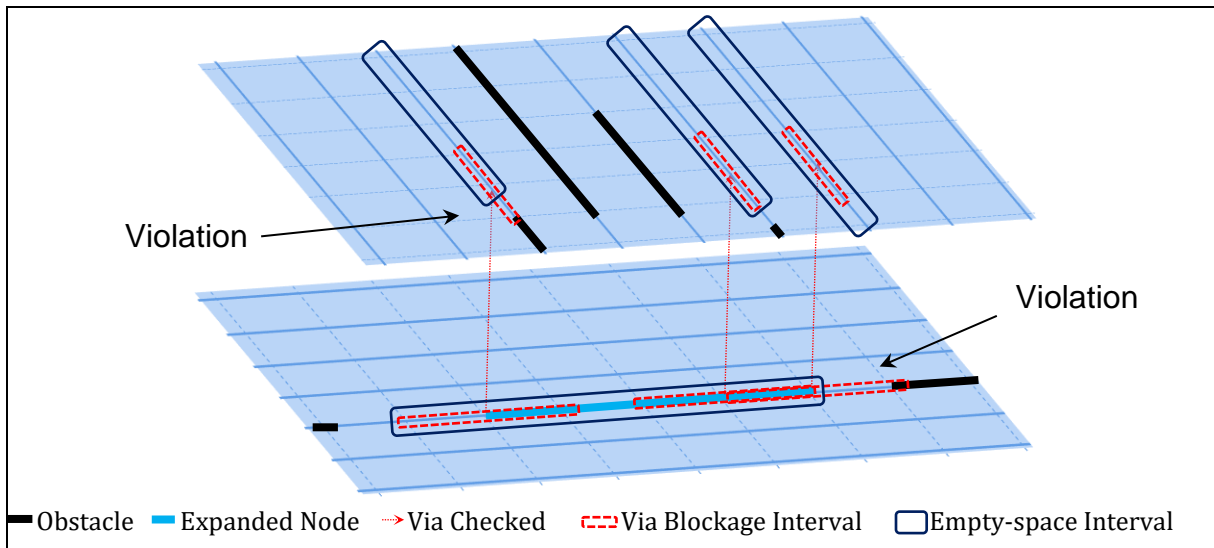


Figure 57- Illustration of the via check using the empty-space intervals stored by DRAPS. When the via pads are aligned to the routing tracks, and the vias are not from higher metal layers, checking the via blockage interval against the empty-space interval is enough to know whether the via causes a violation or not. The empty-space intervals may hold the information whether the adjacent obstacles are spacing blockages, to allow creation of vias whose spacing blockage is out of the empty-space interval.

tracks of the via to be used, as shown in Figure 57. The majority of the via pads, especially the ones of vias in the first place of the sorted via lists, have their blockage information occupying only one track by layer, as in Figure 55. In this case, if the empty-space interval contains the via blockage interval, then the via pad is passed in the check; otherwise, it presents violations, and the via cannot be used. Thus, in these cases, it is not necessary to query the grid, and the via check is very fast.

The pins may also present violation situations similar to the one in Figure 56. In order to guarantee that the vias do not cause violations with pins, DRAPS performs additional grid queries in the surroundings of the metal 1 via pads. If a pin blockage is found on this extended query region, then the pin and the via are geometrically checked for violations. Although this is more costly, as mentioned earlier, these queries in metal 1 are rare and thus, they do not cause alarming runtime impact.

6.4.2 Cut Spacing on Same-Path Vias

It is possible to control cut spacing violations the same way as metal spacing, i.e, storing position and occupation information of via cuts in the *grid*. However, during a path search, the vias used by the paths are not (and should not be) added in the routing space, since a path that is being searched is not necessarily the path that will be implemented. Thus, without a control of the vias used inside a path search, it is

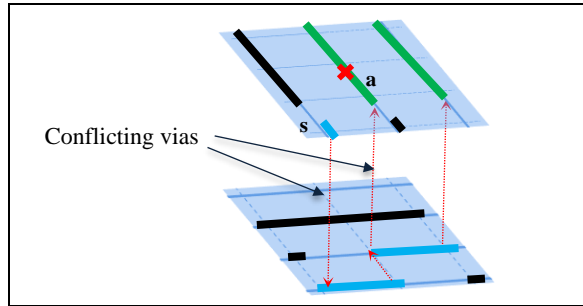


Figure 58: Illustration of a path search scenario with a cut-cut spacing violation within the path. We are assuming the minimum center-to-center via distance is 2 tracks of extension. Node *a* cannot be added into the open set.

possible to occur cut spacing violations of via cuts of the same path after the path is found.

Each time a node expands generating a node in an adjacent layer, this implies the use of a via (as node *a* in Figure 58). When this happens, the DRAPS algorithm backtracks the node path until it finds the first via (line 1 of DRC function). In Figure 58, this via is the one that connects the *s* point. Then, it checks whether the new via causes cut spacing violation with this via. If there is a violation, the created node is not added in the open set. This simple heuristic presents nice results without compromising runtime, as shown in the experiments ahead.

6.4.3 Minimum Area

The minimum area rule requires a metal shape, of a given routing layer, to meet a minimum area. A straightforward approach to solve min area violations is by a post-processing, after routing: pieces of metal which violated the min area requirement are extended. However, it is possible that there is no space available for such extensions, and thus, new violations are created. This can be avoided if the path search algorithm is aware of this design rule.

The proposed algorithm uses the following strategy to handle the min area rule. When the path search creates a node in a different layer from the parent node, the DRC function is called. Then it backtracks the path, as mentioned before (line 1 of DRC function), while storing the area of the path section. If this area does not meet the min area, then the *grid* is queried in order to know whether the wire extension is possible or not (lines 4-6). In Figure 59a, the creation of node *a* implies a previous path that satisfies the min area. In Figure 59b, the previous path does not satisfy the min area, but the path segment can be extended in either direction to meet the area. In Figure 59c, node *a* cannot be created since the previous path does not meet the min area and

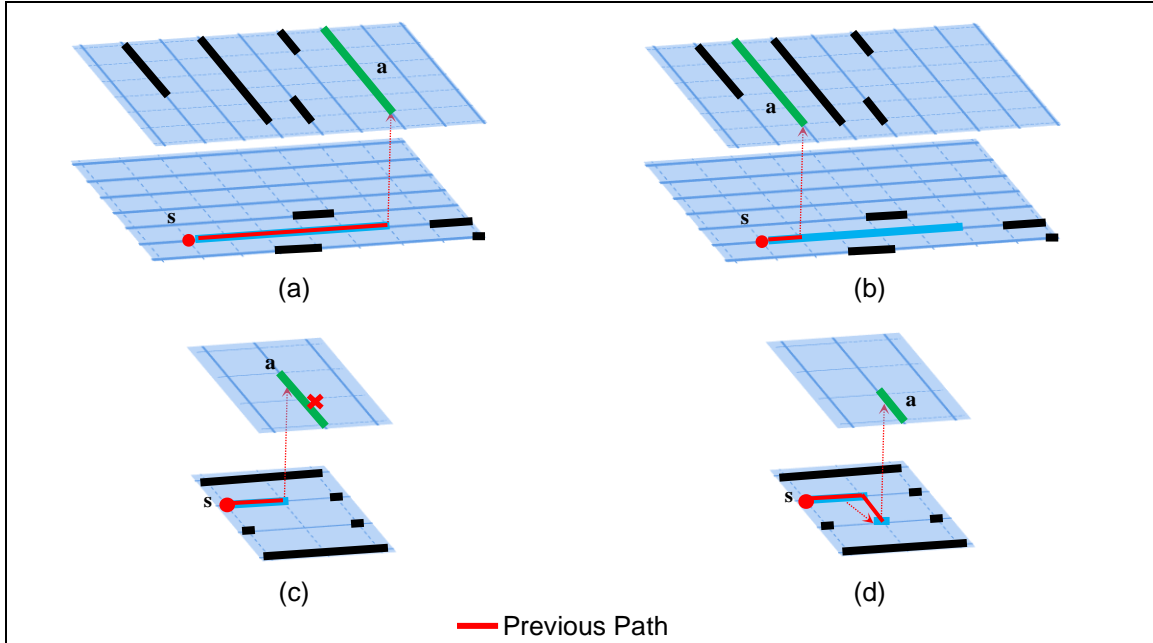


Figure 59: Illustration of situations of min area check. We are assuming that the min area requires a wire segment of at least 2 pitches.

there is not enough space to extend it. Note that we perform the extensions only along the same track of the expanding node, and always on the path segment over it. In Figure 59d, the path formed in the lower layer has enough area, so node *a* can be created. In the DRC function, the *parentSegment* function denotes the segment that connects the newly created node *n* (which is *a* in Figure 59) to its parent node in its metal layer.

The proposed approach does not guarantee the optimal path if we consider the extensions length in the path's cost. It guarantees the optimal path if we ignore the extensions length though, just as Hetzel's algorithm guarantees the optimal path. This constraint was relaxed since it tends to present little impact on WL. The extensions are very short, and are performed in a minority of cases, mostly on metal 2 and metal 3, where the path goes up and down to access the pins. The largest part of a path is composed by long wire segments in the upper layers. This is further discussed in the experiments section 6.6.3.

6.5 Comparison with Related Work

6.5.1 DRAPS

The basic path search mechanics is the same of the path search in (GESTER, 2013), which is based on Hetzel's algorithm (HETZEL, 1998), except by the tie breaking criterion, where DRAPS gives priority to nodes last added in the open set,

and GESTER, 2013) choses the furthest node from the target point. All the other features of DRAPS, as design rule handling and source/target costs, are not present in (GESTER, 2013) .

Table 5 presents a comparison of DRAPS with the other design rule aware path search algorithms in the literature. “Intervals” represents whether the algorithm uses the interval labeling technique or it is a maze search. “Min Area” and “Cut Spc” refer to the awareness of the min area rule, and cut spacing of vias within the same path, respectively. “Vias” is the awareness of the via library and techniques used in the via queries during the path search. “S/T Costs” refers to handle costs in the source and target points.

DRAPS and (AHRENS, 2015) use an interval-based approach, but in (AHRENS, 2015), the algorithm is multi-label and runs in a graph especially constructed for the multi-label system. Regarding the min area handling, (AHRENS, 2015) handles the min area by making this special grid to have a pitch long enough to guarantee that any wire segment meets the minimum area. The algorithm guarantees the optimal path in this grid, but the path length may not be the shortest considering the original grid. DRAPS, (CHANG, 2013) and (CHEN, 2019a, 2019b) handle min area by identifying possible wire extensions. (CHANG, 2013) and (CHEN, 2019a, 2019b) guarantee the optimal path with the extensions, while DRAPS do not. (CHEN, 2019a, 2019b) allows the wire extensions to create DRVs, in some cases. Regarding the via handling, (CHEN, 2019a, 2019b) and DRAPS build LUTs containing the via blockage information. DRAPS makes the queries even more efficient by using the proposed via map, and by exploiting the empty-space intervals built of the path search, which act as a cache of available search space. Finally, DRAPS is the only algorithm that reports dealing with the cut spacing of same-path vias, and is also the only that treats costs in

Table 5: Comparison with design rule aware path search algorithms in literature.

Algorithm	Intervals	Min Area	Vias	Cut Spc	S/T Costs
DRAPS	yes	non-optimal, extensions	LUTs, via map, caching	yes	yes
(CHANG, 2013)	no	optimal, extensions	no	no	no
(CHEN, 2019a, 2019b)	no	optimal, extensions, allow DRVs	LUTs	no	no
(AHRENS, 2015)	yes	optimal*, pitch	no	no	no

the source/target points.

6.5.2 Improved Lowerbounds for the A* Heuristic Function

This section compares the proposed Tunnel Lowerbound technique to (PEYER, 2009). In (PEYER, 2009), the lowerbound of a point in a rectangle is given by a predefined function. In order for this function to provide proper lowerbounds, it is necessary to alter its coefficients according to the situation, in the preprocessing step. Also, the rectangle must meet constraints so that the function can properly work. Thus, the tunnel is partitioned in many rectangles, such that each of them meet the mentioned constraints. Each rectangle has a list of coefficients for the lowerbound function.

In the proposed approach, no rectangle splitting is necessary. The lowerbound is not given by a default function. The rectangles have reference points for each target point. Each of them stores the offset cost from them to their respective target. During the path search the heuristic function $h(n)$ choses the reference point that minimizes the cost from n to the target pin. Although the proposed approach may seem less efficient in the path search, this does not happen, since most rectangles have only one reference point. Also, probably due to the high rectangle splitting of (PEYER, 2009), the precomputation of lowerbounds leads to unfeasible runtimes, as discussed in section 6.6.2.

6.6 Experiments

This section presents experiments to evaluate the proposed path search techniques, namely: PAP costs in the source/target points in the path search (Section 6.6.1), tunnel lowerbound (section 6.6.2) and design rule awareness in the path search (section 6.6.3).

6.6.1 PAP Cost Aware Path Search

The cost of a PAP is the WL it uses, considering cost penalty for wrong-way WL (i.e., WL of jogs), plus the via cost, if it is a via-PAP. If a via-PAP is already implemented in the routing space, the cost is 0. Table 6 shows the results of the impact in considering costs in the PAPs. “WL Red.” is the reduction in WL when adopting the proposed technique. “Via Red.” is the reduction in the number of vias. “WW Red.” Is the wrong-way WL reduction. The “Time” columns show the total runtime with and without the

Table 6: Results of Using PAP Costs in the Path Search.

Bench	WL Red. (%)	Via Red. (%)	WW Red. (%)	Time		
				With	Without	Red. (%)
test1	1.2	12.4	35.0	16	16	0.00
test2	0.6	10.3	50.6	71	76	6.58
test3	0.5	10.6	48.7	157	167	5.99
test4	0.5	6.1	52.4	304	314	3.18
test5	0.5	6.1	64.3	208	215	3.26
test6	0.6	6.0	61.5	286	296	3.38
test7	0.7	6.3	67.9	429	439	2.28
test8	0.6	6.3	67.2	435	429	-1.40
test9	0.8	6.4	68.1	405	420	3.57
test10	0.7	6.5	61.2	717	738	2.91
Avg	0.68	7.8	57.2			2.98

PAP cost awareness, and the runtime reduction (“Red.”) brought by it. A slight WL improvement (0.68, in average) is obtained, since the path search prefers PAPs with less WL. Wrong-way WL is greatly decreased (57%, in average). The most important reduction is in the via count, and this is the main goal of the proposed technique. It provided an average of 11.1% of reduction, for the 45nm benchmarks, and 6.3% for the others. The average reduction for all benchmarks is 7.8%. The runtime was not significantly affected by handling costs in PAPs. It actually decreased by 2.66% in average. This difference in the via reduction of benchmarks of different technologies is probably due to the fact the routing over the pins in the 45nm benchmarks is less congested than in the 32nm ones. This makes the algorithm more able to reuse the same implemented via-PAPs, in 45nm benchmarks, while in the 32nm ones it is harder to reuse them. A possible solution is to increase the via cost in the path search, so that the algorithm will make a higher effort to avoid using new via-PAPs.

6.6.2 Tunnel Lowerbound

This section presents runtime results of using the proposed Tunnel Lowerbound (*TL*) technique. It is presented a comparison of SmartDR with DRAPS using *TL* and the L_1 distance. Table 7 shows the results. The column names follow the same pattern from the other tables. “Total Real Time” is the total time of SmartDR using 8 threads. All “Total Path Search” columns show the total time spent on all threads. “Pre” is the tunnel preprocessing time before the path search. “Red. PS” is the runtime reduction, in the path search, of the *TL* w.r.t. L_1 . “Red. PS+Pre” is the same of “Red. PS”, but considering both path search and preprocessing time. The average path search

Table 7: Results of the Proposed Tunnel Lowerbound Technique

Bench	Total Real Time		Red. Total (%)	Total Path Search Time			Red. PS (%)	Red. PS+Pre (%)
	TL	L ₁		TL	Pre	L ₁		
test1	16	23	30.4	9	0.4	17.7	49.2	46.9
test2	71	131	45.8	82	2.6	197	58.4	57.1
test3	157	188	16.5	166	2.8	258	35.7	34.6
test4	314	328	4.3	256	12.4	424	39.6	36.7
test5	208	365	43.0	232	11.3	620	62.6	60.8
test6	286	515	44.5	340	18.1	866	60.7	58.6
test7	429	718	40.3	571	23.1	1379	58.6	56.9
test8	435	742	41.4	565	23.3	1378	59.0	57.3
test9	405	713	43.2	557	24.2	1352	58.8	57.0
test10	717	1069	33.0	1482	25	2416	38.7	37.6
Avg			34.2				52.1	50.3

runtime reduction is 52.1%. Considering the preprocessing time, it is slightly decreased to 50.3%. This shows that the proposed preprocessing method is very efficient. The path search time reduction brings a total runtime reduction of 34.2%, in average. The testcases 3, 4 and 10 present a deviation of the runtime reduction since they present harder RNR effort. In RNR, TL is used only when one of the path search terminals is a pin, and this usually does not happen, since there are many path searches connecting wire segments. Also, in test4, the pin handling time takes a considerable portion of the total time, around 20%, and this happens for both TL and L_1 .

6.6.2.1 Comparison with (PEYER, 2009)

Since the benchmarks used in (PEYER, 2009) are not publicly available, it was not possible to run the proposed technique in these benchmarks. Also, in (PEYER, 2009), the methods were presented in a high abstraction level, lacking implementation details, which discouraged their implementation. Still, it is possible to reach a conclusion by comparing some relative values.

The basic idea of both TL and (PEYER, 2009) is the same: the tunnel rectangles are preprocessed before the path search, calculating the offset costs to the target pin, and, during the path search, this information is efficiently used to quickly guide the search to its destination. The output of both preprocessing methods is the same, regarding the offset values δ of a rectangle, although this information is represented differently, as they split the tunnel rectangles and the TL method do not. Also, the path search algorithm used in (PEYER, 2009) was the same of (GESTER, 2013), which has

Table 8: Results of the method proposed in (PEYER, 2009).

bench	L_1	Hybrid		New		Red. PS Hyb. (%)	Red. PS+Pre Hyb. (%)	Red. PS New (%)	Red. PS+Pre New (%)
		PS+Pre	Pre	PS+Pre	Pre				
Dieter	607	481	20	1479	1047	24.1	20.8	28.8	-143.7
Paul	580	459	25	1590	1187	25.2	20.9	30.5	-174.1
Lotti	827	688	35	1379	815	21.0	16.8	31.8	-66.7
Hannelore	1546	1031	48	4355	3370	36.4	33.3	36.3	-181.7
Elena	4604	3811	169	8191	4833	20.9	17.2	27.1	-77.9
Heidi	6160	5077	382	14571	10199	23.8	17.6	29.0	-136.5
Garry	9715	8142	529	20552	13247	21.6	16.2	24.8	-111.5
Edgar	12020	8627	555	19898	12215	32.8	28.2	36.1	-65.5
Ralf	11651	9599	544	42862	33944	22.3	17.6	23.5	-267.9
Hermann	51190	39568	1450	83235	43248	25.5	22.7	21.9	-62.6
Avg						25.4	21.1	29.0	-128.8

the same basic path search mechanics as DRAPS, as an A*-interval-based search. Thus, in terms of runtime, both path searches are equivalent¹. If both methods deliver the same lower bound information to the “same” path search, and the time spent to use this information, for the proposed method and (PEYER, 2009), is $O(1)$, then the benefit of both methods to the path search time is the same. Thus, the practical difference of the proposed technique and (PEYER, 2009) is in the runtime of the preprocessing methods.

Table 8 presents the results of Table 4 of (PEYER, 2009), with some added columns. “ L_1 ” is the path search time using the L_1 distance. “New” represents using their preprocessing before every path search, and “Hybrid” means using the preprocessing only in some circumstances. “PS+Pre” is the total time spent on both the path search and the preprocessing. All other column names follow the same pattern as in Table 7. “Hyb.” and “New”, in the last four columns, are referring to the “Hybrid” and “New” columns.

Table 9 presents a comparison of relative values between the proposed TL method and (PEYER, 2009). The values in Table 9 represent an average of all benchmarks. “Always Pre” refers to using the preprocessing before every path search. The column names follow the same patterns as before. In (PEYER, 2009), if the preprocessing is performed before every path search, the total runtime is *higher* than

¹ This is obviously an approximation, since DRAPS has an overhead for DRC and a runtime advantage for being more DFS-directed.

the routing time using just L_1 distance. The runtime *increase* reached an average value of 128%. In *TL*, we have a time *reduction* of 51.8%. In (PEYER, 2009), the time spent on preprocessing was 66% in average of the total runtime of a benchmark. In *TL*, this value is 3.6%. Note that the path search algorithm used in (PEYER, 2009) has the same A*-interval mechanics of DRAPS, and thus, these differences in the proportions are mainly due to the preprocessing. This information shows that the proposed preprocessing method is faster than the one in (PEYER, 2009), which is a contribution over the only existing work addressing this problem.

This is perfectly explained by the rectangle partitioning of (PEYER, 2009). A preprocessing method sensible only to the number of tunnel rectangles would never have a runtime greater than the path search runtime, since the path search complexity is proportional to the number of intervals, which is higher than the number of tunnel rectangles. Besides, in the worst case, the number of intervals is the number of points in the rectangle, which jumps to other order of magnitude. Thus, considering that the path search time cannot be lower than a tunnel preprocessing time proportional to the number of tunnel rectangles, the only thing that can make the tunnel preprocessing time higher is if it is proportional to something greater than the number of tunnel rectangles. This is the case of (PEYER, 2009), since the complexity of their method depends on the number of partitioned rectangles and also the number of possible coefficients in the distance functions. More precisely, the complexity of their method is $O(k_1 k_2 V \log(k_1 k_2 V))$, where k_1 and k_2 are the number of possible x and y coefficients, respectively, and V is the number of partitioned rectangles. In the proposed method, the complexity is $O(r \log r)$, where r is the number of tunnel rectangles. This complexity is also higher than the number of tunnel rectangles, but it is much closer to it than the complexity of the method in (PEYER, 2009).

In order to provide a runtime benefit, (PEYER, 2009) used a heuristic to identify cases where the use of the preprocessing was worthwhile. The “Selective Pre” columns in Table 9, and the “Hybrid” columns, in Table 8, refer to this. Using the

Table 9: Comparison of relative results between *TL* and the method in (PEYER, 2009).

Method	Always Pre (%)			Selective Pre (%)		
	Red. PS	Red. PS+Pre	Pre/PS+Pre	Red. PS	Red. PS+Pre	Pre/PS+Pre
<i>TL</i>	52.1	50.3	3.6	-	-	-
(PEYER, 2009)	29	-128	66	25	21	5.3

heuristic, they reached a total runtime improvement of 21%. We cannot compare this value to the 50.3% of *TL*, since this depends on the benchmark and mostly on the tunnel shapes. However, as the “Red. PS” of “Always Pre” column shows, the maximum possible runtime reduction on the benchmarks in (PEYER, 2009) is 29%. This value is the path search runtime reduction in the case of applying the method before every path search. Thus, using the heuristic, they were able to extract about 73% of the maximum runtime benefit. Using *TL*, it is possible to extract 96.6% of it, since the preprocessing is used before every path search, and its runtime is negligible. In other words, these statistics predict that, if the proposed method was run in the experiments in (PEYER, 2009), the runtime reduction would be about 29%.

6.6.3 Design Rule Aware Path Search

In order to evaluate the effectiveness of handling design rules in the path search algorithm, two routing flows were executed: one using default DRAPS and other using DRAPS with design rule awareness turned off. In the scenario without design rule awareness, the design rules were handled in the following manner. The min area rule was solved in the Post Processing step of the router, by extending the wire segments that did not meet the min area requirement, even if the extension created new DRVs (the algorithm tries to avoid such DRVs though). The via selection was also executed in the same Post Processing step. The algorithm tries to select the most suitable via (using the same sorted list of vias that DRAPS use) that causes no DRV. If no via is DRV-free, the best via of via via list is chosen. The cut spacing of same-path vias was not solved, since there is no straightforward way to handle it.

Table 10 presents the results regarding total routing time and DRV count. “No DRC” refers to DRAPS algorithm without handling any design rules. “Via ON” is DRAPS aware only of Via Selection strategies. “Via + MAR ON” is DRAPS aware of Via Selection and Min Area. “Full DRC” is the default DRAPS. “Cut Sp” are the number of cut spacing violations. “Total Sp” are the number of all spacing violations. “Shorts” are the number of short violations. A short violation happens when two shapes of different nets make contact. The time is measured in seconds.

The via library awareness provides an average DRV reduction of 69%, w.r.t. “No DRC”. The Min Area awareness further reduces these violations, since the router does not need to extend wires in post processing, risking to create spacing and short violations. Almost all short violations are solved after min area handling is turned on.

Table 10: Results of Design Rule Violations of DRAPS

Bench	No DRC					Via ON				
	Cut Sp	Total Sp	Shorts	Total	Time	Cut Sp	Total Sp	Shorts	Total	Time
test1	0	2348	43	2391	16	0	114	52	166	18
test2	1	23208	454	23662	63	0	705	373	1078	78
test3	19	22298	1801	24099	63	0	718	413	1131	192
test4	23652	67588	8118	75706	152	1406	21157	670	21827	256
test5	68409	214873	39209	254082	177	1596	84159	11110	95269	212
test6	100373	257576	63733	321309	252	2250	131638	17726	149364	287
test7	153630	390877	97306	488183	347	3906	200262	28310	228572	422
test8	156286	396582	99814	496396	355	4007	202919	29184	232103	421
test9	155309	397216	99901	497117	363	3790	205892	29800	235692	409
test10	173974	469310	125130	594440	440	4194	224109	33628	257737	714

Bench	Via + MAR ON					Full DRC				
	Cut Sp	Total Sp	Shorts	Total	Time	Cut Sp	Total Sp	Shorts	Total	Time
test1	0	1	0	1	18	0	1	0	1	16
test2	0	9	0	9	80	0	6	0	6	71
test3	0	33	0	33	161	0	33	0	33	157
test4	1416	1609	16	1625	282	0	204	15	219	314
test5	1594	1674	3	1677	222	0	75	8	83	208
test6	2271	2521	0	2521	306	0	228	0	228	286
test7	3918	4233	14	4247	471	2	300	13	313	429
test8	3996	4345	11	4356	441	0	351	12	363	435
test9	3804	4036	3	4039	450	0	238	4	242	405
test10	4181	5563	1542	7105	728	42	1398	1349	2747	717

The technique to handle cut spacing of same-path vias also solves almost all cut spacing violations, and slightly improves the runtime (4.8% in average), since it prunes some search nodes. The average DRV reduction of DRAPS “Full DRC” w.r.t. DRAPS “No DRC” is 99.8%, and the average runtime increase is of 42%.

Results regarding WL and via count were also measured. The design rule handling has no significative impact on the number of vias. There was a via count decrease of “Full DRC” w.r.t. “No DRC” of 0.37%, in average. The major WL change came from the min area awareness. There was an average WL increase of 2.9% of “Via + MAR ON” w.r.t. “Via ON”. This is expected since it is necessary to perform detours to avoid the DRVs of the wire extensions. The total WL of all wire extensions was also measured, and its average proportion w.r.t. the total WL is 5%. This serves as a upperbound for the lack of optimality of DRAPS, when considering the WL of the wire extensions. This means that it is impossible that more than 5% of the total WL is caused by the lack of optimality. However, it is very pessimistic to even consider that the non-optimality causes something near this 5% of WL, because this implies that almost not extensions were performed, which is not realistic.

6.7 Conclusions and Future Works

Section 6 presented the proposed path search techniques. The proposed path search algorithm (DRAPS) is a new design rule aware A*-interval-based path search. The algorithm also properly handles costs in the source and target points, which interconnects it with the proposed pin access method, mitigating the increase in the via count brought by the high PAP diversity. Regarding design rule handling, DRAPS presents new features: the awareness of cut spacing of same-path vias and the efficient via check. For the via check, a new data structure to efficiently manipulate vias was proposed. It was also proposed a new method to preprocess the tunnels to obtain more realistic lowerbounds for the h function. Comparing to other path searches in literature, DRAPS is the only algorithm that unites so many features to improve runtime and design rule handling.

The proposed path search techniques were evaluated. The experiments showed that it is essential to the path search algorithm to be aware of the design rules handled by DRAPS. Also, it was shown that DRAPS is able to provide a good design rule handling without compromising runtime. The experiments also showed that the PAP cost awareness in the path search considerably mitigates the increase in via count brought by the proposed pin access approach. Thus, both techniques make a good match. Finally, the proposed tunnel lowerbound method was shown to be efficient, since it presents a light preprocessing and is able to be used before every path search, considerably reducing the path search and the overall runtime.

As future works, it is intended to create a bidirectional version of DRAPS, evaluate its effectiveness. This seems to imply in a bidirectional TL preprocessing. It is also intended to make the TL method consider some routing obstacles. The source and target cost handling may also incorporate costs regarding patch metals and TMSs.

7 Comparison with State-of-the-Art Routers

In this section SmartDR is compared against the state-of-the-art academic routers that were tested using ISPD18 benchmarks in (CHEN, 2019a), (CHEN, 2019b), (KAHNG, 2018) and (SUN, 2018). Their binaries were obtained and were executed in the same machine SmartDR was run. The routers were executed using 8 threads, as in ISPD18 Contest. There were problems with the binary of (SUN, 2018) that could not be solved even contacting the authors. Thus, (SUN, 2018) is compared using the data presented in their paper. However, this made the comparison incomplete in some aspects, but not enough to impact in the conclusions of the experiments, as we will see ahead. For the other router's binaries, the results matched the ones in their papers.

Table 11 presents the detailed information of the design rule violation results. "Triton" refers to TritonRoute (KAHNG, 2018), "Sun" refers to (SUN, 2018), "DC" refers to Dr.CU (CHEN, 2019a), "DC2" refers to the last version of Dr.CU (CHEN, 2019b) and "Sm" refers to SmartDR. Table 12 presents the results regarding the area of metal shorts and the total number of violations. "Area Shorts" is the total area of metal shorts in the ISPD18 Area Short Unit (metal2_pitch^2). "Total DRV's" is the total number of DRV's. "DRVR" is the DRV reduction of "Total DRV's" of SmartDR w.r.t. the compared routers. Some columns related to (SUN, 2018) present an approximated result, since they are extracted from (SUN, 2018) and they were presented in scientific notation (i.e. $\times 10^3$). SmartDR presented better DRV count for all benchmarks. From the compared routers, DC2 (CHEN, 2019b) presents the better design rule handling. The average DRV reduction w.r.t DC2 is 82.4%. Table 13 presents the runtime results. "Runtime Reduction" is the runtime reduction of SmartDR w.r.t. the compared routers. SmartDR presented the best runtime of all compared routers, ranging from 94.7% to 44.2% of average runtime reduction.

A direct runtime comparison was not possible with (SUN, 2018), since the machine used in this experiment is different from theirs. However, even if we suppose (SUN, 2018) has better runtime than our router, this is not much of an advantage since the design rule violation count of (SUN, 2018) is far higher than SmartDR (99.7% average DRVR), and a fast router leaving a large number of violations is useless. Also, the runtime of the binary received from the authors of (SUN, 2018) is higher than all routers' runtime.

Regarding the runtime improvement over (KAHNG, 2018) and (CHEN, 2019a, 2019b), it is impossible to precisely know all of its causes, since they depend on the

Table 11: Detailed Design Rule Violation results over (KAHNG, 2018) and (CHEN, 2019b).

Bench	Spacing					Shorts					Min Area				
	Triton	Sun*	DC	DC2	Sm	Triton	Sun*	DC	DC2	Sm	Triton	Sun	DC	DC2	Sm
test1	120	770	122	17	1	4364	100	127	4	0	0	0	0	0	0
test2	1419	6800	1949	73	6	29845	1220	1005	12	0	1	0	0	0	0
test3	1755	8480	2419	161	33	34753	3920	2444	346	0	0	0	0	0	0
test4	3130	45660	11224	1071	204	42024	5970	6914	463	15	54	126	0	6	0
test5	7438	96580	7742	496	75	145826	7040	5466	406	8	118	37	0	10	0
test6	11621	113050	11023	587	228	152388	10380	7988	168	0	188	48	0	21	0
test7	12896	179190	14880	325	300	243375	19800	23141	772	13	270	108	0	38	0
test8	12744	182490	14384	399	351	238519	20940	20641	861	12	240	103	0	20	0
test9	12581	185270	14470	379	238	264230	19250	18830	297	4	260	74	0	28	0
test10	16176	218480	20837	3910	1398	340862	35220	26688	14605	1349	285	55	0	44	0

* Approximated results

Table 12: Area Short and Total Violation Results.

Bench	Area Shorts $\times 10^6$					TOTAL DRV's					DRVR (%)			
	Triton	Sun	DC	DC2	Sm	Triton	Sun*	DC	DC2	Sm	Triton	Sun*	DC	DC2
test1	0.2	0.7	2.5	0.07	0	4484	870	249	21	1	99.98	99.89	99.60	95.2
test2	5.8	27	213	0.2	0	31265	8020	2954	85	6	99.98	99.93	99.80	92.9
test3	241	218	317	59.6	0	36508	12400	4863	507	33	99.91	99.73	99.32	93.5
test4	684	330	1053	17.5	0	45208	51756	18138	1540	219	99.52	99.58	98.79	85.8
test5	72	307	189	3.1	0	153382	103657	13208	912	83	99.95	99.92	99.37	90.9
test6	77	488	516	3.7	0	164197	123478	19011	776	228	99.86	99.82	98.80	70.6
test7	368	838	1322	9.2	0	256541	199098	38021	1135	313	99.88	99.84	99.18	72.4
test8	335	904	894	10	0.001	251503	203533	35025	1280	363	99.86	99.82	98.96	71.6
test9	101	717	693	6.5	0.164	277071	204594	33300	704	242	99.91	99.88	99.27	65.6
test10	538	8861	6028	454.8	1700	357323	253755	47525	18559	2747	99.23	98.92	94.22	85.2
Avg											99.8	99.7	98.7	82.4

* Approximated results

Table 13: Runtime Results.

Bench	Runtime (s)				Runtime Reduction (%)		
	Triton	DC	DC2	Sm	Triton	DC	DC2
test1	154	29	14	16	89.6	44.8	-14.3
test2	1399	169	118	71	94.9	58.0	39.8
test3	2335	196	155	157	93.3	19.9	-1.3
test4	9972	683	450	314	96.9	54.0	30.2
test5	3705	1136	589	208	94.4	81.7	64.7
test6	6124	1695	708	286	95.3	83.1	59.6
test7	10994	3178	1347	429	96.1	86.5	68.2
test8	9793	3104	1413	435	95.6	86.0	69.2
test9	9119	3014	1222	405	95.6	86.6	66.9
test10	16421	3124	1743	717	95.6	77.0	58.9
Avg					94.7	67.8	44.2

implementation of the routers, which are inaccessible, and their techniques, which we have limited knowledge. However, the main reason behind the runtime improvement is probably because SmartDR uses an interval-based path search algorithm. This approach was already shown to be very effective in comparison with the traditional maze routing. In (GESTER, 2013), it is stated that it presents a speedup of at least 6

w.r.t maze routing approach. In (GONÇALVES, 2017), an average speedup of 20 was achieved, in comparison with the classic A* search. Also, the A* heuristic function (TL) that DRAPS uses helps in the runtime improvement. The higher runtime of TritonRoute (KAHNG, 2018) is probably due to the fact they assume a concurrent routing approach and use ILP to solve the routing problem. Regarding Dr.CU (CHEN, 2019b), the local area of the path search (i.e, the area restricted by the global routing guides) is preprocessed and an optimized graph is created, in which the path search is run using a Dijkstra-based algorithm. Running the path search in this optimized graph gives advantage over the traditional maze search approach, but there is a runtime cost to build the graph. Using an interval-based path search, no preprocessing is needed.

In ISPD18 Contest, other routing metrics are also considered, as WL, via count, wrong-way WL (WW), off-track WL (OTW) and vias (OTV), and out-of-guide (i.e. global routing guide) WL (OGW) and vias (OGV). Using ISPD18 Evaluation Script, all of these statistics were obtained. Table 14 and Table 15 present these results. The results of

Table 14: Results of WL, Vias and Out-of-Guide Usage

Bench	WL $\times 10^6$					Vias $\times 10^3$					Out-of-Guide WL $\times 10^5$				
	Triton	Sun	DC	DC2	Sm	Triton	Sun	DC	DC2	Sm	Triton	Sun	DC	DC2	Sm
test1	186	200	174	173	181	39	43	34	32	36	23	4	17	7	46
test2	3239	3330	3127	3123	3227	385	404	339	317	355	256	136	419	137	573
test3	3606	3690	3483	3473	3544	390	398	332	308	340	169	65	707	210	1835
test4	5433	5540	5209	5207	5319	848	823	702	659	816	536	113	1539	266	1965
test5	5841	5900	5570	5546	5849	1143	1073	943	917	1106	557	260	1298	186	1245
test6	7581	7620	7163	7119	7577	1769	1642	1447	1404	1693	818	382	1953	285	1619
test7	13731	13810	13072	12999	13679	2866	2657	2350	2272	2735	1355	908	4376	471	2700
test8	13798	13810	13134	13058	13752	2880	2621	2360	2282	2749	1394	938	4576	581	3022
test9	11651	11660	10999	10921	11596	2873	2622	2359	2282	2744	1242	673	3209	569	2387
test10	14327	14330	13656	13582	14341	3056	2791	2533	2440	2947	1926	1048	5654	2275	6547

Table 15: Results of Wrong-Way WL, Off-Track Usage and Scores.

Ben	Off-Track WL $\times 10^5$					Off-Track Vias $\times 10^3$					Wrong-Way WL $\times 10^5$					Out-of-Guide Vias $\times 10^2$				
	Triton	Sun	DC	DC2	Sm	Triton	Sun	DC	DC2	Sm	Triton	Sun	DC	DC2	Sm	Triton	Sun	DC	DC2	Sm
test1	0.3	0.7	1.1	2	7	0.1	0	0	0	0	0.07	7	9	19	63	13	5	9	4	25
test2	9.9	11.1	17	20	52	1.2	0	0	0	0	0.7	72	88	178	335	127	57	118	59	232
test3	80	7.7	17	23	55	1.1	0	0	0	0	0.8	67	89	182	360	7	60	107	55	234
test4	361	8.7	84	18	389	1.7	0	0	0	0	2.0	112	179	119	653	500	286	314	161	1014
test5	40	32	27	3	44	9.9	0	0	0	1	2.3	216	127	89	153	546	134	431	167	747
test6	62	61	41	17	66	16.4	0	0	0	2	5.5	366	192	140	257	803	218	687	259	1118
test7	187	105	66	33	7196	23.4	0	0	0	76	8.8	604	341	214	372	1314	394	1019	363	1419
test8	177	106	67	34	15778	23.6	0	0	0	79	9.0	615	341	222	378	1351	393	1030	386	1448
test9	104	92	59	25	11521	23.4	0	0	0	77	8.9	603	337	217	365	1272	395	1155	421	1727
test10	476	122	66	61	19557	27.7	0	0	0	125	11.5	717	361	396	480	1381	477	1403	645	2339

SmartDR were better in some cases and worse in the others. However, it is important to note that the compared routers present higher spacing and short violation number, and higher area of metal shorts. These violations imply in higher wire (and spacing radius) overlapping. In order to avoid these overlapping, the router needs to perform more detours. The more detours a router performs, the higher is the: 1) WL; 2) WW, when it detours in the same metal layer; 3) Via count, when detouring using adjacent layers; 4) OGW and OGV, when the detours take the path out of the global routing guide; 5) OTW and OTV, when the router needs to perform off-track paths in order to access the pins without violations. Thus, a router with less DRVs, which is the case of SmartDR, will have higher values on these metrics. Notice that the most important thing is the DRV number, since a circuit cannot be manufactured with DRVs, no matter how good are these metrics.

Comparing to DC2, SmartDR presents more WL and via count. There is an average WL increase of 4.4%, and an average via count increase of 15.6%. Part of the increase in these routing metrics, and possibly the largest, may be explained by the higher DRVs of Dr.CU, but there are some other possible explanations. Dr.CU, uses a net routing approach that is more favorable for these metrics. When routing a net, the resulting paths are used as source components for the next path searches of the same net. This makes the routed net closer to its Steinner tree, reducing WL, detours, and, consequently, the number of vias. In SmartDR, each net is decomposed in two-pin nets, before routing. Thus, all path searches, in Standard Routing step, are performed to connect *pins* only, meaning that the path search is blind to the parts of the net that were already routed. It is still possible to update a two-pin net on-the-fly making it select an already routed wire segment on its tunnel to replace the pins, but this is not trivial, and was not implemented. Another solution is not to decompose the nets, but this may cause some runtime increase, since the number of batches of nets to be routed in parallel will be higher. Also, regarding via count, it is important to note that this increase w.r.t. DC2 is not due to the pin access approach, since the via count increase, in the worst case, was of 2.4% in average w.r.t. a CFS using one via per pin.

Another explanation for the WL increase w.r.t. DC2 is the following. It was empirically observed that DC2 uses patch metals to perform the wire extensions needed to meet the min area rule. However, the ISPD18 Evaluation Script does not count patch metals as WL. In SmartDR, the wire extensions are *not* currently being performed by patch metals, thus, they are counting for WL. Considering that the

Table 16: ISPD18 Score Metrics.

Metrics	Score Weight
Area Shorts/metal2_pitch	500
Number of spacing violations	500
Number of min area violations	500
Out-of-Guide WL	1
Out-of-Guide Vias	1
Off-track WL	0.5
Off-track Vias	1
Wrong-way WL	1
Number of Vias	2
WL	0.5

average wire extension WL is 5% of the total WL of SmartDR, and that it presents 4.4% more WL than DC2, it is even possible that SmartDR has less WL.

The ISPD18 Contest uses a score to measure the effectiveness of a router. This score takes into consideration all of the mentioned metrics. Table 16 presents the ISPD18 score metrics (MANTIK, 2018). Comparing the effectiveness of routers using this score seems reasonable, but it must be clear that it does not represent how good a router is in a precise way, since: 1) The weighted sum used to calculate the score utilizes subjective values, and the “effectiveness of a router” may be subjective or relative according to the circumstances; 2) The runtime impact on score is underestimated. Runtime is very important since detailed routing is highly time-consuming. The ISPD18 Contest also takes into consideration the runtime, but its impact in the score is negligible. The maximum possible runtime benefit is 10% score reduction (the lower the score, the better). In the presented experiments, even with the 44.2% average runtime reduction w.r.t. DC2, this causes about 2-3% score reduction on average. 3) The score calculation has a flaw. It does not penalize short violations, only the area of metal shorts. This way, a short with 0 area has no score penalty. This opens the possibility to solve spacing violations of different nets, by creating patch metals that connect both shapes that create the violation, without having score penalty. This was better discussed in (SUN, 2018).

Table 17 presents the score results (Score $\times 10^4$ column). They do not consider runtime. In (SUN, 2018), it was presented two versions of results, one using patch metals to solve spacing violations, as mentioned earlier, and other without this patch

Table 17: ISPD18 Score Results.

Benchmark	Score $\times 10^4$					Modified Score $\times 10^4$				
	Triton	Sun	DC	DC2	Sm	Triton	Sun*	DC	DC2	Sm
test1	38	35.7	36	29.7	32.9	256	40.7	42.6	29.9	32.9
test2	563	554	637	466	500	2055	615	687	467	500
test3	697	672	743	533	571	2435	868	865	550	571
test4	2583	3708	3411	1530	1654	4684	4007	3757	1554	1655
test5	2163	5100	2281	1614	1766	9454	5452	2554	1635	1766
test6	2901	7339	3391	2120	2351	10520	7858	3790	2128	2351
test7	5065	12466	6382	3772	4337	17234	13456	7539	3811	4338
test8	4959	12578	5850	3799	4592	16885	13625	6882	3842	4593
test9	4181	11786	5001	3259	3910	17392	12748	5943	3274	3911
test10	5629	23402	12814	4791	7246	22672	25163	14149	5521	7313

* Overestimated results

metal insertion. The results of (SUN, 2018) regarding DRV count and solution quality presented before, are without patch metal insertion. The score results of (SUN, 2018) presented in Table 17 are with patch metal insertion, and thus, they are not consistent with their results presented before. Ignoring DC2, SmartDR beats all routers in all benchmarks, except for test10, comparing to TritonRoute. This is because TritonRoute exploits the flaw in the metric mentioned earlier. In order to obtain a fairer and more realistic score, it is also presented a modified score (Modified Score $\times 10^4$ column) which penalizes short violations as any other violation (i.e. 500 score penalty for each short). This was also adopted by the new score calculation of ISPD19 Contest (LIU, 2019). (SUN, 2018) is penalized in the modified score by considering the metal short count of results, which has not patch metal insertion. Thus, the number of metal shorts of the version of results that used patch metal insertion is even higher, making the score of (SUN, 2018) shown in the Modified Score column overestimated (i.e. lower than it actually is). Using the modified score, it is possible to observe the huge difference in the score of TritonRoute.

The best score of all compared routers is of DC2. However, as argued earlier, this does not imply that DC2 is the best router. Comparing routers by this score may be significant only when the scores substantially different, as comparing SmartDR with TritonRoute, for example. However, due to the subjective metrics, this comparison becomes not significant when the compared scores get closer. Also, the runtime is not being considered in the score. Thus, since one of the main objectives of SmartDR is runtime, this comparison is not even fair. The only reason this comparison is being held here is (1) that the benchmarks used are the ones of ISPD18 Contest and this contest has a score to rank the routers, and it seems not acceptable to present results

using these benchmarks without showing the router score, considering that all other routers presented it in their respective works; and (2) although the comparison by this score is limited, it *may* be significative in some cases, as argued before.

While SmartDR focuses on runtime and design rule handling, DC2 is designed for score reduction of ISPD18 Contest. For example, the relaxation of min area violations in the path search of DC2 (CHEN, 2019a, 2019b) is due to the fact that some violations may lead to a better score. The same is true for the high number of short violations of DC2. Even in the simplest test case (test1), DC2 present short violations. Also, in (CHEN, 2019b) it is stated that the number of RNR iterations is limited by 4, since more iterations lead to a slight score improvement at the expense of high runtime. All of this show that the goal of DC2 is the score reduction, which is not much realistic. By aiming at runtime and design rule handling, and presenting considerably better results in these goals, SmartDR is closer than DC2 to present a manufacturable solution in practice.

8 Conclusions

This work proposed an initial detailed router (SmartDR) addressing two important routing metrics: runtime and design rule handling. The main contributions to attend these objectives are in the pin access and path search techniques.

Regarding pin access, it was proposed a new approach enabling resource sharing among PAPs, with dynamic legalization and implementation. The main idea of the method is to provide as many DRV-free PAPs as possible to be chosen by the path search so that routability is not penalized as it happens when statically implementing few PAPs. Part of the high PAP diversity is due to the use of patch metals to solve common pin access DRVs. Since these patch metals greatly contribute to TMS creation, it was also proposed an algorithm to efficiently detect TMSs. The proposed pin access approach, along with the TMS detection algorithm, is novel in the literature.

As for the path search techniques, the proposed path search algorithm unites some existing techniques, with slight changes, and integrates new ones, resulting in a new path search algorithm. It is an A*-interval-based path search which integrates design rule handling and is adapted to properly handle costs in the source and target points. The source/target cost handling complements the pin access method by considering the costs of PAPs and avoiding non-necessary multiple PAP selection, which can increase in via count. The main novelty of the design rule handling feature of the proposed algorithm is the efficient via checks with the proposed via map. As for the A* heuristic function h , it was also proposed a new technique to preprocess the tunnels in order to provide better lowerbounds to h during the path search.

SmartDR and all of the proposed techniques were evaluated using the ISPD18 Contest benchmarks, which are derived from real industrial designs. Regarding the proposed pin access method, the experiments have shown that it provides better runtime and routability results when compared to the common approach of statically assigning PAPs to specific locations. The WL was slightly improved and the via count presented slight oscillations. The path search awareness of the PAP costs provided a considerable mitigation of the via count increase brought by the high PAP diversity. The design rule handling in the path search was shown to be essential to reduce the number of design rule violations. Also, the proposed path search algorithm is able to reduce a huge number of design rule violations without compromising runtime. The experiments also showed that the proposed tunnel lowerbound method provides a considerable speedup to the path search. The method was also shown to be more

efficient than the only other proposed method in the literature (PEYER, 2009), since its preprocessing time is negligible, while the preprocessing in (PEYER, 2009) was heavy such that it actually worsens the overall time rather than decreases it. Considering the comparison of SmartDR with other state-of-the-art routers, the experiments have shown that it presents substantially superior results in both runtime and design rule handling, which are the main goals of the proposed router.

References

- AHRENS, M. Detailed Routing Algorithms for Advanced Technology Nodes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 34, n. 4, p. 563 - 576, 2015.
- BATTERYWALA, S. *Track assignment: a desirable intermediate step between global routing and detailed routing*. ICCAD '02 Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design. San Jose: [s.n.]. 2002. p. 59-66.
- CHANG, C.-C. Pseudopin assignment with crosstalk noise control. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 20, n. 5, p. 598 - 611, 2001.
- CHANG, F.-Y. MANA: A Shortest Path Maze Algorithm Under Separation and Minimum Length NANometer Rules. *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, v. 32, n. 10, p. 1557 - 1568, September 2013.
- CHAZELLE, B. Computing the Largest Empty Rectangle. *SIAM Journal on Computing*, v. 15, n. 1, p. 300–315, 1986.
- CHEN, G. *Detailed Routing by Sparse Grid Graph and Minimum-Area-Captured Path Search*. ASPDAC '19 Proceedings of the 24th Asia and South Pacific Design Automation Conference. Tokyo: [s.n.]. 2019a. p. 754-760.
- CHEN, G. Dr. CU: Detailed Routing by Sparse Grid Graph and Minimum-Area-Captured Path Search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, July 2019b.
- CHEN, H. M. *Integrated floorplanning and interconnect planning*. Proc. IEEE/ACM Int. Conf. on computer-Aided Design. [S.l.]: [s.n.]. 1999. p. 354–357.
- CHEN, H.-Y. Global and Detailed Routing. In: *Electronic Design Automation: Synthesis, Verification, and Test*. [S.l.]: [s.n.], 2009. p. 687-749.
- CHO, M. *Double patterning technology friendly detailed routing*. International Conference on Computer-Aided Design (ICCAD'08). San Jose: [s.n.]. 2008.
- CHU, C. FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 27, 2008. 70–83.
- CONG, J. *General models and algorithms for over-the-cell routing in standard cell design*. DAC '90 Proceedings of the 27th ACM/IEEE Design Automation Conference. Orlando: [s.n.]. 1990. p. 709-715.
- CONG, J. Over-the-cell channel routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 9, n. 4, p. 408 - 418, 1990.

CONG, J. DUNE-a multilayer gridless routing system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 20, n. 5, p. 633 - 647, 2001.

CONG, J. MARS-a multilevel full-chip gridless routing system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 24, n. 3, p. 382 - 394, 2005.

CORMEN, T. The algorithms of Kruskal and Prim. In: *Introduction to Algorithms*. Cambridge: The MIT Press, 2002.

CORMEN, T. H. Section 24.3: Dijkstra's algorithm. In: *Introduction to Algorithms*. 2^a. ed. Cambridge: MIT Press, 2001. p. 595–601.

DEUTSCH, D. N. A "DOGLE" CHANNEL ROUTER. DAC '76 Proceedings of the 13th Design Automation Conference. San Fransico: [s.n.]. 1976. p. 425-433.

DING, Y. Self-Aligned Double Patterning Lithography Aware Detailed Routing With Color Preassignment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 36, n. 8, p. 1381 - 1394, Aug. 2017.

DING, Y. Self-Aligned Double Patterning-Aware Detailed Routing With Double Via Insertion and Via Manufacturability Consideration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, March 2018. 657-668.

DU, Y. *Spacer-is-dielectric-compliant detailed routing for self-aligned double patterning lithography*. 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC). [S.l.]: [s.n.]. 2013.

EDELSBRUNNER, H. A new approach to rectangle intersections. *International Journal of Computer Mathematics*, v. 13, n. 3-4, p. 209-219, 1983.

GAO, X. *Enhancing double-patterning detailed routing with lazy coloring and within-path conflict avoidance*. DATE '10 Proceedings of the Conference on Design, Automation and Test in Europe. Dresden: [s.n.]. 2010.

GESTER, M. BonnRoute: Algorithms and data structures for fast and good. *ACM Transactions on Design Automation of Electronic Systems*, 18, n. 2, March 2013. 1-24.

GONÇALVES, S. M. M. *A survey of path search algorithms for VLSI detailed routing*. Symposium on Circuits and Systems (ISCAS), 2017 IEEE International. Baltimore: [s.n.]. 2017.

GONÇALVES, S. M. M. *An Improved Heuristic Function for A*-Based Path Search in Detailed Routing*. Symposium on Circuits and Systems (ISCAS), 2019 IEEE International. Sapporo: [s.n.]. 2019a.

GONÇALVES, S. M. M. DRAPS: A Design Rule Aware Path Search Algorithm for Detailed Routing. *IEEE Transactions on Circuits and Systems II: Express Briefs*, [early access], 2019b.

HAMACHI, G. T. *A switch-box router with obstacle avoidance*. DAC '84 Proceedings of the 21st Design Automation Conference. Albuquerque: [s.n.]. 1984. p. 173-179.

HANAN, M. On Steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 1966. 255–265.

HART, P. E. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, v. 4, n. 2, p. 100-107, 1968.

HASHIMOTO, A. *Wire routing by optimizing channel assignment within large apertures*. DAC '71 Proceedings of the 8th Design Automation Workshop. Atlantic City: [s.n.]. 1971. p. 155-169.

HETZEL, A. *A Sequential Detailed Router for Huge Grid Graphs*. Design, Automation and Test in Europe. Paris: [s.n.]. 1998. p. 332-338.

HIGHTOWER, D. *A solution to line routing problems on the continuous plane*. Proceedings of DAC '69 Proceedings of the 6th annual Design Automation Conference. New York: [s.n.]. 1969. p. 1-24.

HO, T.-Y. *A fast crosstalk- and performance-driven multilevel routing system*. ICCAD '03 Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design. Washington: [s.n.]. 2003. p. 382.

HOLMES, N. D. *Algorithms for three-layer over-the-cell channel routing*. Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on. Santa Clara: [s.n.]. 1991.

IGUSA, M. *ORCA: A sea-of-gates place and route system*. Design Automation, 1989. 26th Conference on. Las Vegas: [s.n.]. 1989.

JIA, X. A Multi-Commodity Flow based Detailed Router with Efficient Acceleration Techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. PP, n. 99, p. 217-230, 2017.

JONKER, R. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, v. 38, n. 4, p. 325–340, 1987.

JOOBANI, R. *WEAVER: a knowledge-based routing expert*. *IEEE Design & Test of Computers*, v. 3, n. 1, p. 12 - 23, 1986.

KAHNG, A. B. Negotiated Congestion Routing. In: KAHNG, A. B. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. [S.l.]: Springer, 2011d. p. 162-163.

KAHNG, A. B. Rip-Up and Reroute (RRR). In: KAHNG, A. B. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. [S.l.]: Springer, 2011c. p. 158-160.

KAHNG, A. B. VLSI Desing Styles. In: KAHNG, A. B. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. [S.l.]: Springer, 2011b. p. 158-160.

KAHNG, A. B. VLSI Desing Styles. In: KAHNG, A. B. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. [S.l.]: Springer, 2011a. p. 11-16.

KAHNG, A. B. *TritonRoute: An Initial Detailed Router for Advanced VLSI Technologies*. ICCAD '18 Proceedings of the International Conference on Computer-Aided Design. San Diego: [s.n.]. 2018.

KAO, W.-C. Cross point assignment with global rerouting for general-architecture designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 14, n. 3, p. 337 - 348, 1995.

KASTNER, R. Pattern routing: use and theory for increasing predictability and avoiding coupling. *IEEE Trans. on Computer-Aided Design*, 2002. 777–790.

KATO, I. *A method of pattern data management of PWB layout system*. Proceedings of the 35th Annual Convention IPS Japan. Japan: [s.n.]. 1987.

KUH, E. Recent advances in VLSI layout. *Proceedings of the IEEE*, v. 78, n. 2, p. 237 - 263, 1990.

LEE, C. Y. An Algorithm for Path Connections and Its Applications. *IRE Transactions on Electronic Computers*, v. 10, n. 3, p. 346–365, September 1961.

LEI, S.-I. *Double patterning-aware detailed routing with mask usage balancing*. Fifteenth International Symposium on Quality Electronic Design. [S.l.]: IEEE. 2014.

LI, Y.-L. NEMO A new implicit connection graph-based gridless router with multi-layer planes and pseudo-tile propagation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 26, n. 4, p. 705 - 718, 2007.

LI, Y.-L. A gridless routing system with nonslicing floorplanning-based crosstalk reduction on gridless track assignment. *ACM Transactions on Design Automation of Electronic Systems*, v. 16, n. 2, 2011.

LIN, S.-P. *A novel framework for multilevel routing considering routability and performance*. Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on. San Jose: [s.n.]. 2002.

LIN, Y.-H. *Double patterning lithography aware gridless detailed routing with innovative conflict graph*. DAC '10 Proceedings of the 47th Design Automation Conference. Anaheim: ACM New York. 2010. p. 398-403.

LIN, Y.-H. *TRIAD: a triple patterning lithography aware detailed router*. ICCAD '12 Proceedings of the International Conference on Computer-Aided Design. San Jose: ACM New York. 2012. p. 123-129.

LIU, I.-J. Overlay-aware detailed routing for self-aligned double patterning lithography using the cut process. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35, n. 9, Sept. 2016. 1519 - 1531.

LIU, L. C. *Chip-level area routing*. Proceedings of the International Symposium on Physical Design. Monterey: [s.n.]. 1998. p. 197–204.

LIU, W.-H. *ISPD 2019 Initial Detailed Routing Contest and Benchmark with Advanced Routing Rules*. ISPD '19 Proceedings of the 2019 International Symposium on Physical Design. San Francisco: [s.n.]. 2019.

LUK, W. K. A greedy switch-box router. *Integration, the VLSI Journal*, v. 3, n. 2, p. 129-149, 1985.

MA, Q. *Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology*. DAC '12 Proceedings of the 49th Annual Design Automation Conference. San Francisco: [s.n.]. 2012.

MANTIK, S. *ISPD 2018 Initial Detailed Routing Contest and Benchmarks*. ISPD '18 Proceedings of the 2018 International Symposium on Physical Design. Monterey: [s.n.]. 2018. p. 140-143.

MAREK-SADOWSKA, M. *Two-dimensional router for double layer layout*. Design Automation, 1985. 22nd Conference on. Las Vegas: [s.n.]. 1985.

MARGARINO, A. A tile-expansion router. *IEEE Transactions on Computer-Aided Design*, v. 6, n. 4, p. 507–517, 1987.

MIKAMI, K. *A computer program for optimal routing of printed circuit connectors*. Proceedings of the International Federation for Informatics. [S.l.]: [s.n.]. 1968. p. 1475–1478.

MIRSAEEDI, M. *Self-aligned double-patterning (SADP) friendly detailed routing*. Design for Manufacturability through Design-Process Integration V. San Jose: [s.n.]. 2011.

NATARAJAN, S. *Over-the-cell channel routing for high performance circuits*. DAC '92 Proceedings of the 29th ACM/IEEE Design Automation Conference. Anaheim: [s.n.]. 1992.

NIEBERG, T. *Gridless pin access in detailed routing*. DAC '11 Proceedings of the 48th Design Automation Conference. San Diego: ACM New York. 2011.

OHTSUKI, T. *Gridless routers—New wire routing algorithms based on computational geometry*. Proceedings of the International Conference on Circuits and Systems. [S.l.]: [s.n.]. 1985.

OUSTERHOUT, J. K. Corner stitching: A data-structuring technique for VLSI layout tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 3, n. 1, p. 87 - 100, 1984.

OZDAL, M. M. Detailed-routing algorithms for dense pin clusters in integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 28, n. 3, p. 340 - 349, 2009.

PARNG, T. M. *A new approach to sea-of-gates global routing*. Computer-Aided Design, 1989. ICCAD-89. Digest of Technical Papers., 1989 IEEE International Conference on. Santa Clara: [s.n.]. 1989.

PEYER, S. A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing. *Journal of Discrete Algorithms*, v. 7, n. 4, p. 377-390, 2009.

RUBIN, F. The Lee Path Connection Algorithm. *IEEE Transactions on Computers*, v. C-23, n. 9, p. 907 - 914, 1974.

SATO, M. A fast line-search method based on a tile plane. *IEEE International Symposium on Circuits and Systems*, v. 5, p. 588-591, 1987.

SCHULTE, C. *Design Rules in VLSI Routing*. Bonn: [s.n.], 2012.

SHERWANI, N. Steiner Tree Algorithms. In: SHERWANI, N. *Algorithms for VLSI Design Automation*. 3 ed. ed. New York: [s.n.], 1998. p. 111-115.

SHIN, H. A Detailed Router Based on Incremental Routing Modifications: Mighty. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 6, n. 6, p. 942 - 955, 1987.

SUN, F.-K. *A multithreaded initial detailed routing algorithm considering global routing guides*. ICCAD '18 Proceedings of the International Conference on Computer-Aided Design. San Diego: [s.n.]. 2018.

WU, B. *Over-the-cell routers for new cell model*. DAC '92 Proceedings of the 29th ACM/IEEE Design Automation Conference. Anaheim: [s.n.]. 1992. p. 604-607.

XU, X. Self-Aligned Double Patterning Aware Pin Access and Standard Cell Layout Co-Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 34, n. 5, p. 699 - 712, May 2015.

XU, X. PARR: Pin-Access Planning and Regular Routing for Self-Aligned Double Patterning. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 21, n. 3, Jul 2016.

XU, X. *Concurrent Pin Access Optimization for Unidirectional Routing*. DAC '17 Proceedings of the 54th Annual Design Automation Conference 2017. Austin: [s.n.]. 2017.

XU, Y. *FastRoute 4.0: global router with efficient via minimization*. ASP-DAC '09 Proceedings of the 2009 Asia and South Pacific Design Automation Conference. Yokohama: [s.n.]. 2009.

YU, H.-J. *DSA-Friendly Detailed Routing Considering Double Patterning and DSA Template Assignments*. 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). San Francisco: [s.n.]. Aug. 2018. p. 1381 - 1394.

YUAN, . *Double patterning lithography friendly detailed routing with redundant via consideration*. DAC '09 Proceedings of the 46th Annual Design Automation Conference. San Francisco: ACM New York. 2009.

ZHANG, Y. *GDRouter: Interleaved global routing and detailed routing for ultimate routability*. Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE. San Francisco: [s.n.]. 2012.

ZHANG, Y. *RegularRoute: An Efficient Detailed Router Applying Regular Routing Patterns*. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 21, n. 9, p. 1655 - 1668, 2013.