UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

ANDRÉ SALDANHA OLIVEIRA

Initial Detailed Routing Algorithms

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Microeletronics

Advisor: Prof. Dr. Ricardo Reis

Porto Alegre
March 2021

CIP — CATALOGING-IN-PUBLICATION

Oliveira, André Saldanha

Initial Detailed Routing Algorithms / André Saldanha Oliveira. – Porto Alegre: PGMICRO da UFRGS, 2021.

110 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2021. Advisor: Ricardo Reis.

I. Reis, Ricardo. II. Título.

AGRADECIMENTOS

Agradeço aos meus pais pelo amor e apoio incondicional.

ABSTRACT

In this work, we present a study of the problem of routing in the context of the VLSI physical synthesis flow. We study the fundamental routing algorithms such as maze routing, A*, and Steiner tree-based algorithms, as well as some global routing algorithms, namely FastRoute 4.0 and BoxRouter 2.0. We dissect some of the major state of the art initial detailed routing tools, such as RegularRoute, TritonRoute, SmartDR and Dr.CU 2.0.

We also propose an initial detailed routing flow, and present an implementation of the proposed routing flow, with a track assignment technique that models the problem as an instance of the maximum independent weighted set (MWIS) and utilizes integer linear programming (ILP) as a solver. The implementation of the proposed initial detailed routing flow also includes an implementation of multiple-source and multiple-target A* for terminal andnet connection with adjustable rules and weights.

Finally, we also present a study of the results obtained by the implementation of the proposed initial detailed routing flow and a comparison with the ISPD 2019 contest winners, considering the ISPD 2019 and benchmark suite and evaluation tools.

Keywords: Microelectronics, Electronic Design Automation, Physical Design, Initial Detailed Routing.

RESUMO

Neste trabalho, apresentamos um estudo do problema de roteamento no contexto do fluxo de síntese física de circuitos integrados VLSI. Nós estudamos algoritmos de roteamento fundamentais como roteamento de labirinto, A* e baseados em árvores de Steiner, além de alguns algoritmos de roteamento global como FastRoute 4.0 e BoxRouter 2.0. Nós dissecamos alguns dos principais trabalhos de roteamento detalhado inicial do estado da arte, como RegularRoute, TritonRoute, SmartDR e Dr.CU 2.0.

Também propomos um fluxo de roteamento detalhado inicial, e apresentamos uma implementação do fluxo de roteametno proposto, com uma técnica de assinalamento de trilhas que modela o problema como uma instância do problema do conjunto independente de peso máximo e usa programação linear inteira como um resolvedor. A implementação do fluxo de rotemaento detalhado inicial proposto também inclui uma implementação de um A* com múltiplas fontes e múltiplos destinos para conexão de terminais e redes, com regras e pesos ajustáveis.

Por fim, nós apresentamos um estudo dos resultados obtidos pela implementação do fluxo de roteamento detalhado inicial proposto e comparamos com os vencedores do ISPD 2019 contest considerando a suíte de teste e ferramentas de avaliação do ISPD 2019.

Palavras-chave: Microeletrônica, EDA, Síntese Física, Rotemento Detalhado Inicial

LIST OF ABBREVIATIONS AND ACRONYMS

EDA	Electronic Design Automation
MWIS	Maximum/Minimum Weighted Independent Set
VLSI	Very Large Scale Integration
G-CELL	Global Cell or Global Routing Grid Cells
MST	Minimum Steiner Tree
RMST	Rectilinear Minimum Steiner Tree
STVT	Single Trunk Vertical Tree
LUT	Lookup Table
ILP	Integer Linear Programming
MCF	Multicommodity Flow

CONTENTS

LIST OF FIGURES

LIST OF TABLES

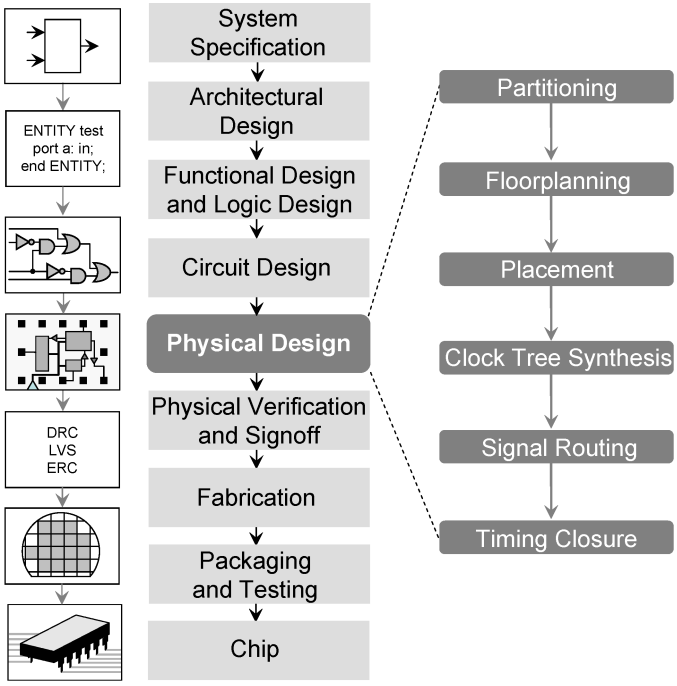
1 INTRODUCTION

Initial detailed routing has an important set of objectives to meet. To better understand these goals and the available information for this class of algorithms we have resorted to a snippet of the VLSI flow, shown in figure 1.1. The initial detailed routing is within the physical design scope, during which the tools allocate the circuit components to a position and connects the devices using electronic structures such as wires and vias. More specifically, it is a sub-step of the Signal Routing step shown in the figure, and the initial detailed routing follows the global routing. Each step in the flow has its domain of problems and objectives. Generally, one step in the flow assumes that the previous steps have produced a roughly optimized result, which means that each step has the challenge of addressing its problems while minimizing disturbance of the previous results. The signal routing step generally divides itself in its own set of steps. Traditionally, the first step is called global routing, where the algorithms create a coarse routing for the nets. This coarser routing contains only the general topology of the net. Following the global routing, the initial detailed routing attempts to create the actual routing with electrical structures according to the set of rules. After an initial routing is available, the flow can apply other algorithms to optimize a certain metric. One example is timing driven detailed routing, where an algorithm pays closer attention to critical nets, or manufacturability driven detailed routing where the algorithm attempts to increase yield. It is important to understand where initial detailed routing is in the flow because the global routing has, hopefully, optimized its result to a certain set of metrics and constraints. Given the global routing result and information propagated through the flow, initial detailed routing has to fulfill its role without disregarding what the flow has already accomplished.

The initial detailed routing is slightly different than the general detailed routing term. The difference is that the initial detailed routing tries to obtain a routing solution considering only a subset of the design rules, or a modified version with relaxed constraints. The detailed routing can start with an initial detailed routing stage and then consider the full design rules later in the flow.

The initial detailed routing is usually the first routing step that attempts to satisfy physical, technological and design constraints. It has access to all the results generated by the flow, the most important being cells and pins positions and the solution of global routing. Traditionally, the information generated by the global router is the set of routing guides, but potentially layer assignment, congestion maps, and track assignment.

Figure 1.1: VLSI flow highlighting physical design steps (KAHNG et al., 2011).



Source: (KAHNG et al., 2011)

The introduction is divided into Preliminaries (Section 1.1), problem formulation (Section 1.2), global routing (Section 1.3), congestion map (Section 1.4), and research on routing at UFRGS (Section 1.5). The remainder of this work is divided into fundamental algorithms (Chapter 2), detailed routing algorithms (Chapter 3), mixed routing frameworks (Chapter 4), proposed initial detailed routing flow (Chapter 5), implementation of the proposed initial detailed routing flow (Chapter 6), and conclusions and future work (Chapter 7).

1.1 Preliminaries

Because the initial detailed routing situates in a flow, it inherits information from the previous steps. There are many fundamental concepts required to understand the context, which we explain below.

- Routing Region: A stack of rectangular metal layers that contains all elements relevant to routing. Figure 1.2 (a) shows a routing region containing a stack of five metal layers.
- Metal Layer: A rectangular area with edges aligned to the x- and y-axis. It is hierarchically a grid-like structure. Figure 1.2 (b) shows a metal layer with some of

the structures usually present. Each layer has a preferred direction that wires ideally would span through. Adjacent layers have orthogonal preferred directions.

- Cell: A logic device with a set position in the routing region. Figure 1.2 (c) shows a group of cells placed in the routing region. The cells have a unique label shown in the upper left corner of its boundary.
- Pin: A metal polygon that interfaces the cell. Figure 1.2 (b) shows multiple pins, one of which appears labeled as such. Also, in figure 1.2 (c) multiple cells have their pins' polygons contour in blue.
- Via: A routing structure that connects two adjacent metal layers. Figure 1.2 (a) features a via connecting the two bottom metal layers.
- Wire: A routing structure that electrically connects any other structures that make contact with its boundaries.
- Tracks: The horizontal and vertical lines that divide the metal layers in a grid. Figure 1.2 (b) shows a metal layer divided into a grid. All lines of the grid are tracks.
- G-Cell: A coarser division of the metal layer into bins. Figure 1.2 (b) has a slightly coarser grid made of thicker divisions that represent the G-Cells, one of which we have highlighted in blue and labeled.
- Routing Guides: Rectangles consisting of G-Cells generated by the global routing as an approximated routing solution.

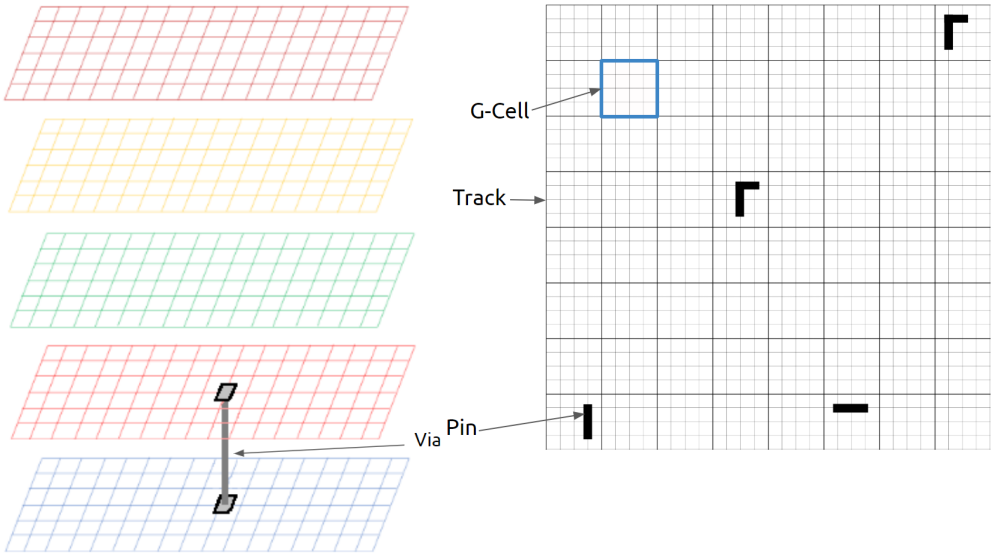
1.2 Problem Formulation

Given a netlist, a routing region, routing guides, and the design rules the initial detailed routing has to define the layout of each net using wires and vias. Note that contacts that connect diffusion and poly silicon are not considered for detailed routing. The main objective is to complete the routing of every net in the design, even if it requires breaking design rules. As secondary objectives, the algorithm should respect the design rules and routing guides as much as possible. The design rules depend on the technology used. There are different priorities for each of these violations. The major secondary metrics are:

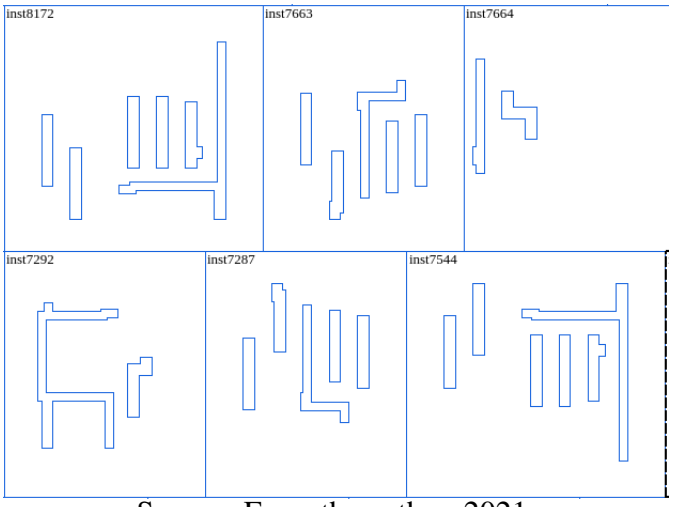
- Connectivity Rules:

Figure 1.2: Routing relevant concepts and structures. In (a) the figure shows a stack of metal layers, with a via connecting the two bottom metal layers. In (b), the figure shows a routing grid with tracks, G-Cells and a few pins. In (c), the figure shows a snippet of a design with some placed cells with their pins shown.

(a) (b)



(c)



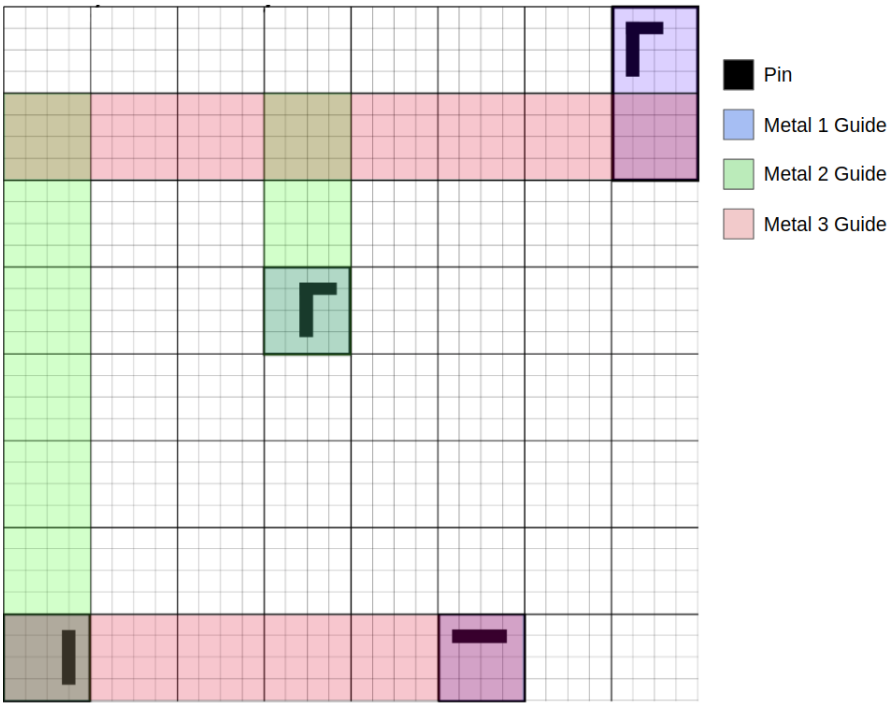
Source: From the author, 2021.

    - Open Net: When a net has at least one pin not attached to the routing structure, the net is considered open.
    - Short: A short is caused when a wire or via overlaps with another wire, pin or via that does not belong to its net or a blockage.

- Layout Rules:
    - Spacing: The layout shapes must respect a minimum distance from each other. In some cases, the minimum distance depends on the specific topology of the shapes.
        - Parallel Run Length: This is a special case of spacing rules. Two wires that are parallel to each other must have a variable minimum distance between them. The minimum distance increases the more length of the two wires run in parallel.
        - Corner-to-Corner Spacing: The layout shapes must maintain a distance between a convex corner and any edges.
        - End of Line: All wires' extremities that are close to another orthogonal polygon must respect a minimum distance called End of Line.
        - Via Cut Spacing: The distance between two vias is in general not the same that is for wires.
        - Adjacent Via Cut Spacing: If a specific via has more than one cut, that is, redundant vias, the distance between the multi-cut via and other vias is different. In general, it is greater than normal via spacing.
    - Minimum Area: All polygons in the layout must have a minimum specified area.
- Routing Preference:
    - Routing Guide Honoring: It is desirable that all wires and vias are within the routing guides.
    - Wrong-way Routing: The algorithm should avoid spanning wires in the non-preferred direction.
    - Off-track Routing: Wires and vias should be aligned to the tracks.

To summarize, figure 1.3 shows a complete set of inputs: a routing region made out of three layers, the routing tracks, pin shapes, and guides. In this example, all pins

Figure 1.3: Example instance of a detailed routing input, with the routing area, a net consisting of four pins and the routing guides.



Source: From the author, 2021.

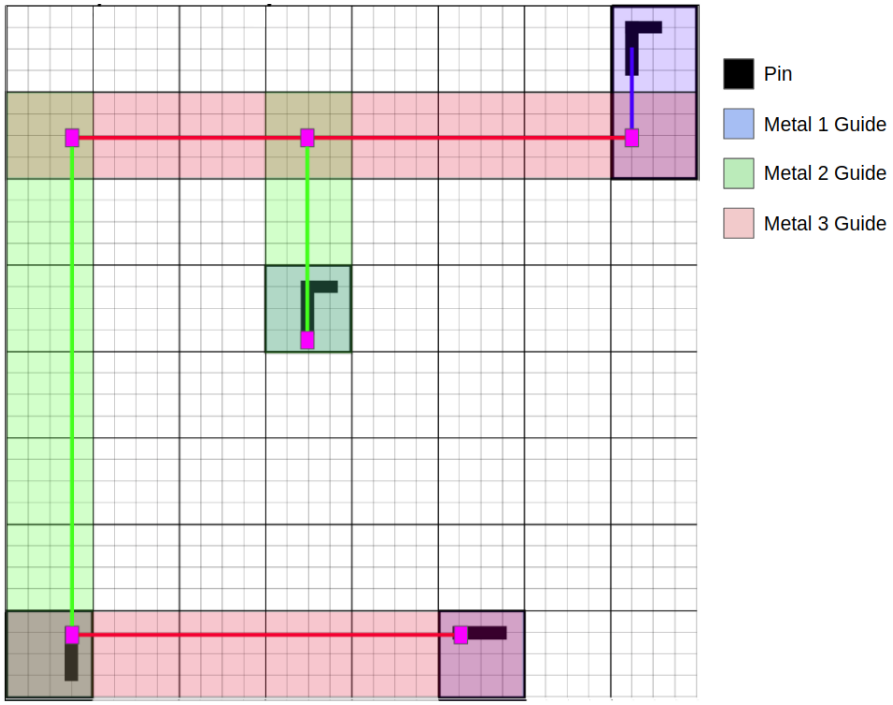
belong to the same net. Figure 1.4 shows one of the possible initial detailed routing solutions for this problem instance. Notice how every wire and via is within its guide, in the preferred direction and aligned to a track. Furthermore, there are no spacing, minimum area or end of line problems, since there isn't any wire that could cause these problems.

1.3 Global Routing

Initial detailed routing has a strong connection to global routing. The result of global routing contains both the desired topology of the net and, in cases of layer assigned global routers, the minimum number of vias. Although the focus of this work is on initial detailed routing, it is important to understand the general strategies employed by the global routers and how they work.

The problem of global routing is to define a coarse assignment of routes to routing regions. More specifically, assigning a set of G-Cells to each net. When performing the assignment, the global router can try to optimize different metrics. Given the placement solution, the router usually divides the region in G-Cells and assign pins to the G-Cells that contain them. After this initialization step, the router defines paths in the G-Cell grid for every net. Commonly used algorithms for the paths are Steiner trees and pathfinding

Figure 1.4: Example solution for an instance of a detailed routing problem. Note that every wire respects the routing guides.

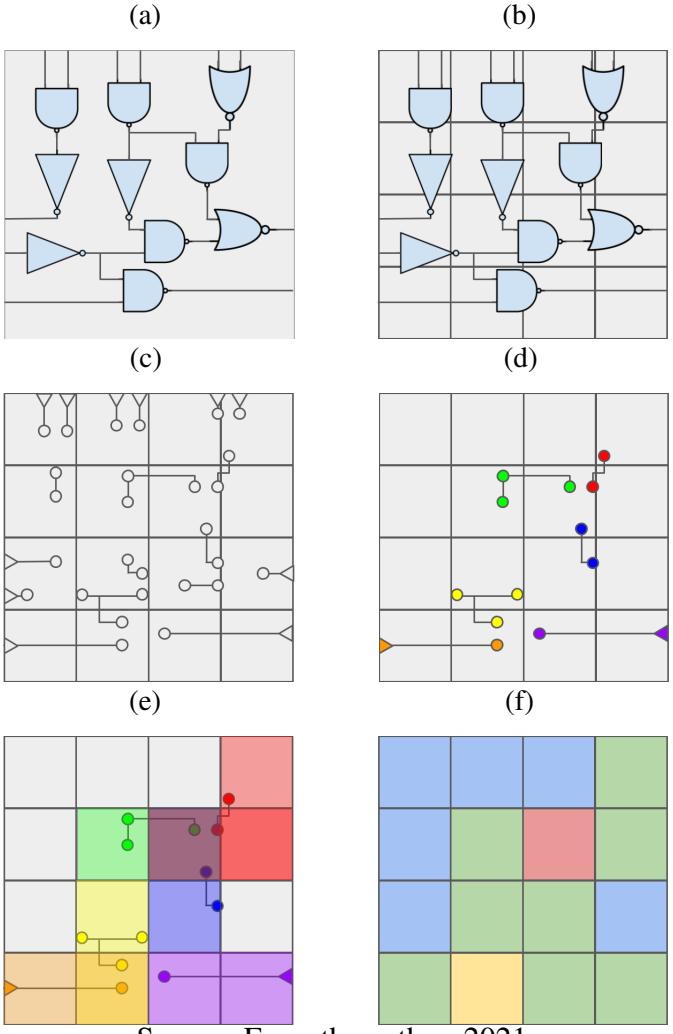


Source: From the author, 2021.

algorithms. It is not uncommon to find global routing algorithms that explicitly work with initial detailed routing in mind. These algorithms apply techniques to make routing more manageable, such as congestion avoidance and layer assignment.

In figure 1.5 there is an example of a simplified global routing instance. In (a) there are cells placed in a bounded region and connected according to the netlist. Note that the cells appear drawn using their symbols for simplicity. Some cells take inputs from the pads. In (b), the image is displaying the same scenario but with the routing region split into a 4x4 grid of G-Cells. In (c) we have hidden the cell symbol so that only the pins and their connections appear. The structures left are the nets. Notice that the triangles represent the pads, and the circles represent the cell pins, but there is no distinction between driver and sink pins for simplicity. In (d), the local nets - nets contained entirely within a G-Cell - do not appear. Also, the pins follow a color scheme and pins of the same net are in the same color. The global router is interested majorly in the connections that span between G-Cells. When estimating congestion or usage, considering local nets could be helpful, but, in practice, they are ignored in several cases (ALPERT et al., 2010). In (e) there is one possible solution for the global routing. Notice that there is a G-Cell that overlaps with the green, blue and red guides, while some G-Cells contain one, two or no guides. In general, attempting detailed routing within a congested G-Cell or more is more complicated. To

Figure 1.5: Simplified global routing example. In (a) simplified cells are placed and connected in a routing region. (b) shows the routing region's division into G-Cells. In (c) we show only the pins and connections. In (d) the local nets are hidden. In (e) the global routing outputs the guides. Finally, in (f) there is a simple congestion map labeling the areas with colors according to congestion.

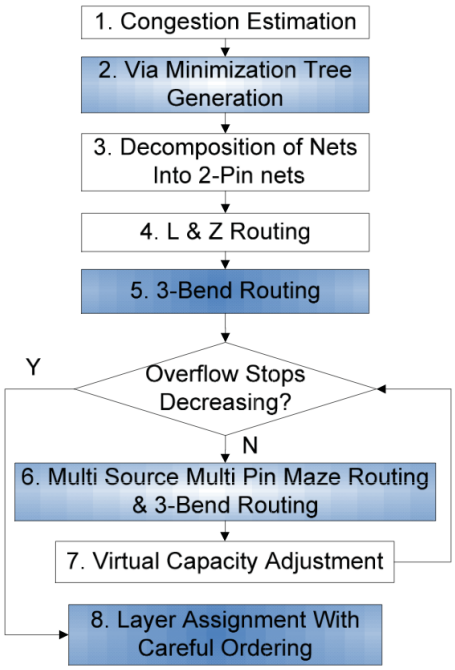


Source: From the author, 2021.

estimate the difficulty of routing one can build a congestion map. The picture in (f) labels the G-Cells with colors according to the congestion. G-Cells with three guide overlaps are red, with two guide overlaps yellow, with one guide overlap green and with no guide overlaps blue.

In the following subsections, we will study a set of global routing frameworks. The emphasis of the study will be on techniques employed to improve the solution for the initial detailed routing.

Figure 1.6: FastRoute 4.0 Framework flowchart. (XU; ZHANG; CHU, 2009)



Source: (XU; ZHANG; CHU, 2009)

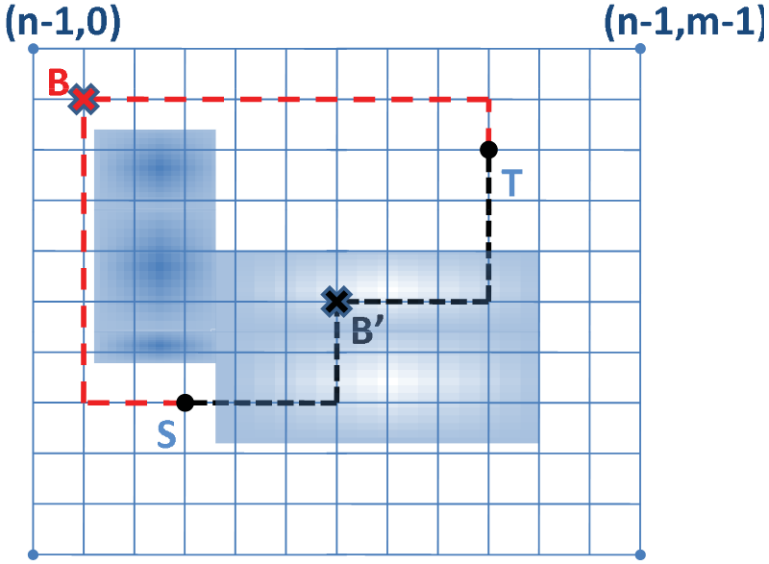
1.3.1 FastRoute 4.0 (XU; ZHANG; CHU, 2009)

In their work, the authors expand on the previous iterations of their global routing framework FastRoute (XU; ZHANG; CHU, 2009). The routing flow employed by FastRoute is in figure 1.6, with the steps introduced in version 4.0 highlighted in blue. The main objective of this specific iteration is to address the via count. In step 1 of the flow, the algorithm creates an estimation of the congestion using an initial global routing solution. This solution is created using FLUTE (CHU; WONG, 2008). Then, in step 2, the topology of the tree is changed to account for the number of vias.

Steps 3, 4 and 5 try to solve the congestion problem by dividing the net into two terminal connections and applying pattern routing and 3-bend routing. The authors state that 3-bend router is faster than the maze router and monotonic router because the search space is limited to three bends per connection. The limit to the number of bends also limits the number of vias to three - not accounting for pin access vias. Furthermore, the authors also state that 3-bend router is better than pattern router in congestion avoidance. Figure 1.7 shows an instance of a net with a source terminal S and a target terminal T. There is a congested region in blue between S and T.

There are two 3-bend paths in this figure - namely $S \rightarrow B \rightarrow T$ and $S \rightarrow B' \rightarrow T$. The path that goes through B avoids the congested area. The authors state that avoiding

Figure 1.7: FastRoute 4.0 3-bend routing technique (XU; ZHANG; CHU, 2009).



Source: (XU; ZHANG; CHU, 2009)

this congested area is impossible with Z-, L- and U-shaped pattern router.

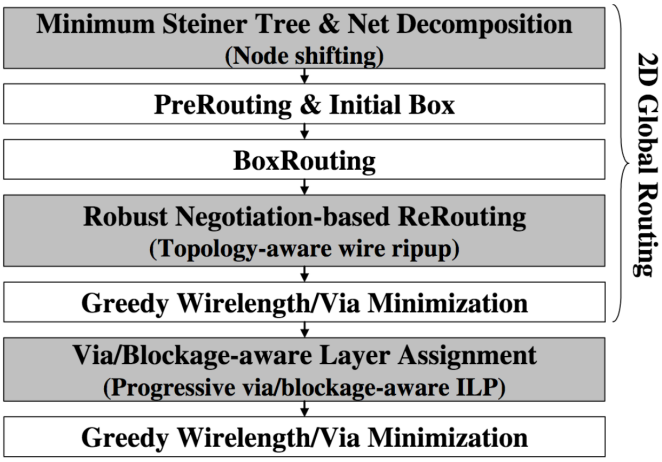
The flow proceeds to a loop that iterates through until the overflow - that is, more connections between two G-Cells than there are resources - stops decreasing. Step 6 tries to rip-up and reroute individual nets with maze router and 3-bend router to reduce the congestion. When the overflow stops decreasing, the final step is called layer assignment with careful ordering takes place. The authors cleverly show that the layer assignment is strongly related to the number of vias. Each time a path changes layers, it implies in the insertion of vias. The technique employed orders the nets based on total wire length and number of pins and assigns the smaller nets to the lower metal layers while assigning the longer segments to higher layers. To choose the layer sequence for a path, they describe a construction technique for a graph where each vertex is a segment, and an edge represents a connection between the two segments. The vertices adjacent to pins are given low metal layers and, using dynamic programming, the algorithm attempts to assign layers to each segment while minimizing the total number of vias.

Table 1.1 shows the results obtained by the new flow including via reduction techniques. The results show that the improved flow allows for a reduction of 11% of the number of vias when compared to FastRoute 3.0, a wire length reduction of 1% and a reduction of the runtime of 50%.

Table 1.1: Results of FastRoute 4.0 compared to FastRoute 3.0 in terms of number of vias, segment wire length and CPU time (XU; ZHANG; CHU, 2009).

	FastRoute 4.0			FastRoute 3.0			4.0:3.0		
Benchmark	# vias	seg wl	cpu (s)	#vias	seg wl	cpu (s)	# vias	seg wl	cpu (s)
bigblue1	1990K	3795K	423	2645K	3866K	773	0.75	0.98	0.55
bigblue2	4455K	5104K	913	4628K	5138K	1797	0.96	0.99	0.51
bigblue3	5179K	7891K	278	5521K	7940K	342	0.94	0.99	0.81
bigblue4	11340K	12825K	674	12767K	12841K	3711	0.89	0.99	0.18
newblue4	4892K	8502K	1135	5334K	8521K	2459	0.92	0.99	0.46
newblue5	8659K	15013K	607	10223K	15184K	1419	0.85	0.98	0.43
newblue6	7701K	10561K	574	9266K	10611K	1357	0.83	0.99	0.42
newblue7	16949K	18742K	11060	19238K	18789K	18084	0.97	0.99	0.61
Average	7869K	10304K	1958	8702K	10361K	3742	0.89	0.99	0.50

Figure 1.8: BoxRouter 2.0 overall flow (CHO et al., 2007).



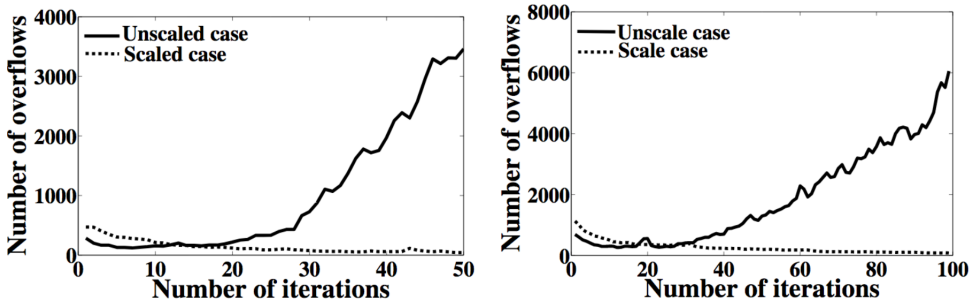
Source: (CHO et al., 2007)

1.3.2 BoxRouter 2.0 (CHO et al., 2007)

BoxRouter 2.0 (CHO et al., 2007) is the second iteration of a global router called BoxRouter. The main contributions are a dynamically scaled negotiation based A* search, topology-aware wire rip-up and an ILP based formulation for layer assignment. The routing flow employed is in figure 1.8. The steps labeled as 2D Global Routing in the figure refer to the fact that all horizontal and vertical layers are collapsed into two layers, one with all horizontal layers and another with all vertical layers. After the 2D routing is available in these two layers, the layer assignment allegedly distributes the guides to the original set of existing layers.

The first two steps of the flow are called minimum Steiner tree (GILBERT; POLLAK, 1968), and net decomposition and Prerouting and Initial Box. These steps refer back to BoxRouter, which produces an initial global routing solution by connecting all nets with a minimal rectilinear Steiner tree built in the G-Cell grid. After this step, the nets are divided into point-to-point connections with two nodes only.

Figure 1.9: BoxRouter 2.0 negotiation based rerouting with scaling factor for ibm01 (left) and imb04 (right) circuits (CHO et al., 2007).



Source: (CHO et al., 2007)

The expansion introduced in BoxRouter 2.0 starts with the Robust Negotiation-based Rerouting. During this step, a modified A* search takes place. The equation for cost used by this A* implementation is in equation 1.1.

$$cost^i(e) = h^i(e) + \alpha p(e) + \beta d(e) \tag{1.1}$$

The term $h^i(e)$ represents the historical cost of the edge e, $p(e)$ represents the present congestion and $d(e)$ represents the distance of the edge to the target. The authors describe a situation where, in a very congested instance, the historical cost increases so much that it becomes the dominant term, which causes the algorithm to route several paths through the same presently congested region while completely ignoring regions with no present routing, but high historical cost. What the authors propose is to add one scaling parameter α. This parameter increases the value of $p(e)$ according to equation 1.2.

$$\alpha = \frac{max_e[h^i(e)]}{p(e)|_{1.0}} \tag{1.2}$$

The result of the scaling parameter is in figure 1.9. In both cases, the implementation without the scaling factor causes the number of overfull G-Cells increases with the number of iterations, while it converges to a tiny amount with the scaling factor.

The layer assignment is performed using an ILP model. The details of this model and the techniques used to create it are out of the scope of this analysis. The authors state that their implementation focuses on reducing the via count. However, the results do not include the number of vias. The achieved results achieved appear in tables 1.2 and 1.3 in terms of number of overfull G-Cells, wire length and, in the latter, the runtime.

Table 1.2: Results of BoxRouter 2.0 in the ISPD07 benchmark suite.

Benchmark	# Overfull G-Cells	Wirelength
adaptec1	0	9204K
adaptec2	0	9428K
adaptec3	0	20741K
adaptec4	0	18642K
adaptec5	0	27041K
newblue1	400	9294K
newblue2	0	13464K
newblue3	38958	17244K

Table 1.3: Results of BoxRouter 2.0 in the ISPD 2008 contest benchmark suite (NAM; SZE; YILDIZ, 2008).

Benchmark	# Overfull G-Cells	Wirelength	CPU (s)
bigblue1	0	5698K	1147
bigblue2	0	9042K	2346
bigblue3	0	13133K	380
bigblue4	472	23156K	52644
newblue4	200	12974K	78225
newblue5	0	23294K	1700
newblue6	0	17975K	1785
newblue7	208	35859K	84743

1.3.3 Summary

To summarize this brief review of some state of the art global routing algorithms, table 1.4 shows the results of FastRoute 4.0 (XU; ZHANG; CHU, 2009), NTHU-R (Chang; Lee; Wang, 2008) and BoxRouter 2.0 (CHO et al., 2007) in terms of the number of overflowing G-Cells, wire length and CPU time in seconds. In terms of the number of overflowing G-Cells, (XU; ZHANG; CHU, 2009) can produce the best results of the three. In terms of runtime, (XU; ZHANG; CHU, 2009) is also the best, being 47% faster than (Chang; Lee; Wang, 2008) and much faster than (CHO et al., 2007). As for the wire length, (Chang; Lee; Wang, 2008) reaches the best results, being 2% on average shorter than (XU; ZHANG; CHU, 2009) and 1% shorter than (CHO et al., 2007).

1.4 Congestion Map

The congestion map is an important tool to estimate routability in any step of the physical design flow (WESTRA; GROENEVELD, 2005). We call congestion map any data structure associated with the routing region that contains information relevant to

Table 1.4: Results of (XU; ZHANG; CHU, 2009), (Chang; Lee; Wang, 2008) and (CHO et al., 2007) in the ISPD07 global routing contest benchmark suite (NAM et al., 2007).

	(XU; ZHANG; CHU, 2009)			(Chang; Lee; Wang, 2008)			(CHO et al., 2007)		
Benchmark	# ovfl	Wirelength	CPU (s)	# ovfl	Wirelength	CPU (s)	# ovfl	Wirelength	CPU (s)
bigblue1	0	5789K	423	0	5631K	586	0	5698K	1147
bigblue2	0	9559K	913	0	9059K	594	0	9042K	2346
bigblue3	0	13070K	278	0	13075K	259	0	13133K	380
bigblue4	152	24165K	674	182	23076K	7533	472	23156K	52644
newblue4	144	13394K	1135	152	12990K	4023	200	12974K	78225
newblue5	0	23672K	607	0	23166K	854	0	23294K	1700
newblue6	0	18262K	574	0	17696K	818	0	17975K	1785
newblue7	62	35691K	11060	68	35357K	8433	208	35859K	84743

routability. In some cases, global routing is used as congestion estimation. However, the correlation between global routing congestion estimation and actual violations is weakening in recent technologies (CHAN et al., 2017). Figure 1.10 shows in (a) the resulting congestion map based on global router, with green and blue representing less congested regions and red and pink representing more congested regions, and in (b) the red dots represent opens and shorts of detailed routing. It is clear that there are false positives in the global router prediction and some violation hotspots in the detailed router that weren't marked in the congestion map, which strengthens the conjecture that global routing generated congestion map only weakly correlates to actual poor routability hotspots.

Therefore, more sophisticated algorithms have been proposed in the literature. In (Tabrizi et al., 2018) and (Tabrizi et al., 2017) the authors propose a machine learning framework that predicts shorts in a given placed netlist. In (CHAN et al., 2017) and (Qi; Cai; Zhou, 2014) the proposed machine learning framework attempts to predict DRC violations for a given global routing solution. In (SHOJAEI; DAVOODI; LINDEROTH, 2011) a similar technique is employed but uses ILP instead of machine learning.

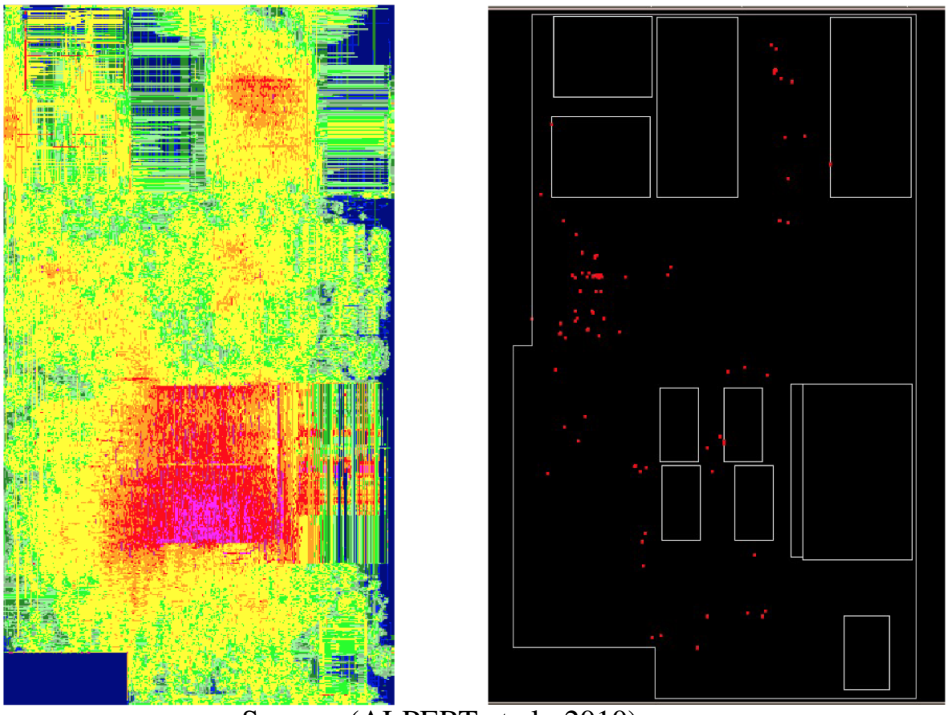
Depending on the flow, the congestion map could have many uses. In extremely congested regions, a detailed placer could attempt to move cells out of the congested region. The global router can also run more iterations of its algorithms using the congestion map to try to create a global routing solution that has less congested G-Cells. Finally, the initial detailed router can use the congestion map to estimate the priority of a particular net when allocating routing resources to segments of a net.

1.5 Research on Routing at UFRGS

For more than three decades, the researchers and students at UFRGS have produced many works in the area of physical design automation: automatic cell layout gen-

Figure 1.10: (a) Global routing generated congestion map, where blue and green represent less congested regions and red and pink represet more congested retions, and (b) where the red dots represent detailed routing opens and shorts map (ALPERT et al., 2010)

(a) (b)



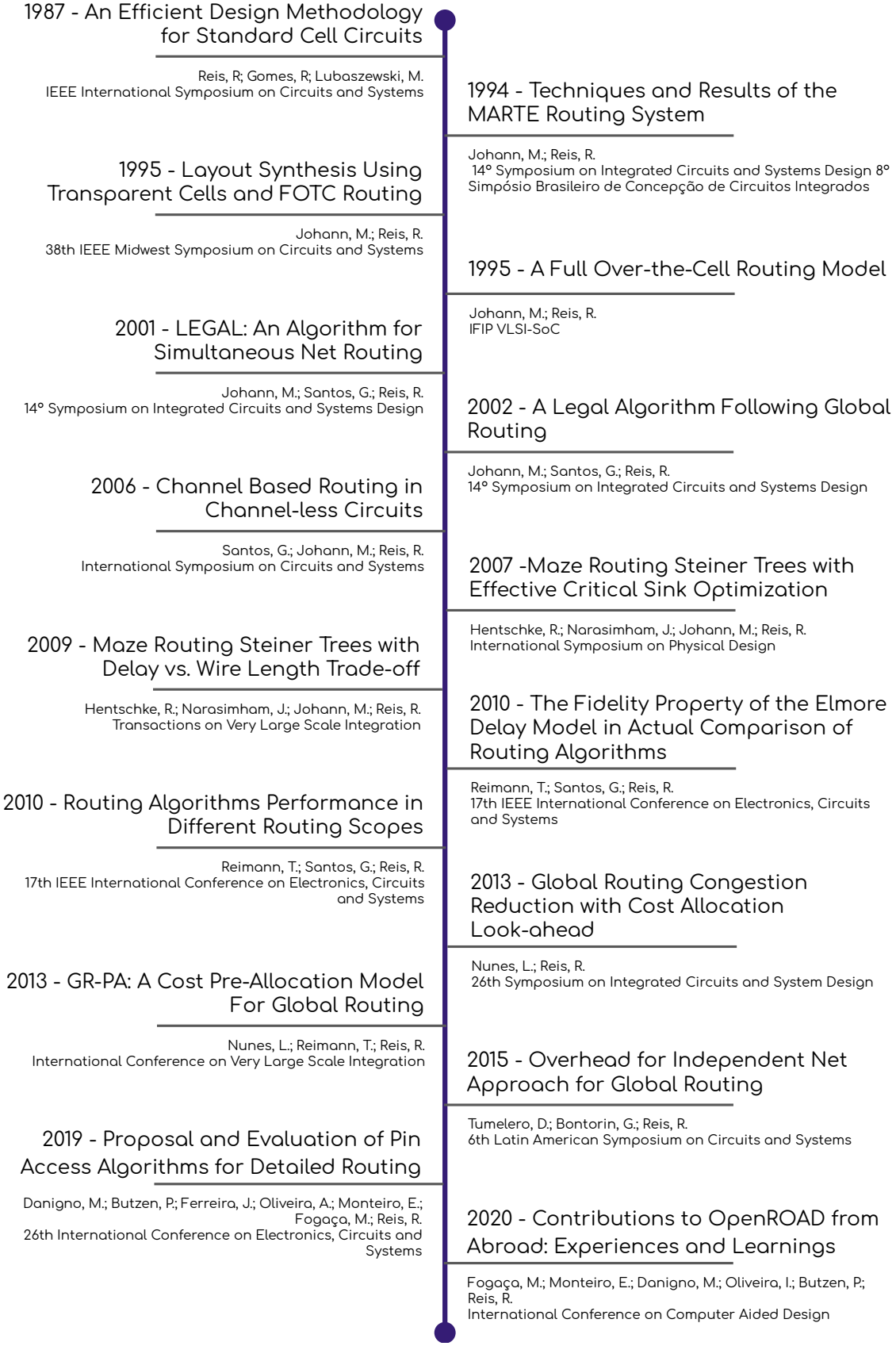
Source: (ALPERT et al., 2010)

eration, placement, gate sizing, legalization, routing, and more. For the purpose of this work, we will study the timeline of the routing related works researched in UFRGS. Figure 1.11 shows a timeline of some of the routing related works published by UFRGS students and researchers.

In (REIS; GOMES; LUBASZEWSKI, 1988) the authors propose a standard cell design methodology that connects pins through routing over-the-cell instead of through the channel. At that time, there were reserved bands of space between the rows of standard cells reserved for channel routing. Modern designs no longer have these channels, and routing is done over the cell. One of the challenges of routing over the gates at the time was the limited number of metal layers.

In (JOHANN; REIS, 1994) and (JOHANN; REIS, 2001), the authors propose a routing technique called LEGAL, where a combination of channel and greedy routing is used to route all nets simultaneously. The authors also describe that this work is intended to route over the cell, using the extra available metal layers instead of the more traditional channel-based routing at the time. In 2002, (JOHANN; SANTOS; REIS, 2002) propose an implementation of LEGAL that is global routing-aware. The authors state that a better separation of global and detailed routing is necessary to handle cases with very large nets.

Figure 1.11: Timeline of some of the routing related research at UFRGS - not to scale.



Source: From the author.

In (JOHANN; KINDEL; REIS, 1995) and In (JOHANN; REIS, 1995), the authors describe a model of full over-the-cell (FOTC) routing to connect the pins by using the called *"transparent cells"*. The transparency of the cells comes from the fact that in simple over-the-cell (OTC) routing the first metal layer is reserved for intra-cell routing, whereas the FOTC model allows using of all layers for routing. Another advantage is that FOTC eliminates the necessity for feed-through structures that OTC requires.

In (SANTOS; JOHANN; REIS, 2006), the authors propose an algorithm that can route horizontally aligned terminals in a cell-based design with no channel using channel routing techniques. The main advantage of this strategy is the possibility of routing the horizontally aligned terminals in almost linear time, enabling acceleration of the design flow.

In (HENTSCHKE et al., 2007) and (HENTSCHKE et al., 2009) the authors propose a rectilinear Steiner tree building algorithm based on path search. The work's main contribution is the introduction of factors that allow the trade-off between wire length and delay of the routed nets.

In (REIMANN; SANTOS; REIS, 2010) and (SANTOS et al., 2010) the authors study the methodology of comparison between different routing algorithms. In (REIMANN; SANTOS; REIS, 2010), the authors study the differences between different interconnection delay models, and the major contribution of the work is the discovery that the academy standard model has a high standard deviation. In (SANTOS et al., 2010), the authors study the significance of the methodology of comparison when considering different scopes - for example, the size of the nets used in the comparison. The main contribution is the discovery that different methodologies are adequate to different scopes.

In (NUNES; REIS, 2013) and (NUNES; REIMANN; REIS, 2013) the authors propose two techniques that identify congestion-prone regions during global routing. The first technique looks for regions with a high density of pins, and the second looks for congestion of wires during each routing iteration. According to the authors, these techniques allow for a significant speedup of the global routing flow with a minimal penalty to wire length.

In (TUMELERO; BONTORIN; REIS, 2015) the authors analyze the available academic benchmarks of ISPD 2008 and identify that most of the nets are small. Then, the authors propose a clustering technique that processes the small nets in parallel.

In (DANIGNO et al., 2019) the authors propose five algorithms for pre-processing the design and find valid options for accessing pins. The major contribution is an algo-

rithm that can generate valid pin access for 99% of the pins in the ISPD 2018 contest in under five minutes without using off-track vias.

In (FOGAÇA et al., 2020) the authors talk about the OpenROAD project, an ambitious open-source project that aims to develop a fully automated RTL-to-GDSII flow. The project included an implementation and improvements to FastRoute (XU; ZHANG; CHU, 2009) developed at UFRGS.

2 FUNDAMENTAL ALGORITHMS

The problem of efficiently finding routes to connect points is neither recent nor restricted to design automation. Many of the algorithms were developed and studied by other fields, such as artificial intelligence and logistics. In the EDA field, these algorithms are usually the building blocks of more sophisticated algorithms. This section will cover some of these algorithms that have reached a certain level of generality. The classes studied are pathfinding algorithms, Steiner tree generation algorithms, and rip-up and reroute.

2.1 Pathfinding Algorithms

A pathfinding algorithm's objective is to output a path in a particular environment that connects a set of points. In the routing scope, we are interested in algorithms that can produce a set of wires that connect pins. There are many algorithms designed to find such paths. In the following subsections, we will explain how some of the main algorithms achieve their objectives.

2.1.1 Maze Router (LEE, 1961)

One of the first algorithms for solving path connection problems (LEE, 1961) describes a breadth-first search algorithm in a grid. The applications cited by the author are logic drawing, wiring diagramming, and optimal route finding. The algorithm proposed, known as the maze router, works in a two-dimensional region with three types of entities: sources, targets, and blockages. Each source is bound to a target. The goal is to find a path from each source to its target without crossing any blockages. If multiple sources/targets exist, the already existing paths act as a blockage. The algorithm moves from the source point toward the neighbor points, labeling them as 1. The algorithm continues forward from each neighbor, labeling the next neighbors as 2, and so on. When the wave of increasing numbers reaches the target, it backtracks by finding decreasing numbers until the source. Figure 2.1 shows this process occurring. In (a) there is a 2D region with the three markers, S being a source, B a blockage, and a T target. In (b), the search has already expanded the first neighbors. In (c), the first neighbors labeled 1,

expanded to their neighbors, labeled 2. The expansion continues until, as in (d), one of the nodes expands the target. When this happens, the backtracking process begins, traveling back following the decreasing numbers until the source is found, as in (e). Aker later discovered and published in (AKERS, 1967) that only three different labels are sufficient, or even two, reducing the algorithm's memory usage. Using three labels $k-1$, k, and $k+1$, for example, 1, 2 and 3, allows for backtracking as well. Instead of decreasing the label indefinitely until reaching the source, the pattern wraps around at 1 and goes back to 3. The scheme with two values for labels uses, for example, 1 and 2 as the labels. The labeling technique has to insert labels in a sequence such as 1, 1, 2, 2, 1, 1, 2, 2 until it reaches a target. This way, the backtracking algorithm can look for a path that follows the known pattern.
(SOUKUP, 1978) proposed an improvement on the maze router. Instead of relying solely on breadth-first search (MOORE, 1959), when a node n expands and finds a node n' that is closer to the target than n, it runs a $depth$-first search in n' direction and adds all discovered nodes to the front.

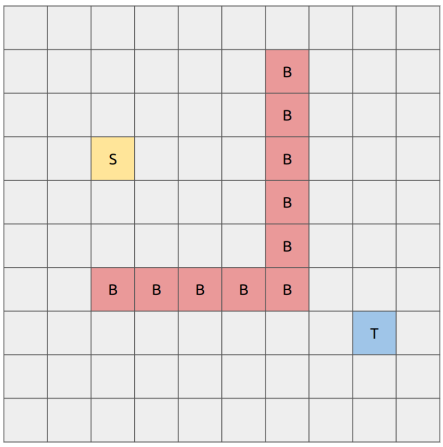
2.1.2 A* (HART; NILSSON; RAPHAEL, 1968)

Best-first search is another class of generic pathfinding algorithms. Instead of *breadth*-first, that is, expanding first the nodes at the front at every iteration, A* proposes to incorporate heuristic estimations of the distance into the search (HART; NILSSON; RAPHAEL, 1968), which allows the algorithm to focus on expanding the more promising nodes. To find such higher potential nodes, the A* algorithm always has two functions associated with each node: $g(n)$ and $h(n)$.

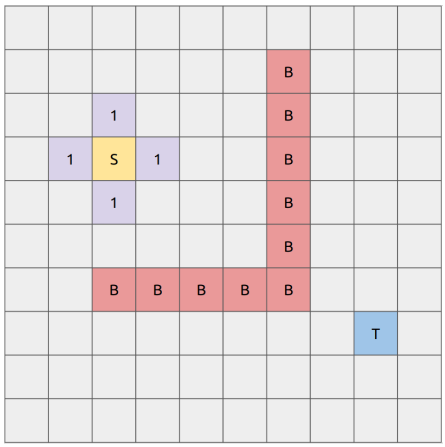
The algorithm updates both functions whenever it expands a node, even if it has already expanded it before. $g(n)$ is the cost of the path from the source to the node n, and $h(n)$ is the estimated cost of the route from node n to the target. By keeping all of the nodes sorted by weight and expanding first the best nodes, that is, the smallest $f(n) = g(n) + h(n)$ the algorithm prioritizes the most promising paths. The optimized search allows for better efficiency compared to the maze router since it will expand fewer nodes. Furthermore, the authors have proven that if $h(n)$ is optimistic, that is, always estimates the cost from n to the target to be smaller or equal than it is, A* will always find the shortest path. Figure 2.2 shows the same route searched by maze router in (a) and A* in (b), and it is clear that A* has expanded through fewer nodes.

Figure 2.1: Maze router's breadth-first search algorithm.

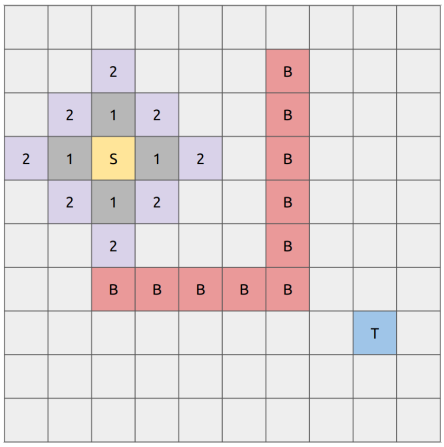
(a) Initial state, with blockages in red, start point in yellow and target point in blue.



(b) First expansion of the search front, labeled 1 in the figure.

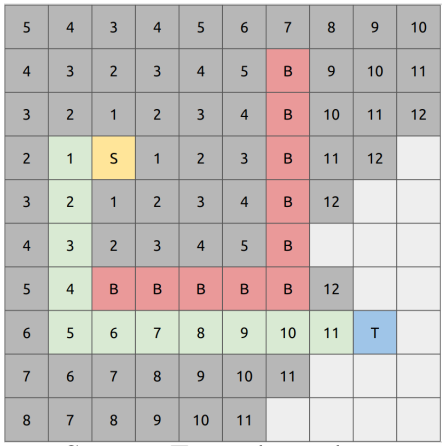


(c) Second expansion of the search, labeled 2 and in light purple, where the already expanded nodes are in gray.



(d) All expansions of the search, with all expanded nodes in gray, all nodes in the search front in light purple and one expanded node adjacent to the target.

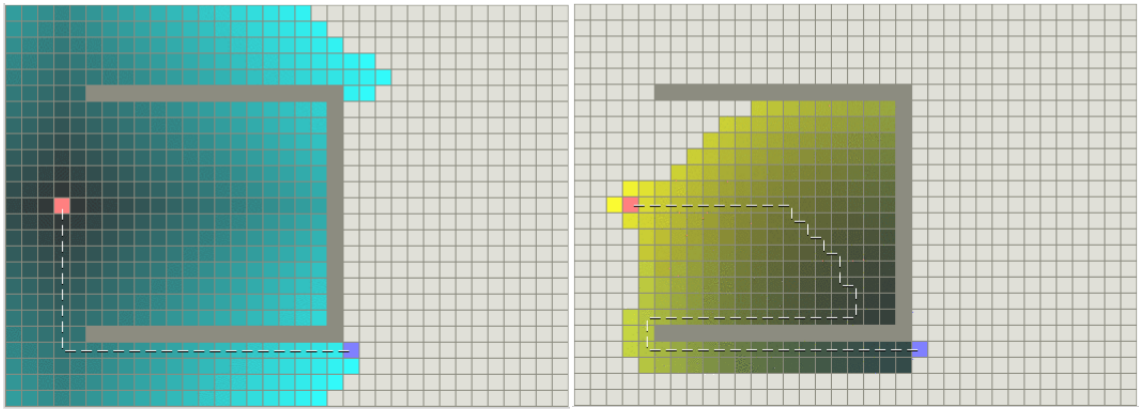
(e) Path obtained by backtracking from the target through the expanded nodes in descending order, forming a green path.



Source: From the author.

Figure 2.2: A* and maze router searching for the same path (PATEL, 2018).

(a) (b)



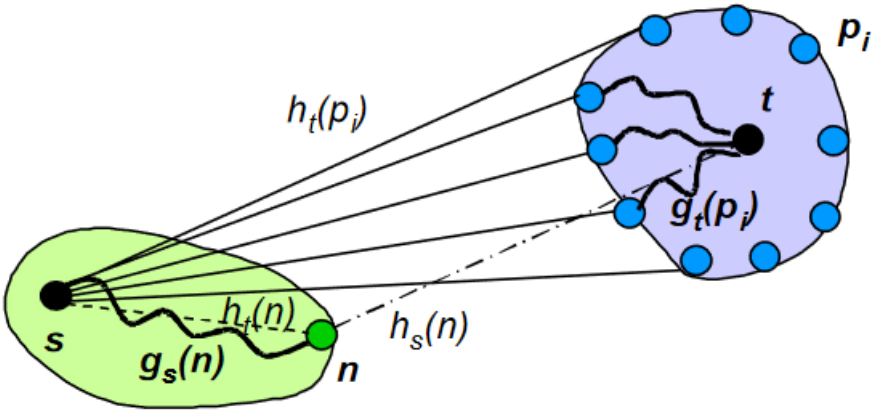
Source: (PATEL, 2018).

The first published work that applies A* to VLSI routing is (CLOW, 1984). The author adds important concepts from physical design to the A* technique - first, he raises the problem of handling multiple pin nets and multiple terminal pins. The work proposes grouping all terminals of the same pin and running multiple sources multiple targets A*. Instead of one source and one target initially, the algorithm starts with multiple, and the goal is to connect one source terminal to one target terminal with the shortest path. Furthermore, the author describes a problem called *inverted corner*, where two paths of the same cost tie and the A* algorithm chooses any of the possibilities. Instead, one would wish that the route had as few changes in direction as possible and have most of the wire in the preferred direction. The solution is to add a small penalty called ϵ for spanning in the non-preferred direction that serves as a tiebreaker in these situations.

Since the original paper, researchers have published some works on bidirectional A* path search. The idea is to start a search from the source and target nodes simultaneously. (POHL, 1971) proposed a bidirectional search that expands until the two fronts find each other. From the meeting node, a new A* search takes place. Based on this work, (KWA, 1989) proposed in his work four techniques. Let us assume that there is a search front called S from the source, and from the target, there is a front called T.

- Nipping: If front T (S) tries to expand a node that S (T) has already discarded by the node is not expanded because the other front has already discovered that it does not belong to any shortest path.
- Pruning: The nipped node may have open nodes in the discoverer front. The algorithm discards these open nodes because if the node is not in the shortest path, neither are the nodes opened from it. Removing these nodes from the open set is

Figure 2.3: LCS bidirectional search with two search fronts, one expanding from the source and one from the target (JOHANN, 2001).



Source: (JOHANN, 2001)

called pruning

- Trimming: When a meeting node expands, it removes any node in the open set with a higher cost than it since a shorter path exists.
- Screening: The screening process is to try to trim a node before adding it to the open set by checking the smallest cost of the meeting nodes discovered.

Even with this progress, the bidirectional A* search was still slower than A* in some cases. The bidirectional search schemes added overhead that did not overcome the benefits. The first algorithm that was able to outperform A* consistently was (JOHANN, 2001). The work's main contribution is the use of the other search front's information to improve the cost function. One front's g(n) is the other front's h(n) and vice-versa. The authors introduce two variables, Py, and ω. Figure 2.3 illustrates these concepts.

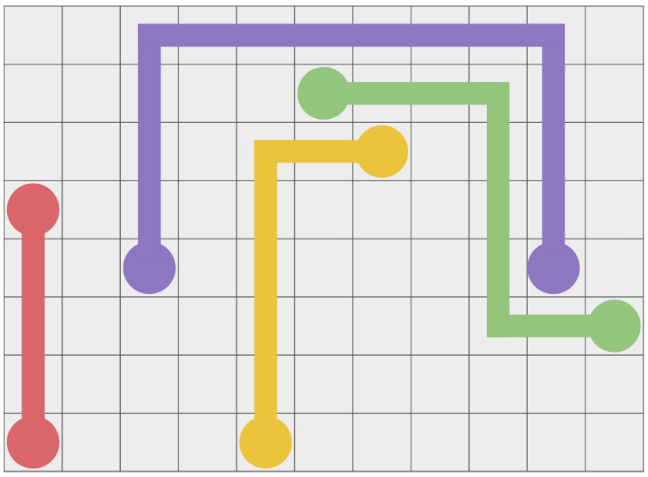
The A* function incorporates these values. Equation 2.1 shows the new equation for a node n in the source search front. Notice how it uses the other search front information in $Py_t(n)$ and $h_t(n)$

$$f_s(n) = g_s(n) + Py_t(n) - h_t(n) \tag{2.1}$$

2.1.3 Pattern Router (ASANO, 1982)

The pattern router technique consists of connecting two points through predefined topologies (ASANO, 1982). In his work, the author showed an exhaustive search method for routing. Figure 2.4 shows some of the possible basic topologies. These are four of

Figure 2.4: Four basic topologies of Pattern Router: line in red, L in yellow, U in purple, and Z in green.



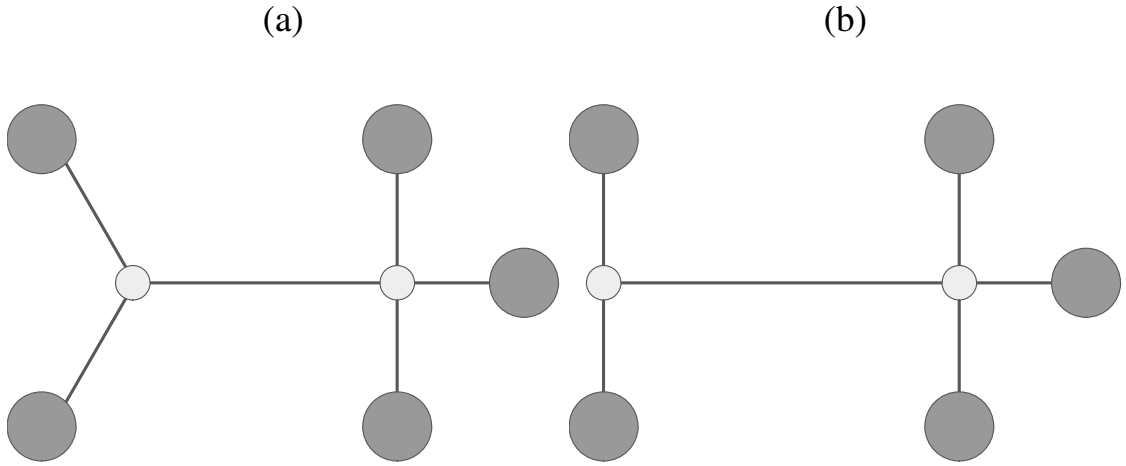
Source: From the author.

the possible structures, the line-, L-, U- and Z-shapes. The process of deciding which shape is the most adequate to connect two given points is called zoning. It consists of exhaustively searching for a solution. The author limits the search space to a region called a zone. When a connection is impossible, the algorithm reroutes previously made connections as an attempt to complete the routing - which turns out to be understandably very slow. However, the basic idea can apply for straightforward routing in more complex routing flows because of its simplicity. One of the works that accomplished this is (DAS; KHATRI, 2001), where the authors combined the pattern router with the maze router to achieve a routing solution with fewer corners, therefore, more regular.

2.2 Steiner Tree-based Algorithms

Steiner trees are structures used in many fields. In VLSI design flow, there is special interest in *rectilinear* Steiner trees, where every branch is aligned to the x- and y-axis. These trees are used to generate a path between multiple points in a grid; the main idea is adding points to the problem, Steiner points. There are multiple types of Steiner trees, the most important in the context being Rectilinear Minimum Steiner Tree, a rectilinear tree that is also the smallest possible. Figure 2.5 contains two Steiner trees connecting five nodes. The tree in (a) is a Minimum Steiner Tree (MST), and the tree in (b) is a Rectilinear Minimum Steiner Tree (RSMT). Notice that in both cases, there are two light gray nodes. These nodes are not part of the original problem. Instead, the tree construction algorithm added these so-called Steiner nodes to assist in the tree construction.

Figure 2.5: Two Steiner trees, in (a) a minimum Steiner tree (MST) and in (b) a rectilinear minimum Steiner tree (RMST).



Source: From the author.

In 1966, Hanan showed that if one wishes to obtain an RSMT, it is sufficient to consider only the columns/rows that contain points (HANAN, 1966). Constructing a grid with only these columns and rows produces a Hanan grid. Figure 2.6 shows the process of building the Hanan grid. In (a), there is a grid consisting of nine vertical and nine horizontal tracks. There are four points in this grid, and the objective is to connect all four with an RSMT. Since we know that the RSMT is in the Hanan grid, we can simplify the problem by building it. In (b) the relevant tracks appear highlighted, and finally in (c) the Hanan grid. The number of possible Steiner nodes in (a) was sixty, and in (c) it is twelve. The algorithms covered consider the Hanan grid as the routing region. Notice that obstacle vertices are points in the grid.

Many algorithms that create such trees appeared, some at the placement stage of the flow for routability and timing estimation, some at the global routing stage, and detailed routing. The existence of obstacles in the routing region is also a research topic, and some algorithms are developed take them into account.

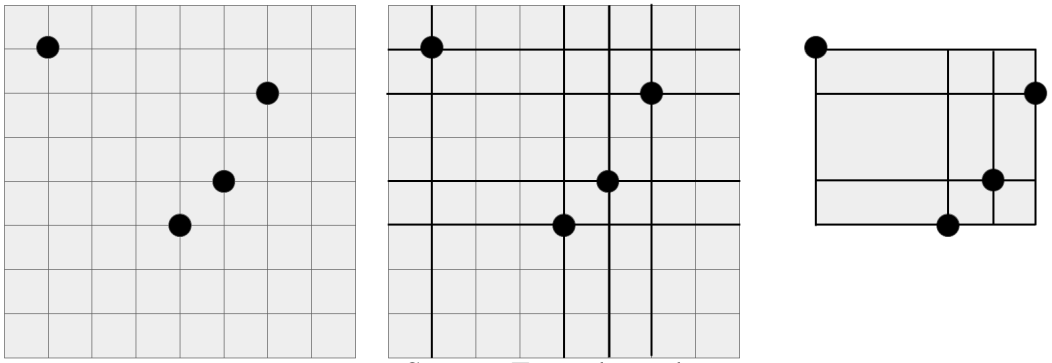
There are four major classes of algorithms that use Steiner trees: track graph, escape graph, spanning graph and look-up table based. We will present the main works for each category in the following subsections.

Figure 2.6: Construction of a Hanan grid.

(a) Base grid with the four relevant points.

(b) The base grid but with the relevant horizontal and vertical tracks highlighted.

(c) A trimmed grid created from the maximum and minimum points of the intersections between the relevant tracks.



Source: From the author.

2.2.1 Track Graph Based

One of the most important track graph algorithms is (WU et al., 1987). It creates a graph using the tracks found at the routing points and obstacle vertices and runs a shortest path search algorithm on this graph to find the shortest path from any point to all other points. An example of a track graph in an instance of a Steiner tree problem is presented in figure 2.7. The algorithm's complexity is $\mathcal{O}(n^4)$, and the authors state that it guarantees the generated tree is minimal for up to six points.

2.2.2 Escape Graph Based

From the escape graph class, the idea is to create a graph by extending the horizontal and vertical segments of every vertex of every polygon until the net bounding box's boundaries. In (GANLEY; COHOON, 1994), the authors proved that if a problem instance is solvable, then the solution is in the escape graph. However, the escape graph is larger than a track graph. Figure 2.7 shows an escape graph in (a) and a track graph in (b).

Given the size difference, one can conclude that the shortest path search algorithms will be slower in the escape graph. The authors propose that two algorithms for 3- and 4-nodes instances - with complexities of $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ respectively - can be used to produce an approximated solution for larger instances, dividing the problem into smaller subsets of K points. The technique, called K-Steinerization, has a worst-case complexity

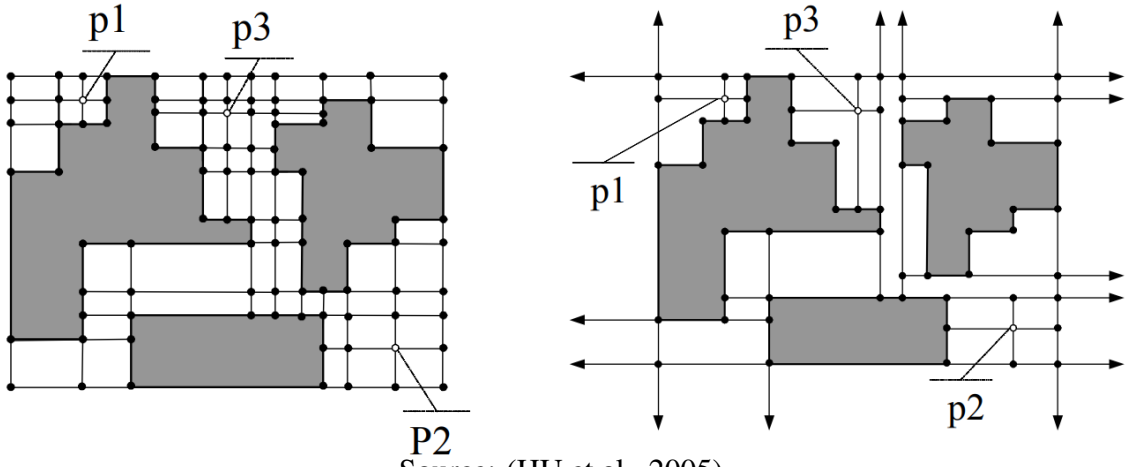
of $\mathcal{O}(k^3n^2)$ where k is the number of terminals and n is the number of candidate Steiner points.

More recently, in (CHANG et al., 2008) the authors proposed a technique to construct a Steiner tree by applying the shortest path search and minimum spanning tree algorithms in the escape graph. The generated tree is minimal for n=2. The worst-case complexity is $\mathcal{O}(n^3)$, but the authors state that the complexity for practical cases is $\mathcal{O}(n^2 \log n)$.

Figure 2.7: An escape graph (a) and track graph (b).(HU et al., 2005).

(a) The escape graph, where every vertex of every polygon produces an edge.

(b) The track graph, where the vertices of the polygons produce edges that only span until they collide with a polygon, resulting in much less vertices and edges in the graph.

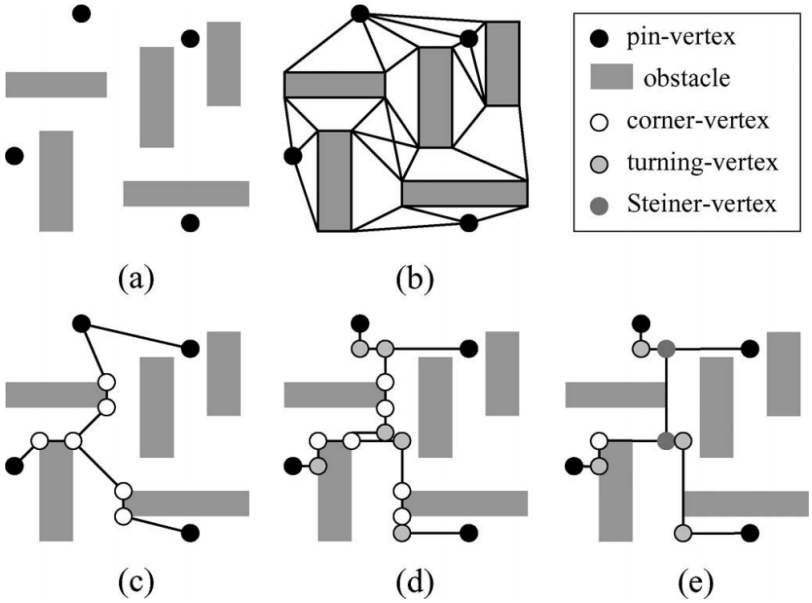


Source: (HU et al., 2005)

2.2.3 Spanning Graph Based

The third class of Steiner tree construction algorithms utilizes a spanning graph. This graph is built by creating a node for each obstacle vertex and pin, then connecting the neighboring vertices by an edge. One of the published works that make a tree using the spanning graph is (LIN et al., 2008). Figure 2.8 shows their flow, where the graph building and tree construction happens. In (a), the author displays the input: a set of pins and obstacles in a 2D region. In (b), the spanning graph is constructed, with the pins' and obstacles' vertices acting as graph vertices and edges appearing between close vertices in the region. In (c), the authors show the shortest spanning tree obtained from the spanning graph. The remaining steps (d) and (e) correct the spanning tree to be rectilinear using

Figure 2.8: Example Steiner Tree construction using a Spanning Graph (LIN et al., 2008). In (a), the problem input consists of four pins and five obstacles. In (b), the vertices of the obstacles and the pins form connections in the spanning graph, based on their distance. In (c), the graph is trimmed into the shortest spanning tree. In (d), Steiner nodes are introduced to make the spanning tree rectilinear. In (e), the redundant nodes are hidden and the final result is obtained.



Source: (LIN et al., 2008)

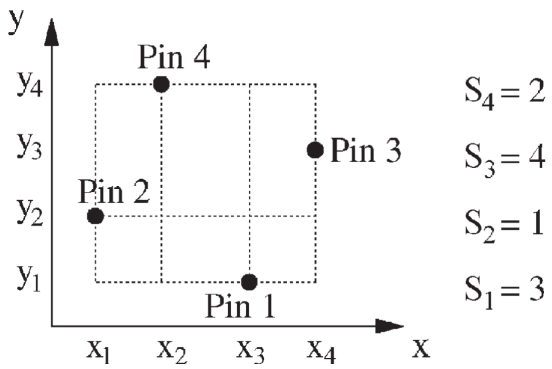
Steiner nodes and then hide all redundant nodes to generate the final result.

2.2.4 Look-up Table Based

Finally, the currently most important type of Steiner tree routing algorithm is the LUT-based. Instead of constructing the tree on the fly, this technique constructs a database of premade trees. The algorithm that wishes to use a Steiner tree can access the database and retrieve a solution. In (CHU; WONG, 2008), the authors state that their database contains optimal trees for up to nine nodes, and accessing the database has a complexity of $\mathcal{O}(n \log n)$, where n is the number of pins in the problem. The tool has to calculate the pin order vector and the wire length vector to access the database. Figure 2.9 shows the pin ordering vector generation. From bottom to top, the pin's horizontal position concerning the other pins is its order. In this example, the bottom pin is the third horizontally, which makes its order 3. The final ordering of the example is 3, 1, 4, 2. The wire length vector contains information about the distance between the lines and columns of the grid. Based on the ordering string and the distance between lines and columns, the algorithm can search the database and get the solution faster than solving the instance by itself. The

solution contains the y and x segments that should contain a wire.

Figure 2.9: Example configuration of four pins generating an order vector (CHU; WONG, 2008).



Source: (CHU; WONG, 2008)

2.2.5 Summary

Table 2.1 compares the works explained in the previous subsections in terms of data structure and used algorithms, the worst-case complexity, and the maximum number of points for which the algorithm generates minimal trees. Note that the worst-case complexity refers to theoretical scenarios that usually do not occur in practice. For the complexity, n denotes the number of nodes, and k denotes the number of possible Steiner points. The data structure and algorithm fields for (CHU; WONG, 2008) are highlighted because although the Steiner tree generation uses them accessing the tree from the database is a simple access, and no actual algorithm is executed. Also, denoted by †, the complexity stated is for accessing the database, not building the tree. Although the authors propose a tree generation algorithm, studying the uses of such trees suffices for this document's scope. In the algorithm column, SPS stands for shortest path search and MST for minimal spanning tree. In data structures, POWV stands for potentially optimal wire length vector.

2.3 Rip-up and Reroute

Initially proposed in (SHIRAKAWA; FUTAGAMI, 1983), the rip-up and reroute technique removes part of the existing routing and replaces it with another. The primary goals are to route previously impossible connections and reduce wire length routing un-

Table 2.1: Table comparing different Steiner tree related works. #Points refers to the maximum number of points in the instance that the algorithm can generate a minimal tree.

Algorithm	Data Structure	Algorithm	Complexity	#Points
(WU et al., 1987)	Track Graph	SPS	$\mathcal{O}(n^4)$	6
(GANLEY; COHOON, 1994)	Escape Graph	MST	$\mathcal{O}(k^3n^2)$	4
(CHANG et al., 2008)	Escape Graph	SPS and MST	$\mathcal{O}(n^3)$	2
(LIN et al., 2008)	Spanning Graph	MST	$\mathcal{O}(n^2)$	2
(CHU; WONG, 2008)	POWV*	MST*	$\mathcal{O}(n \log n)^\dagger$	9

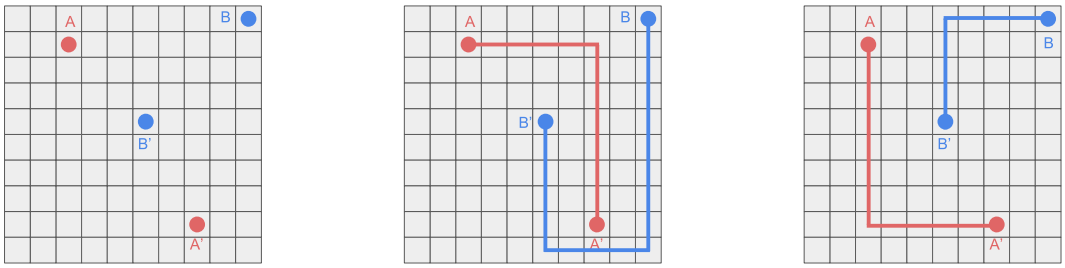
necessary detours. The underlying phenomenon that rip-up and reroute tries to handle is the order of routing. Figure 2.10 exemplifies the first problem of unnecessary detours. Notice how in (b) choosing to route A to A' first can cause the path from B to B' to be longer than necessary. By changing the order and routing B to B' first, the router finds the desired solution (c).

Figure 2.10: Ordering problem example where longer than optimal wire length is avoided.

(a) Input problem, with two nets A and B, both with two pins, namely A and A' and B and B' respectively.

(b) Possible routing solution for nets A and B, routing net A first. Note that net B has sub-optimal wire length.

(c) Another possible routing solution for nets A and B, routing net B first. Note that both nets have optimal wire length.



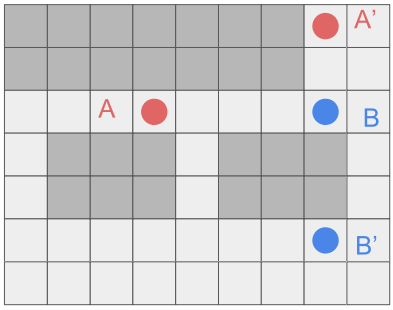
Source: From the author.

Figure 2.11 shows an instance where routing A to A' or B to B' causes the other to be unroutable in a particular configuration. This instance is harder to solve since the ordering is not the only factor in place. Notice from (b) and (c) that merely changing the order does not solve the problem. A partial rip-up and reroute is needed, or a negotiation based on rip-up and reroute. The negotiation flow increases the cost of using a region based on how many nets want to route through it. By raising the conflict region's weight and rerouting both nets, the router achieves the desired solution (d).

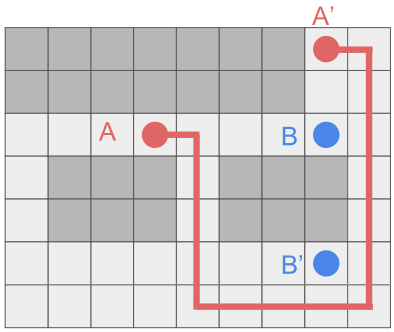
The rip-up and reroute scheme can use multiple routing algorithms, for example, A*. However, if one wants to use the negotiation scheme, the routing algorithm needs to consider a more sophisticated *cost* of routing, not only wire length. For example, inserting a wire of length L in a congested region could cost $2 * L$, thus encouraging the use of

Figure 2.11: Ordering problem example where an unroutable net is avoided.

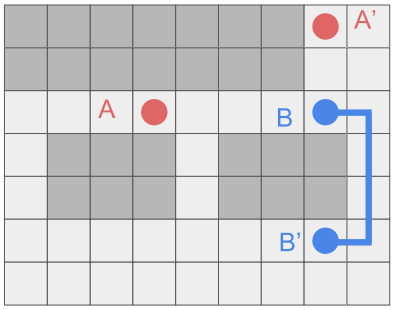
(a) Input problem, with two nets A and B, both with two pins, namely A and A' and B and B' respectively.



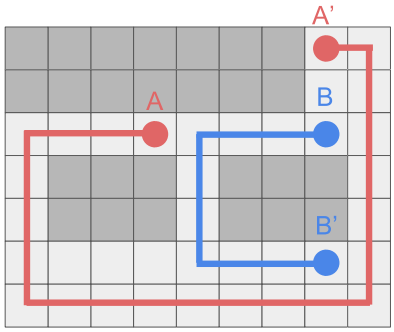
(b) Routing net A first with the shortest path renders net B unroutable.



(c) Routing net B first with the shortest path renders net A unroutable.



(d) A negotiation based rip-up and reroute needs to be used to route both nets.



Source: From the author.

Table 2.2: Comparison of the fundamental algorithms.

Algorithm	#Sources/ #Targets	Time Complexity	Guarantees Path	Guarantees Min Length
Maze Router	1/1	$\mathcal{O}(h * w)$	Yes	Yes*
A*	n/n	$\mathcal{O}(\log h * w)$	Yes	Yes*
Bidirectional A*	n/n	$\mathcal{O}(\log h * w)$	Yes	Yes*
Track Graph Steiner Tree	n/n	$\mathcal{O}(n^4)$	Yes	Yes (2)†
Escape Graph Steiner Tree	n/n	$\mathcal{O}(n^3)$	Yes	Yes (4)†
Spanning Graph Steiner Tree	n/n	$\mathcal{O}(n^2)$	Yes	Yes (2)†
LUT Based Steiner Tree	n/n	$\mathcal{O}(n \log n)$	Yes(9)†	Yes(9)†
Pattern Router	1/1	$\mathcal{O}(h * w)$	No	No

alternative routes. Many modern works utilize rip-up and reroute in their flow. (GESTER et al., 2013) uses a negotiation-based rip-up and reroute in their detailed routing flow. (SUN et al., 2018) routes every net in parallel and then resolves conflicts using rip-up and reroute combined with a conflict graph. (JIA et al., 2014) utilizes rip-up and reroute to fix min-area violations, but not to fix net ordering.

2.4 Summary

Table 2.2 compares the algorithms studied in terms of the number of sources and targets supported, time complexity, the guarantee of finding a path, and the guarantee of the path being of minimum length. Note that these values assume the classic implementation unless otherwise stated.

The fields assigned with * state that the algorithm finds the path with minimum length, which is true if we consider only the path itself and no obstacles. Even then, considering the path routed at the time that it executes, the shortest path *available* is guaranteed to find. The fields assigned with † have the maximum number of nodes that the algorithm guarantees to find a minimal path (PEARL, 1984). In the complexity notation, h stands for the height of the routing region, w for the routing region's width, and n for the number of points.

3 DETAILED ROUTING ALGORITHMS

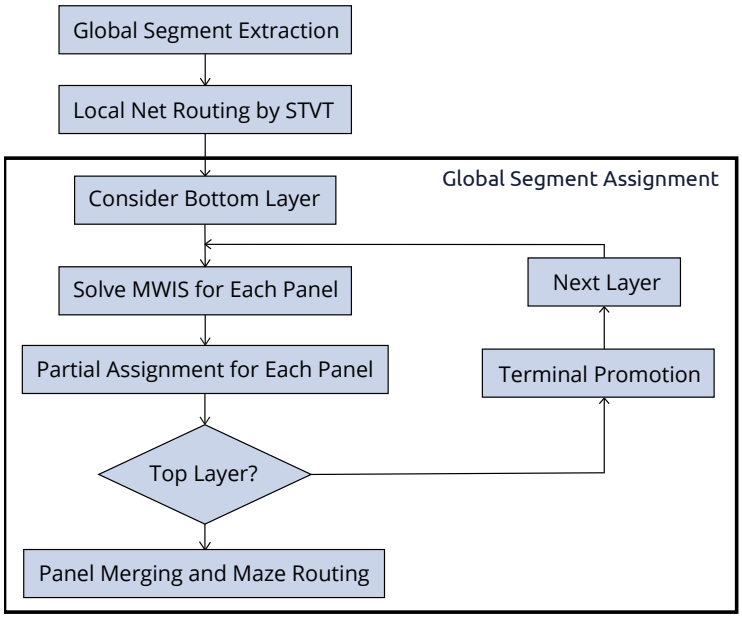
The more recent technologies brought more challenges to initial detailed routing. The major problems that have arisen are pin accessibility and more strict design rules. The first occurs when a pin is no longer accessible because of other structures in the vicinity. For example, suppose a pin is in metal 1, and there are wires in metal 2 covering the pin entirely so that there is no room for a via to access the pin. In that case, this configures a pin accessibility violation. As for design rules, the more recent technologies added many new constraints, most of which are a huge challenge to incorporate them in an initial detailed routing algorithm. Even simpler constraints such as minimum area are difficult to treat unless the routing polygons are created correctly by construction and require no later analysis and repairs.

Some of the challenges that emerged with the newer technologies are the focus in (LEUNG, 2003), where the author also discusses how modern routers can improve. One of the constraints that are more complex in new technologies is the set of spacing rules. For example, instead of requiring a fixed distance between two wires, the minimum spacing is a function of both the width and length. In even newer technologies, this can be a function of the parallel run length, that is, the distance that two wires span in parallel. Another exciting topic is redundant via insertion. The author states that foundries encourage the use of redundant vias to tolerate failure, indicating that 70-80% of single vias can have a redundant via added without causing violations. The ISPD Initial Detailed Routing Contest of 2019 incorporates these additional constraints (POSSER et al., 2018).

3.1 RegularRoute (ZHANG; CHU, 2011)

RegularRoute is a framework for initial detailed routing (ZHANG; CHU, 2011). The authors proposed a flow divided into three main steps, local net routing, global segment routing, and maze routing. Figure 3.1 shows a flowchart where the algorithm's sequence is detailed. The flow starts extracting the global segments by finding guides that span through more than one G-Cell. Afterward, local nets - nets contained entirely in a G-Cell - are routed using Single Trunk V-Tree (STVT). Next, the algorithm works in a layer-by-layer fashion. For each layer, each panel is independent. On each panel, a maximum weighted independent set (MWIS) formulation models the global segment assignment. By solving the MWIS, the algorithm can route a subset of the segments in

Figure 3.1: RegularRoute flow chart.



Source: (ZHANG; CHU, 2011)

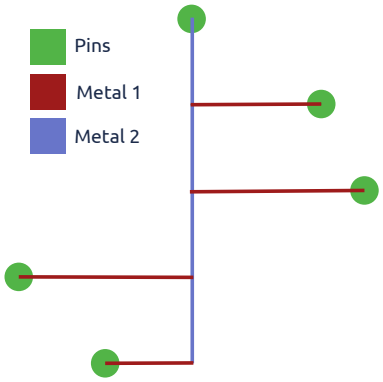
the panel. After considering all the metal layers, the algorithm employs a panel merging and maze routing for routing closure.

It is important to note that RegularRoute has two limitations. The first limitation is that it takes a 2D global routing solution as input, which means the segments are not assigned to a layer and are free to be routed in any layer. In their work, the authors briefly describe a technique to consider layer assignment. However, they insist that the global router does not have enough information to assign layers to segments, and the authors believe the initial detailed router should perform the layer assignment. The second limitation is that the work considers that all pins are in metal 1, which is not always the case. This limitation is not addressed in detail, as the authors found satisfactory to state *"In theory, our algorithm is applicable to test cases with pins on different layers."* (ZHANG; CHU, 2011)

3.1.1 Single Trunk V-Tree

The technique proposed by the authors of RegularRoute to route local nets utilizes a single trunk tree approach. In this specific work, a vertical trunk is created in the local net's central horizontal point, spanning from the lowest to the top vertical pin position. This trunk is in the metal2 layer. From this trunk, horizontal branches of metal1 connect all pins of the net to the trunk. Figure 3.2 shows an example of a net with five pins routed by an STVT. The authors state that the algorithm can construct the tree with linear time

Figure 3.2: Single trunk V-Tree routing a net.



Source: From the author.

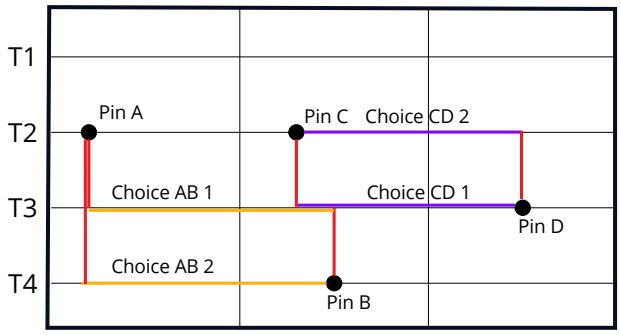
concerning the number of pins. Furthermore, they claim that the STVT is a good choice because it reduces the usage of metal 2 when compared with other candidate topologies, rectilinear Steiner minimum tree (RSMT), for example. In the event of the same gcell containing two local nets, there is a chance of the trees conflicting. The algorithm proposes two alternatives. First, it considers neighbor tracks for the trunk and branches. If this approach fails to resolve a conflict, it can use higher metal layers.

3.1.2 Global Segment Assignment

To route the global segments, the algorithm works with independent panels. In each panel, there are a set of global segments. The objective at this stage is to assign tracks to the segments. To achieve that, the authors introduce the concept of choices. A choice is a candidate solution to route a segment. The authors define a choice by a segment from a track of the panel plus the terminal connections. The terminal connections are any set of vias or wires that allow the connection of pins or other segments to the current segment. Following the definition, a segment has multiple alternatives for routing, each alternative represented by a choice. Figure 3.3 shows two global segments, one connecting pin A to pin B and the other connecting pin C to pin D. In this example, there are four tracks. The track assignment for connection AB is in orange, and for connection CD, it is in purple. The terminal connections are in red. Notice that the different tracks cause different terminal connections. Only two choices are shown for this example, although more choices could exist - and, in reality, probably *would*.

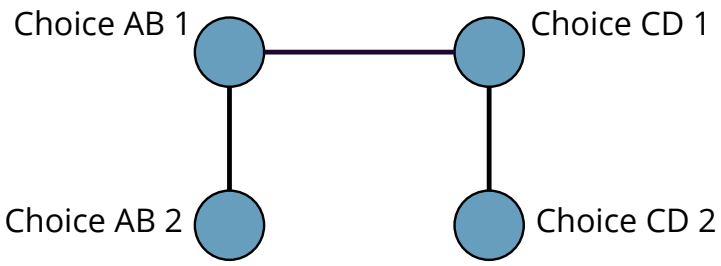
The essential concept of this formulation is conflict. Notice in figure 3.3 how choices AB1 and CD1 both wish to use a part of track 3. Physically, it is impossible to share the same

Figure 3.3: Two global segments within a panel with two choices each.



Source: From the author.

Figure 3.4: Conflict graph created from the circuit in figure 3.3



Source: From the author.

segment of a track in the same metal layer, and thus it is said that choice AB1 conflicts with choice CD1. One particular property of the choices model is that when one choice routes a particular segment, all other choices for the same segment are no longer relevant since the segment is complete. Successfully routing a segment using a specific choice means that all choices of the same segment conflict with each other. The authors describe a method of modeling this phenomenon in a conflict graph. In such a graph, each node corresponds to a choice, and edges connecting two nodes represent a conflict between the said nodes. To build this graph, a node is created for each choice of a segment. All nodes originated from the same segment are connected in a clique, since they all conflict with each other, as stated before. After translating all segments within the panel to this graph, it connects all conflicting choices in the circuit through an edge. Figure 3.4 shows the conflict graph for the circuit in figure 3.3. Once a choice is routed, all conflicting choices are now impossible to route and have their correspondence in the graph removed. For example, routing the choice AB1 renders routing choice AB2 and choice CD1 impossible because it is unnecessary since the segment doesn't require routing anymore. However, since this interferes with the choices of another segment selecting a node must be a careful

decision.

Modeling the problem as an instance of the MWIS is the proposed method by the authors. Each node has a weight, containing information about how important the segment is to route and how much interference the specific choice, represented by the node, will cause to other segments. The calculation of the weight follows equation 3.1.

$$W(v) = L + \alpha_1 * ||R|| + \alpha_2 * (\frac{\sum_{b \in B}(D_b)^2}{||B||}) + \alpha_3 * (F1 + F2) \tag{3.1}$$

$$\alpha_1 = 0.1 * (\frac{AvgD_b}{T_p})^2 \tag{3.2a}$$

$$\alpha_2 = 0.9 * \frac{1}{W_p} \tag{3.2b}$$

$$\alpha_3 = 0.3 * \frac{AvgS_p}{W_p} \tag{3.2c}$$

The equation terms represent as follows:

- L is the number of gcells that the segment spans, and its purpose is to add weight to large segments that are usually harder to route.
- $||R||$ is a component that contains information about the terminal connections. Longer terminal connections are less desired, but this term also considers non-preferential direction usage and via count.
- $\frac{\sum_{b \in B}(D_b)^2}{||B||}$ is the number of segments that cross the boundary of a G-Cell squared and divided by the number of boundaries. This term is used to add weight to segments that cross more utilized boundaries since these segments are harder to route.
- $(F1 - F2)$ is used to model flexibility. The algorithm prioritizes routing a segment using a choice that is incident to unassigned neighbor segments, which means that a routed segment will improve the routability of another segment.
- α_1, α_2 and α_3 are three experimentally derived coefficients. Their purpose is to increase or decrease the effect of the equation components. Their equations are listed in equations 3.2. $AvgD_b$ is the average segment density in the borders of all gcells within the panel, T_p is the number of tracks in each panel, W_p is the width of the panel, and $AvgS_p$ is the average space between tracks in the panel.

Thus, the conflict graph is now weighted. The algorithm's objective is to find the independent set that sums the highest weight, that is, solve the MWIS problem. Unfortunately, the MWIS problem is NP-Complete, so the authors propose a heuristic to solve it. The heuristic's intuitive idea is that routing a segment using a specific choice has a benefit to the routing proportional to the choice's weight in the graph. However, it renders it impossible to route any other choice that conflicts with it. Selecting a choice with a considerable weight that conflicts with multiple smaller weight choices may be undesirable, since routing all the conflicting choices could prove better globally. Therefore, the authors define the *benefit* of routing a segment using a choice $B(v)$ expressed in equation 3.3a.

$$B(v) = W(v) - \beta * W_i(v) - \gamma * W_o(v) \tag{3.3a}$$

$$\beta = 0.4 * \frac{W(v)}{max(Wi(v), Wo(v)} \tag{3.3b}$$

$$\gamma = 0.2 * \frac{W(v)}{max(Wi(v), Wo(v)} \tag{3.3c}$$

In these equations, the terms are as follows:

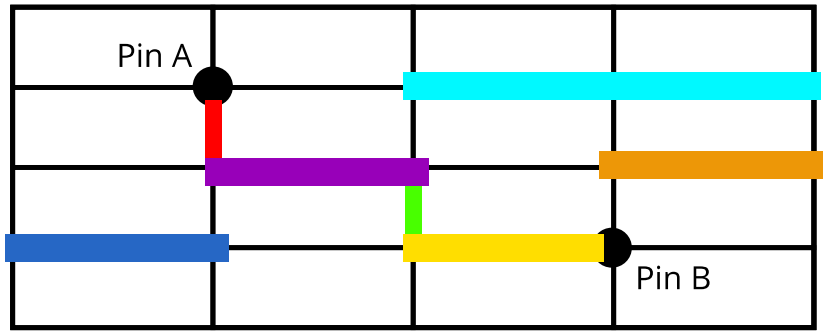
- $W(v)$ is the weight of the vertex v that represents the choice of the segment.
- $W_o(v)$ is the sum of the weights of all vertices originated from different segments that conflict with vertex v.
- β and γ are coefficients used to increase or decrease the significance of each term.

The algorithm calculates the benefit of every choice in the panel and sorts it using a heap structure. At every iteration, the algorithm selects the vertex with the highest benefit and routes its choice. Then, it is removed from the graph, alongside all nodes connected to it. The benefit value of the affected nodes is updated, and this process repeats. This heuristic causes the segments with high weight to get priority when deciding which choice will get the routing resources.

3.1.3 Partial Assignment

After routing using the MWIS solution, there may be multiple segments that were left unassigned to a track. The authors of RegularRoute propose a technique to assign only part of a segment to a track and connect parts of the same segment through a wire spanning

Figure 3.5: Example of a global connection routed by two partial assignments.



Source: From the author.

between the segments using the non-preferred direction. An example of a global segment routed by two partial assignments is in figure 3.5. In the figure, pins A and B cannot be connected by a single segment because of the three other segments routed in the same panel, namely the blue, orange, and cyan segments. However, it is possible to partially assign the segment from pin A through the red terminal connection to the partial purple wire. On the other hand, a partial assignment can be achieved through the yellow wire from pin B. These two partial segments, purple and yellow, can be connected through the green wire, thus successfully routing the global connection. The MWIS. problem can incorporate this partial assignment technique. According to the authors, though, the added extra vertices and edges could potentially cause a longer run-time, which is why the framework considers partial assignment only as a post-processing step.

3.1.4 Terminal Promotion

The RegularRoute framework assumes all pins are on metal 1. After considering layer 1, the algorithm may still not have assigned segments to a track. In this situation, the algorithm proceeds to the next layer, in this case, metal 2. The authors describe a peculiar situation where the algorithm leaves pins on layer 1. When the algorithm finally assigns the segment on metal 5, there is such a congested area above the pin that it cannot be attached to the segment. Whether this situation occurs or not in practice, the authors propose a terminal promotion technique, where pins left unattached in a particular layer are moved up using a via when considering the next metal layer.

In a way, keeping all unattached terminals from the previous layer in the current layer could address the problem of pins in higher metal layers. In this situation, the pins on metal 2, for instance, will only start being considered when the algorithm finally reaches

layer 2. After a segment is assigned, it checks if there are terminals on upper layers that need to be connected. If there are such terminals, the algorithm promotes the lower layer terminals to the upper layer. It is not described in detail which terminals would move up a layer in this situation, or even how many.

3.2 TritonRoute (KAHNG; WANG; XU, 2018)

TritonRoute is an initial detailed router (KAHNG; WANG; XU, 2018) similar to RegularRoute (ZHANG; CHU, 2011). The major contributions are an Integer Linear Programming-based algorithm to route panels and the more sophisticated interlayer routing. They divide the flow into intra-layer parallel routing and inter-layer sequential routing. However, before the actual routing occurs, a pre-processing stage aims to make the routing simpler by modifying the guides slightly.

3.2.1 Preprocessing

Before routing, TritonRoute processes the guides to generate uniformly wide guides with the largest possible lengths in the preferred direction. Three techniques are employed: splitting, merging, and bridging.

3.2.1.1 Splitting

A guide's width is its length in the orthogonal direction to the layer preferred direction. The tool divides any guide with a width larger than the unit width into a set of unit-wide guides. Figure 3.6 (b) shows the result of applying splitting in the configuration of (a).

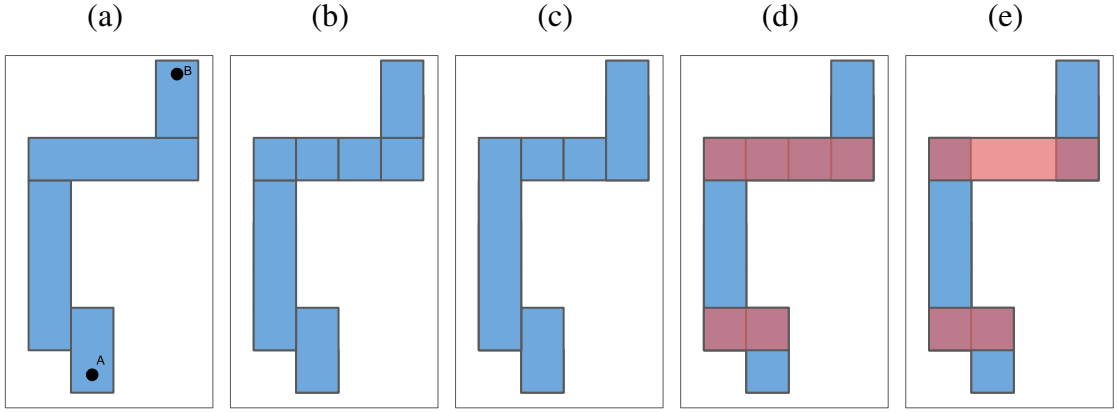
3.2.1.2 Merging

If two guides have edges that overlap in the direction orthogonal to the layer preferred direction, they are merged. Figure 3.6 (c) shows the result of applying merging in the configuration of (b).

3.2.1.3 Bridging

When two guides have touching edges in the direction parallel to the layer preferred direction bridges over with the help of an additional guide in a higher layer. Figure 3.6 (d) shows the result of applying bridging in the configuration of (c). After these three steps are applied, the tool discards any redundant guides. Figure 3.6 (e) shows the final result of the pre-processing applied in the initial configuration (a). With the guides pre-processed, the routing flow starts.

Figure 3.6: Preprocessing of routing guides. Blue rectangles are in metal 2, red rectangles are in metal 3.

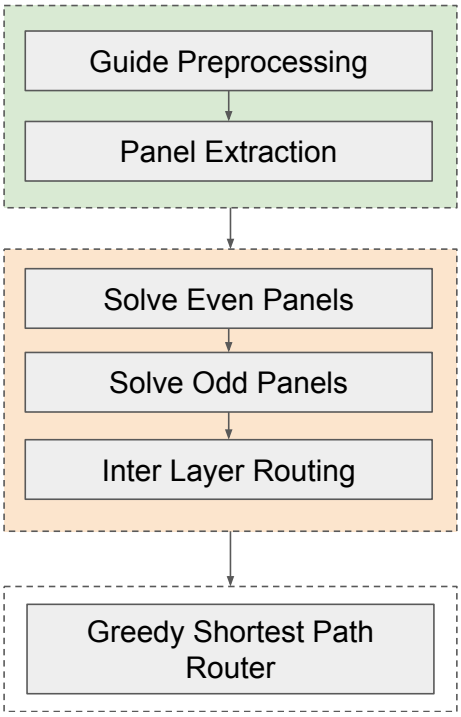


Source: (KAHNG; WANG; XU, 2018)

The overall flow is in Figure 3.7. For each layer, the same panel extraction of (ZHANG; CHU, 2011) occurs. From the grid of gcells, the algorithm extracts the lines (columns) if the layer's preferred direction is horizontal (vertical). These lines (columns) of gcells are the panels. The panels have indexes, starting from 0 for the bottom panel and increasing. First, the algorithm routes the even index panels, all in parallel, and then the odd index panels in parallel. This strategy allows for the algorithm to detect design rule violations in the boundary between two boundaries, which would not be possible if the tool was to route all panels in parallel. After routing every layer, the flow combines every layer's solution sequentially, creating the complete routing solution. An unspecified shortest path router greedily routes any net that remains unrouted.

All even index panels are then routed in parallel using ILP. To solve the Maximum Weighted Independent Set, first, the algorithm creates a conflict graph. In this graph, nodes represent different possible choices of wires to route a segment. All choices for the same segment conflict with each other, and choices from different segments conflict if they overlap. Furthermore, every choice has a weight, although the authors do not specify how to calculate it. The weight's purpose is to prioritize higher quality choices for a given

Figure 3.7: TritonRoute flow chart (KAHNG; WANG; XU, 2018).



Source: (KAHNG; WANG; XU, 2018)

Table 3.1: Notations used by (KAHNG; WANG; XU, 2018).

Notation	Rationale
$G(V, E)$	Conflict graph containing set of vertices V and set of edges E.
$v_{i,j}$	A vertex in the conflict graph representing the j^{th} choice of the i^{th} segment.
$e_{i,j}^{i',j'}$	An edge in the conflict graph indicating conflict between $v_{i,j}$ and $v_{i',j'}$.
$b_{i,j}$	Binary indicator of wether $v_{i,j}$ was used or not in the routing.
$w_{i,j}$	Weight of $v_{i,j}$.

segment and a difficult segment over a more straightforward segment. Table 3.1 shows the key terms used to create the graph.

The ILP formulation is in Equation 3.4, and it utilizes the definitions from Table 3.1. The objective is to maximize the weight $w_{i,j}$ of the used choices, which is why $b_{i,j}$ is 0 or 1 depending on whether it is in the solution or not. As for the constraints, every pair of vertices $v_{i,j}$ and $v_{i',j'}$ that have an edge $e_{i,j}^{i',j'}$ connecting them in the conflict graph cannot have $b_{i,j}$ and $b_{i',j'}$ 1 at the same time. That is, no vertices that conflict in the graph can be in the same solution.

$$\begin{aligned} &\textbf{Maximize:} && \sum w_{i,j} * b_{i,j} \\ &\textbf{Subject to:} && b_{i,j} + bi', j' <= 1, \forall e_{i,j}^{i',j'} \in E \end{aligned} \tag{3.4}$$

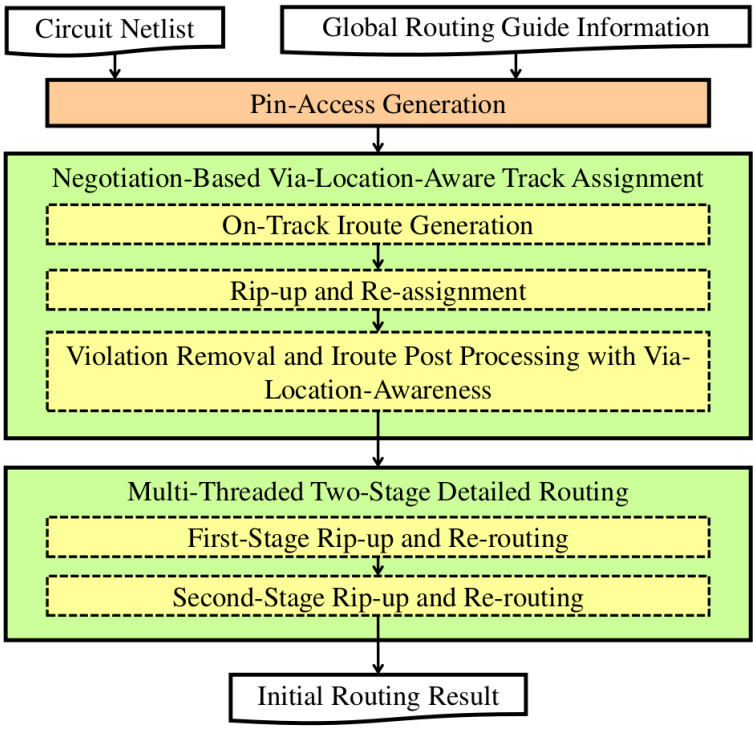
The output from the ILP formulation of the MWIS problem is the tracks that the global segments will use when routed. Sometimes a segment of a congested region may

have no track assigned. The authors do not specify how to handle these cases, but we assume the greedy shortest path router performs nets routing with unassigned segments. As for the wholly assigned nets, the algorithm finishes by inserting vias in the access points where two segments of the same net in adjacent layers intersect. The interlayer routing process is performed sequentially because there is no guarantee of data independence.

3.3 A Multithreaded Initial Detailed Routing Algorithm Considering Global Routing Guides (SUN et al., 2018)

In their work, the authors propose a framework for initial detailed routing (SUN et al., 2018). Their main contributions are a pin access generation algorithm, a track assignment technique, and a two-stage multithreaded initial detailed routing algorithm based on negotiation. Figure 3.8 illustrates the overall proposed flow. The inputs are the circuit netlist, the global routing guides, and the design rules - which they omitted in the flowchart. The following subsections explain all steps.

Figure 3.8: Overall flow of the algorithm (SUN et al., 2018).

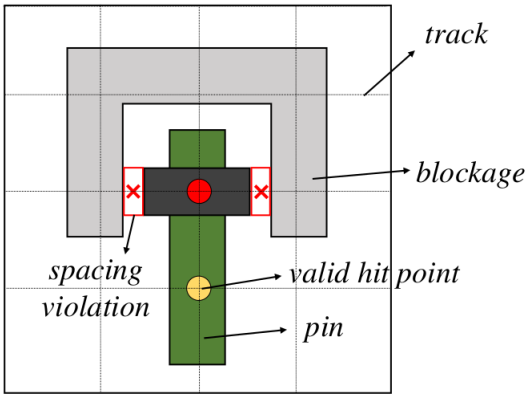


Source: (SUN et al., 2018)

3.3.1 Pin Access Generation

The first step that the algorithm takes is generating the pin access. The technique adopts the concepts of *hit point* and *valid hit point* from (XU et al., 2015) and rework the definitions slightly. They define a *hit point* as the *intersection of tracks with preferred direction on the current layer with the tracks with preferred direction on adjacent layers*. A valid hit point is *a hit point where the tool can insert a via without any design rule violations* (SUN et al., 2018). Figure 3.9 shows a pin shape with two hit points. The top hit point is not valid because inserting a via there would cause spacing violations with the surrounding structures. The bottom hit point, however, is a valid hit point, and a via can fit. The algorithm first processes each cell type and tries to find valid hit points within the pin shape. If any valid hit points exist, they are the candidate access points. If not, the pin shape is expanded one track space at a time until there is a valid hit point.

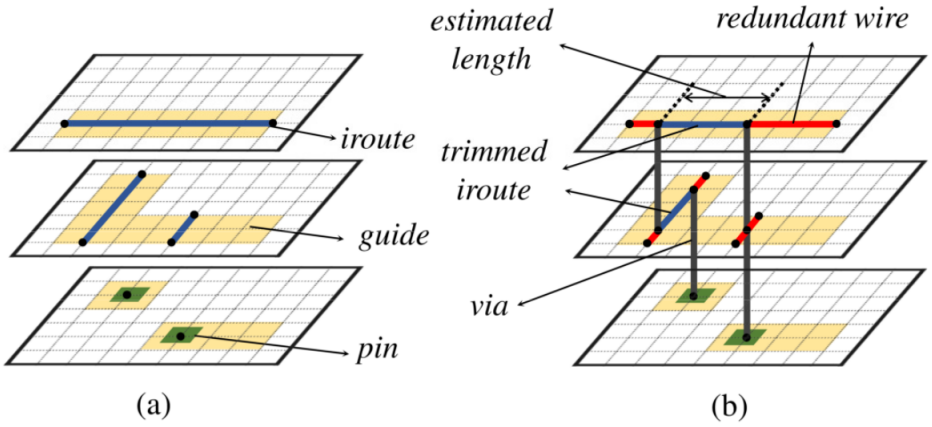
Figure 3.9: Pin access point extraction problem instance. Note that the hit point in red would cause two spacing violations with the blockage, but the valid hit point in yellow would not.



Source: (SUN et al., 2018)

After selecting a group of candidate pin access points, the algorithm attempts to remove points that are too close to one another while keeping as many access points as possible. It achieves that by modeling the problem as a conflict graph and solving the maximum weighted independent set. However, the authors do not clearly specify how to create the conflict graph or solve the MWIS problem.

Figure 3.10: Rudimentary routing produced by track assignment. Figure (a) shows only the track assignment, and figure (b) shows the track assignment and the vias, with redundant wires that are removed in red.



Source: (SUN et al., 2018)

3.3.2 Track Assignment

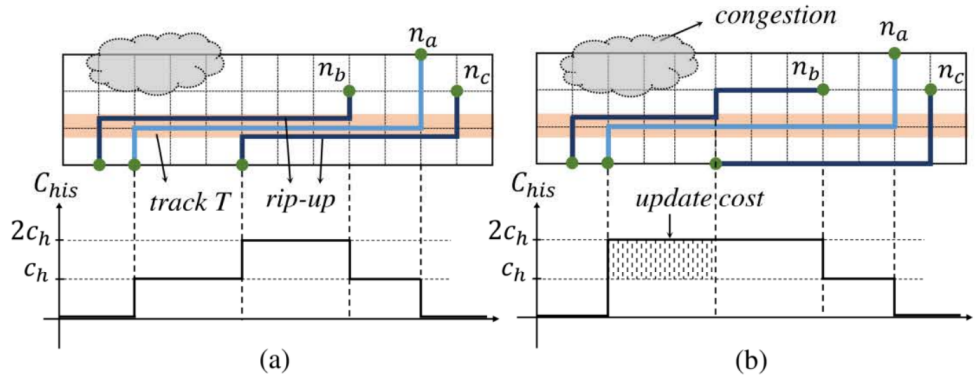
The next step in the flow is called negotiation-based Via-Location-Aware Track Assignment. The authors cleverly state that a partial routing solution can be obtained by simply selecting one track per routing guide. After inserting wires on the tracks selected, simply inserting a via where two wires intersect in adjacent layers produces a rudimentary routing. Figure 3.10 shows such routing in an example configuration. The authors call iroute a wire spanning through the entire guide. Notice how on (b) the algorithm can find the points where it can trim the iroutes based on the via positions. This track assignment technique is an important building block for the overall flow.

The algorithm removes all iroutes that overlap with blocked regions, and a conflict resolution technique handles the iroutes that overlap with other iroutes. The primary goal is to prioritize the longest iroute in a conflict instance. A maximum weighted independent set (MWIS) in an interval graph model based on a proven $\mathcal{O}(n \log n)$ complexity algorithm (HSIAO; TANG; CHANG, 1992). The weights of the vertices in the graph are the iroute lengths. After applying this technique, some nets are left disjoint.

3.3.3 Multithreaded Negotiation Based Detailed Routing

At this stage of the flow, some nets are complete, but some other nets are disjoint. To handle the disjoint nets, the authors applied an interval-based A* pathfinding algorithm

Figure 3.11: (a) Routing generated by an iteration of parallel A* routing and the cost of the track T calculated for its segments. (b) Cost update at the end of another iteration taking into account the historic cost of the segment that is still containing an overlap. (SUN et al., 2018)



Source: (SUN et al., 2018)

with multiple targets. If a net has n disjoint structures, one of the structures becomes the source for the pathfinding algorithm, and the other $n-1$ structures are the sources, causing the A* to be used $n-1$ times per net. All nets have their structures joined in parallel. After each iteration, track segments that contain two overlapping wires have their cost increased proportionally to the number of overlapping wires. The next iteration will consider the costs of using the track segment, which encourages the pathfinding algorithm to use alternative routes. Figure 3.11 shows in (a) a routing generated for three nets, all three using different segments of track T. At the bottom of (a), the cost calculation result shows that the cost increases by a unitary cost called C_h. The track segments with n wires have a cost increase of $(n-1) * C_h$. The algorithm then creates another interval graph for any track segment that has overlapping wires and attempts to keep the most critical wire and rip up the rest. The weights calculation follows equation 3.5. The α are weighting constants with unspecified values, L_{wl} is the length of the wire itself, K_{pin} is the number of pins of the net in question, and K_{fail} is the consecutive number of failures in the rip-up phase. The weight function has the objective of trying to prioritize nets that are more difficult to route. With the conflicts resolved, the flow proceeds to create a new routing for all paths that still need routing. In (b), a second routing appears with only one segment where three wires overlapped fixed. However, the overlap on the left remains a problem. The negotiation scheme solves this conflict by updating the cost taking into account the historical cost of the previous routing attempts. In this specific case, the cost increases by one C_h because two wires overlap at the particular track segment in question.

$$W_{mwis} = \alpha_{wl} * L_{wl} + \alpha_{pin} * K_{pin} + \alpha_{fail} * K_{fail} \tag{3.5}$$

The negotiation flow stops after eleven iterations, and the second stage of negotiation then starts and runs for two iterations - both numbers defined by the authors experimentally. The second stage's main difference is that wires are, instead of discouraged, completely prohibited from attempting to route through a track segment that another wire has claimed in a previous iteration. The goal is to explore paths that the first stage would hardly consider - if at all. According to the authors, the percentage of nets entirely routed by the first stage may reach over 95%. The authors do not explicitly state that the routing is complete after the second stage. However, their results by the ISPD 2018 metrics show that there were no open nets for all benchmarks.

3.4 MCFRoute (Jia et al., 2018)

MCFRoute is a detailed routing framework based on multicommodity flow (Jia et al., 2018). The main contributions of their work are:

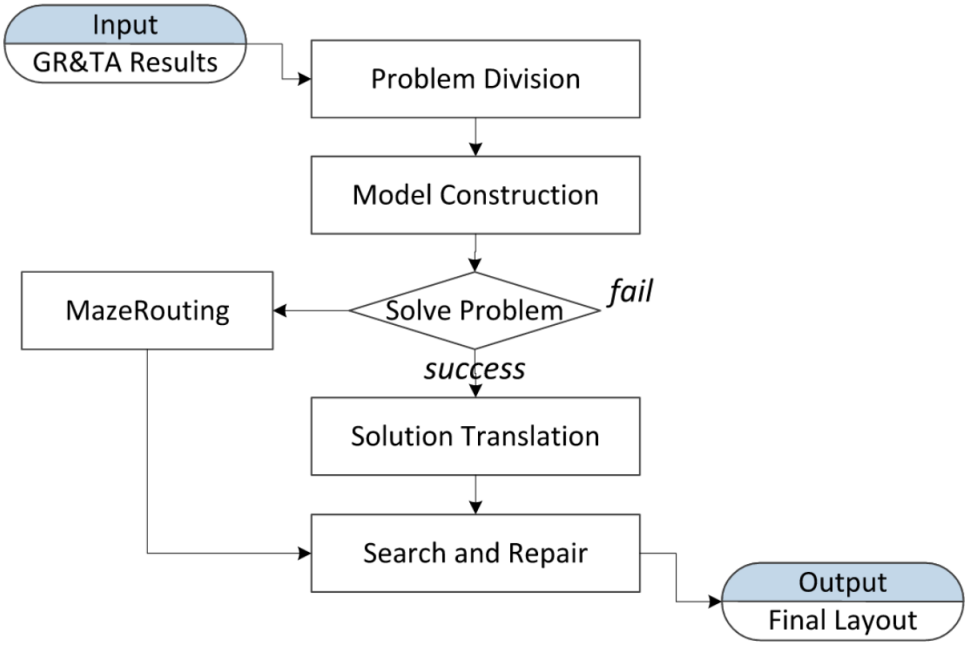
- A concurrent detailed router using the MCF model.
- Support to spacing rules for 28nm and above technologies.
- Heuristics to increase MCF model construction speed.
- An efficient algorithm to solve the MCF model.

The flow used is in figure 3.12. It is important to notice that their router requires both a global router and track assignment results. The step called Problem Division applies a G-Cell-based partitioning. The model construction step first formulates the detailed routing into an ILP problem using first a basic model formulation, then a design rule formulation that incorporates the rules into the model, and finally a model simplification is done. The next step is a solver. In some cases, the solver succeeds in solving the model. In this situation, the algorithm searches for violations, attempts to fix them and outputs the final layout. If it fails because it could not find a valid solution or ran out of time, the failsafe method is a maze router in the regions where it could not route.

3.4.1 Model Construction

The model construction step treats each net as a commodity with a corresponding command unit of flow. For each net, one component is treated as a source and the other

Figure 3.12: MCFRoute overall flow (Jia et al., 2018)



Source: (Jia et al., 2018)

as a target. During the execution, the flow will be shipped from the source to the target. Table 3.2 shows the notation used by the authors to explain the MCF modeling.

The edge capacity is the capacity that the routing graph edge allows to route through it. Since a detailed routing solution requires that each edge of the routing graph is occupied by at most one object, the capacity is either 1 when it is free or 0 when it is a blockage. The edge cost is a positive real number that depends on the edge type (via or wire) and preferred direction. The value of the flow command $d(k, v_j)$ depends on the vertex. If the vertex is a component of net n_k and this component is a source, the value

Table 3.2: Notation used in the MCF model by MCFRoute (Jia et al., 2018).

Notation	Rationale
$E/V/N$	Set of edges, vertices and nets indexed by i, j and k respectively.
E_{v_j}	Set of edges of vertex v_j.
$E_{v_{j,out}}$	Set of edges that start at vertex v_j.
$E_{v_{j,in}}$	Set of edges that end at vertex v_j.
$E^{\eta}v_j$	Set of via edges of vertex v_j.
$u(e_i)$	Capacity of edge e_i.
$c(e_i)$	Cost of edge e_i.
$d(k, v_j)$	Flow command of vertex e_i with value -1, 0 or 1 demanded by net n_k.
$f(k, e_i)$	Flow of net n_k by edge e_i.

of $d(k, v_j) = 1$. If instead the vertex is a component of net n_k but if the component is a target, the value of $d(k, v_j) = -1$. Otherwise, $d(k, v_j) = 0$. $f(k, e_i)$ assumes two possible values. If the edge e_i is being occupied by a component of net n_k, $f(k, e_i) = 1$, and if it is not occupied then $f(k, e_i) = 0$.

According to the MCF theory, the authors proceed to create a set of constraints. Equation 3.6 is the connectivity constraint and 3.7 is the capacity constraint. The first equation's interpretation is that the sum of all edges that go in the flow and the edges that go out of the flow for the same net must be equal to the value of $d(k, v_j)$, whatever that is. The second equation states that for every edge e_i, the sum of all nets usage must be equal to or less than the capacity, where e_i and $\bar{e}i$ are brother edges (the same edge in the graph but different directions). These equations are enough to produce a routing without opens, but it is not enough to avoid shorts. Equation 3.8 models the shorts, stating that every edge can only be used by either a connection (hence the 2) or is left unused. The authors also propose an equation that calculates the total cost of the routing. The cost equation is in equation 3.9, It is fairly straightforward. The total cost is the cost of every edge used.

$$\sum_{e \in E_{v_{j,out}}} f(k, e) - \sum_{e \in E_{v_{j,in}}} f(k, e) = d(k, v_j) \tag{3.6}$$

$$\sum_{k=1}^{K} (f(k, e_i) + f(k, \bar{e}_i)) \leq min\{u(e_i), u(\bar{e}_i)\} \tag{3.7}$$

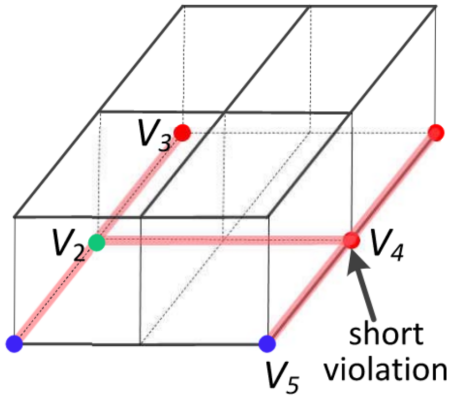
$$\sum_{k=1}^{K} \sum_{e \in E_{v_j}} f(k, e_i) \leq 2 \tag{3.8}$$

$$Cost = \sum_{k=1}^{K} \sum_{e \in E} (f(k, e) * c(e)) \tag{3.9}$$

With this final equation, the problem can finally be formulated as an ILP model. The model has to minimize equation 3.9 while subject to equations 3.6, 3.7 and 3.8. However, this formulation only works for nets with two components and where each component only covers one vertex. The authors prove that a set of components can be clustered into a super vertex and considered only one component without losing generality. As for the limit of two components per net, notice that it is equation 3.8 that starts to fail. Figure 3.13 shows an instance of this failure, where vertex V4 has three edges. The authors state that only 1% of the nets in the studied circuits have three or more components and propose to decompose these nets into multiple two-component nets using Steiner points.

This decomposition enables the flow to route all nets given enough time.

Figure 3.13: Multi-component net false short (Jia et al., 2018)



Source: (Jia et al., 2018)

3.4.2 Design Rule Modeling

One of the major contributions of MCFRoute is the incorporation of the design rules in the MCF modeling. The authors state that their model does not support some common rules in 45nm technologies and below, but it does cover some complex rules.

3.4.2.1 Spacing Rule Constraints

Some equations can assist in the task of modeling the most common spacing rules. For all nets $n_k \in N$ and vertices $v_j \in V$ the model includes equations 3.10 and 3.11. $E^{\eta}_{v_j}$ is a set of vias in vertex v_j. The function $\phi(k, v_j)$ simply indicates whether net n_k occupies vertex v_j. Analogously, the function $\eta(k, v_j)$ indicates whether or not net n_k occupies a via edge of vertex v_j. Also, equation 3.12 defined for any two vertices v_j1 and v_j2 with coordinates in the grid (x_1, y_1, z_1) and (x_2, y_2, z_2) respectively, represents the distance between them. If the two vertices are in different layers, represented by different z values, the distance doesn't matter for spacing rules. Otherwise, the distance is defined by the shortest distance between the x- and y-axis.

$$\phi(k, v_j) = \begin{cases} 1, \text{ if } \exists e_i \in E_{v_j} \text{ so that } f(k, e_i) = 1 \\ 0, \text{ otherwise} \end{cases} \tag{3.10}$$

$$\eta(k, v_j) = \begin{cases} 1, \text{ if } \exists e_i \in E^{\eta}_{v_j} \text{ so that } f(k, e_i) = 1 \\ 0, \text{ otherwise} \end{cases} \tag{3.11}$$

$$\Gamma(v_{j1}, v_{j2}) = \begin{cases} min(|x_1 - x_2|, |y_1 - y_2|) \text{ if } z_1 = z_2 \\ \infty, \text{ otherwise} \end{cases} \tag{3.12}$$

Using these three equations, some design rules can be modeled and verified. Assuming a fixed layer, let α_m be the wire width, β be the via enclosure length and γ_m be the metal spacing for the technology. For any vertex pair $(v_{j1}, v_{j2}) \in (V \times V)$ where $j1 \neq j2$, the authors introduce four situations:

- No violation: When $\alpha_m + 2 * \beta + \gamma_m \leq \Gamma(v_{j1}, v_{j2})$, v_{j1} and v_{j2} can be occupied by any type of geometry and still respect the constraints. This situation appears in figure 3.14.
- Enclosure to enclosure violation: If $\alpha_m + \beta + \gamma_m \leq \Gamma(v_{j1}, v_{j2}) < \alpha_m + 2 * \beta + \gamma_m$ there may be enclosure to enclosure violation if both v_{j1} and v_{j2} are vias. This situation appears in figure 3.14. The possibility of this violation requires the introduction of equation 3.13, which states that the two vertices in question cannot be both vias of different nets.
- Wire to enclosure violation: In a situation where the distance between v_{j1} and v_{j2} satisfies $\alpha_m + \gamma_m \leq \Gamma(v_{j1}, v_{j2}) < \alpha_m + \beta + \gamma_m$ both geometries involved must be wires. This situation appears inf figure 3.14. Equations 3.14 and 3.15 model this restriction. Remember that $\phi(k, v_j)$ and $\eta(k, v_j)$ represent the existence of a wire or via respectively in vertex v_j.
- Wire to wire violation: When the distance between v_{j1} and v_{j2} satisfies $\Gamma(v_{j1}, v_{j2}) < \alpha_m + \gamma_m$ and both vertices are occupied by any geometry there is a violation. This situation appears in figure 3.14 (d). This case requires the introduction of equation 3.16, denoting that the two vertices cannot be both occupied by wires of different nets.

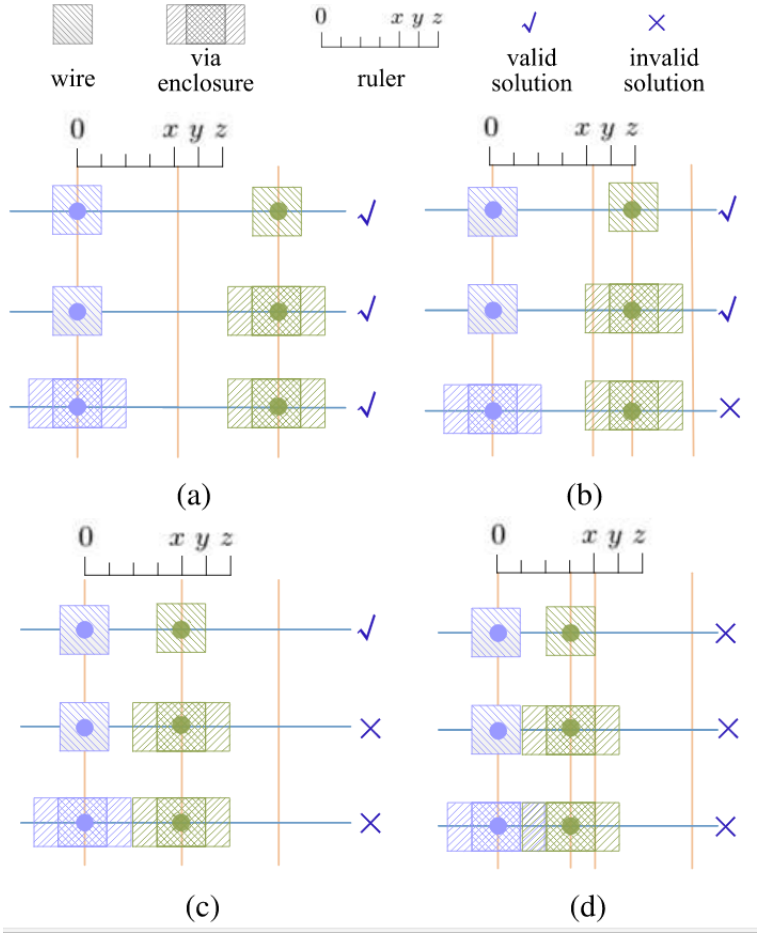
$$\eta(k_1, v_{j1}) + \eta(k_2, v_{j2}) \leq 1 \; \forall n_{k1}, n_{k2} \in N, k1 \neq k2 \tag{3.13}$$

$$\eta(k_1, v_{j1}) + \phi(k_2, v_{j2}) \leq 1 \; \forall n_{k1}, n_{k2} \in N, k1 \neq k2 \tag{3.14}$$

$$\phi(k_1, v_{j1}) + \eta(k_2, v_{j2}) \leq 1 \; \forall n_{k1}, n_{k2} \in N, k1 \neq k2 \tag{3.15}$$

$$\phi(k_1, v_{j1}) + \phi(k_2, v_{j2}) \leq 1 \; \forall n_{k1}, n_{k2} \in N, k1 \neq k2 \tag{3.16}$$

Figure 3.14: The four cases of spacing handled by MCFRoute (Jia et al., 2018). In (a), all configurations of wire and via are allowed because of the distance between the considered points. In (b), only the top two configurations are valid, because two via enclosures would cause a spacing violation. In (c), only two wires would not cause a violation, because a wire and a via enclosure would be too close. In (d), all configurations would cause spacing violation.



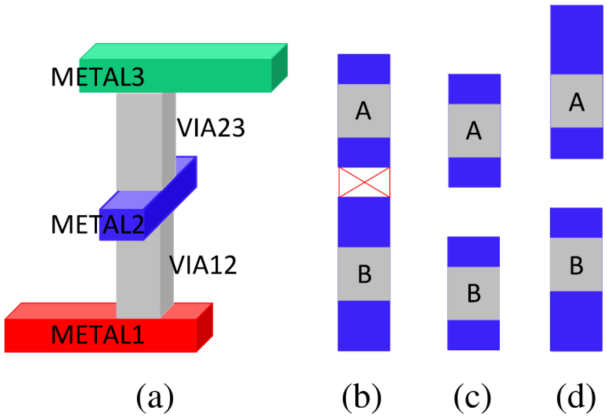
Source: (Jia et al., 2018)

The authors note that these equations can model the end-of-line spacing rule. To model this equation, it suffices to use the γ_{eol} instead of γ_m for the technology. Other spacing rules are left for the maze router and search and repair stages.

3.4.2.2 Minimum Area

Because the wire width does not change for a fixed layer, the authors model the minimum area using a minimum length constraint. Instead of adding equations to the model that would, allegedly, cause the runtime to be unacceptable, the authors leave this verification after the ILP solver has defined all the wires. Even if the algorithm were to check all wire lengths, there is still one specific case where a minimum area violation could occur. The authors describe a situation where the layout of a tower of vias would incur a violation of minimum area, and the algorithm decides to increase the size of this via enclosure. Sequentially, another tower of vias of another net has its layout created, and it is impossible to extend its area to avoid a minimum area violation. This situation appears in figure 3.15. In (a), a tower of two vias span through layers 1, 2, and 3. The via enclosure in the middle would violate the minimum area constraint. However, instead of simply increasing its size on the fly, which could cause the situation illustrated in (b), the algorithm can leave the via unattended as in (c). After all the vias are inserted properly, a more precise algorithm can increase the area of the enclosures while paying attention to the structures nearby, reaching the result in (d). The authors state that this technique can resolve 93.2% of the minimum area violations in their experiments.

Figure 3.15: Tower of vias minimum area violation handling by MCFRoute (Jia et al., 2018). In (a), a tower of vias where the metal 2 via enclosure in the middle of the tower is could cause a minimum area violation. In (b), a configuration where an attempt to patch the wire to occupy more area would cause a violation. In (c), the enclosures were not modified. In (d), both enclosures were patched with metal, but without causing violation.

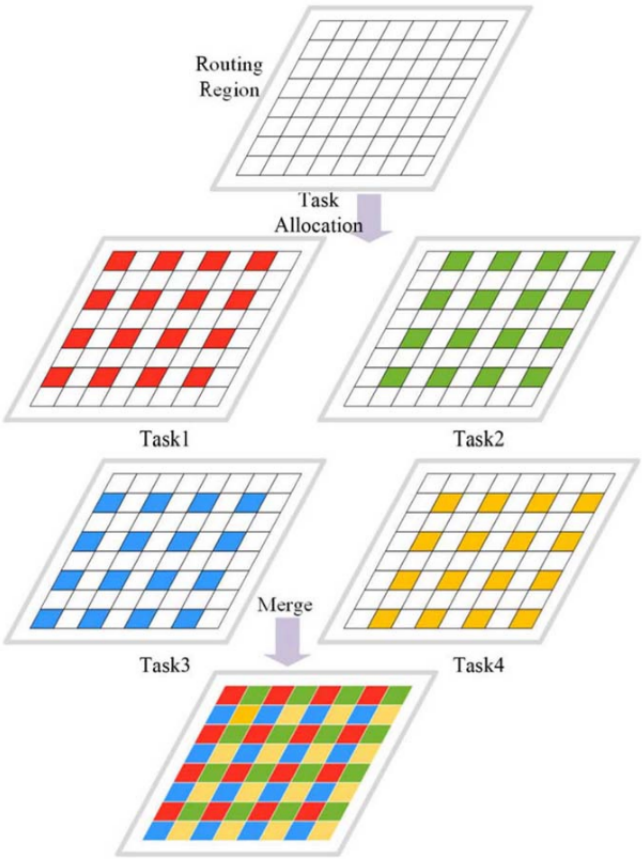


Source: (Jia et al., 2018)

3.4.3 Multithread Strategy

One of the significant contributions of MCFRoute is the multithreaded strategy. The authors cleverly state that there is a dependency problem in the boundaries of two adjacent regions. This conflict with the boundaries would cause a high overhead to a standard parallelization based on routing regions. What the authors propose is a four-stage strategy where, in each stage, the algorithm can route a set of independent routing regions. Figure 3.16 shows this strategy, where the flow partitions the initial grid of routing regions into four tasks, sets of regions that are not adjacent to each other. By sequentially routing each task set, it is possible to parallelize the routing greatly without conflict overhead. The runtime scaling is very good, achieving $7.82\times$ speedup with eight threads.

Figure 3.16: Four stage multithreaded strategy employed by MCFRoute (Jia et al., 2018). The entire task is divided into four parallel workloads, and the results of the four are then merged to produce a final solution.



Source: (Jia et al., 2018)

3.5 SmartDR (GONÇALVES; JR; MARQUES, 2020)

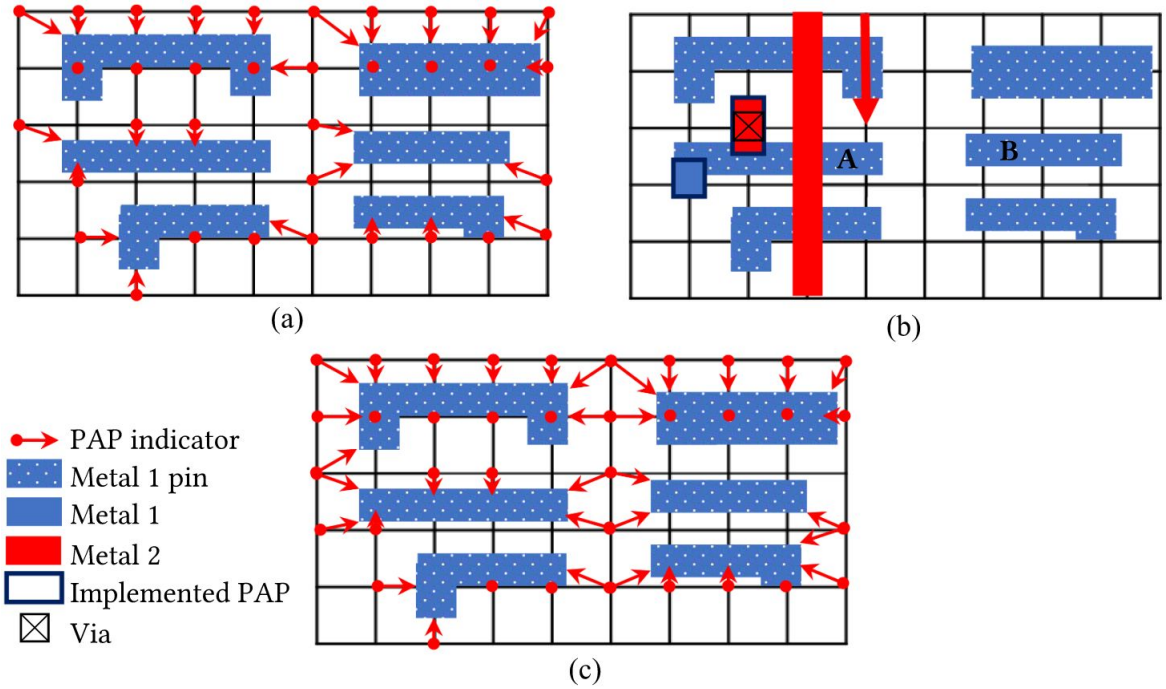
In (GONÇALVES; JR; MARQUES, 2020), the authors present a set of techniques and algorithms for initial detailed routing, in practice, a complete initial detailed routing tool. The main contributions are a pin access methodology and a special A* implementation that is design rule aware.

3.5.1 Pin Access

The authors cleverly point out that there are only a few layouts out of all instances of cells in the design. Out of these base cells, there are a small number of alignments the instances can have with the grid. The pin access methodology they propose processes all cells with all possible alignments with the grid and produces Pin Access Path (PAP) sets for each configuration. Consider figure 3.17, extracted from (GONÇALVES; JR; MARQUES, 2020). In (a), the figure shows the Simultaneously Valid Solution (SVS) for this configuration, denoting all possible PAPs that would not cause any design rule violation and could all coexist simultaneously. In (b), the figure illustrates a situation in which a path in metal 2 cannot reach the implemented PAPs. In (c), the figure shows an extension of the concept of SVS, where some grid nodes can access multiple pins, but only one at a time. The authors call this phenomenon Resource Sharing. The authors propose that these RS SVS live as Ghost PAPs, defined as PAPs created by the pre-processing but not yet implemented physically. The strategy proposed utilizes these ghost PAPs as different possibilities for pin access. According to the authors, the ability to choose a PAP on-the-fly increases the flexibility of the solution.

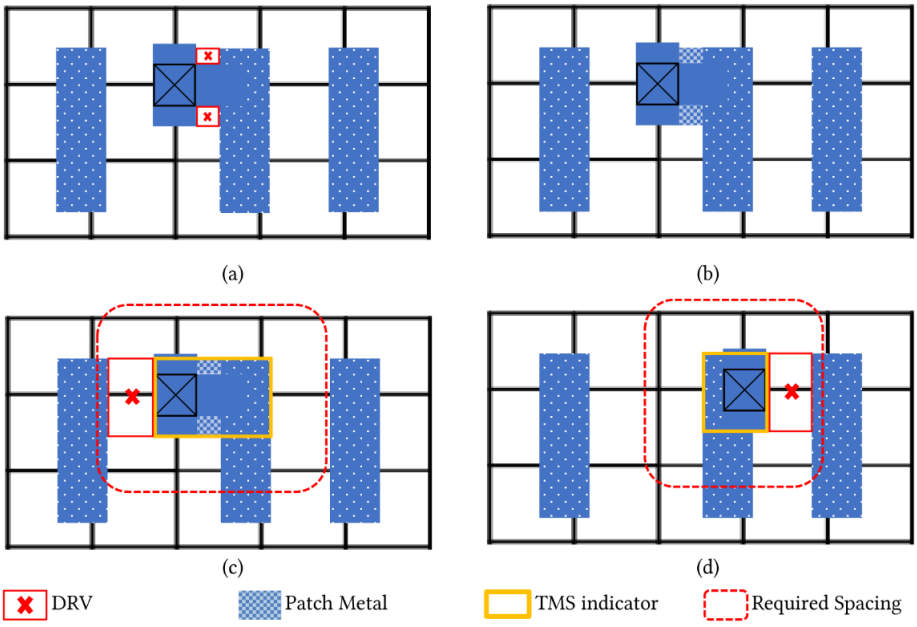
The authors elaborate further on the subject of pin access. Figure 3.18, extracted from (GONÇALVES; JR; MARQUES, 2020), shows an attempt of creating a valid PAP to the middle pin of the cell in the given grid alignment. The authors state that using a PAP on the left and a wire connecting the pin to the via would cause two spacing violations, shown in (a). In (b), the authors show that a patch's insertion would solve these two spacing violations. However, this patch's insertion would create a particular metal configuration that the authors called Thick Metal Shape (TMS). This TMS would alter the spacing rules, causing another violation shown in (c). In this specific example, the PAP utilizing a via to the right would also cause a violation because of the TMS, shown in (d). These two PAPs are therefore not legal.

Figure 3.17: Pin Access Path (PAP) processing as proposed by (GONÇALVES; JR; MARQUES, 2020): *"in access situations. (a) A SVS, denoted by the PAP locations and their respective pins. (b) A path (red arrow) in metal 2 tries to connect to pin A but cannot reach the implemented PAPs (we are assuming metal 3 is unreachable here). (c) A pin access solution with resource sharing"*



Source: (GONÇALVES; JR; MARQUES, 2020)

Figure 3.18: Patch metal insertion and TMS (Thick Metal Shape) calculation as proposed by (GONÇALVES; JR; MARQUES, 2020). Figure (a) shows one possible via location, and two violations it would cause. Figure (b) shows that two metal patches would solve these violations. Figure (c) shows that the patch created a TMS, and now the special spacing rules for wide objects cause a violation with the adjacent shape. Figure (d) shows that the other possible via location would also cause a spacing violation.



Source: (GONÇALVES; JR; MARQUES, 2020)

3.5.2 Design Rule Aware Path Search (DRAPS)

This work's other major contribution is the proposal of an adapted path search that incorporates design rules and the PAPs in the algorithm. The algorithm's general working is analogous to the A*, but the authors state that the $h(n)$ (the estimation of distance from a node to the target) can be optimized. When the $h(n)$ is optimistic, A* is guaranteed to find the optimal path. However, the less optimistic it is, the faster the search ends (GONÇALVES; JR; MARQUES, 2020). The authors point that when using the Manhattan distance as $h(n)$, it becomes too optimistic, mainly because the guides are ignored completely. The authors propose using a processing of the guides to generate a function called Tunnel Lower bound (TL), which is the minimum distance between the two points when constrained by a tunnel, in this case, the guides.

Consider the definitions in table 3.3 (GONÇALVES; JR; MARQUES, 2020), with the definitions used by the authors. The algorithm proposed precomputes the TL for each TR before the path search. Then, during the path search, it calculates the $h(n)$ using equation 3.17. The authors state that the operation of iterating over the rectangle's RPs is of complexity $\omega(n)$, but that n in practice is very small. Finally, the authors state that the TL's precomputation for each TR is performed by a TL-aware implementation of the Dijkstra algorithm.

$$h(n) = \min\{lb(n, r), TL(r)\}, \text{ where r is a RP of rect(n)} \tag{3.17}$$

Given the optimized $h(n)$, the authors describe the design rule awareness. The proposed algorithm is aware of the via library, the minimum area rule, and the cut to cut spacing violations in the same path, not between two different paths.

Table 3.3: Definitions used by (GONÇALVES; JR; MARQUES, 2020) to compute the Tunnel Lower-bound (TL).

Notation	Definition
$rect(n)$	The tunnel rectangle (TR) that contains the point n.
$lb(a, b)$	The Manhattan Distance between a and b modified by the via cost when the points are in different layers.
RP	A reference point n, stored in a TR, used for the TL calculation during the path search. It has an associated target point and a $TL(n)$
$TL(n)$	TL from the RP n to its corresponding target point.

3.5.3 Via Library

The input files define a set of vias that can be used. If the algorithm waits to select a via after the full path has been defined, it may be impossible to find a via that would not generate design rule violations (GONÇALVES; JR; MARQUES, 2020). The authors state that their methodology selects the via that has the smallest footprint in the non-preferred direction, giving preference to the bottom layer.

3.5.4 Minimum Area

Similarly to the via library, waiting to post-process minimum area shapes may find a situation where it is impossible to extend wires without causing further violations (GONÇALVES; JR; MARQUES, 2020). The proposed methodology is to check if the wire in the previous layer has the minimum area whenever a layer change occurs. If it does not, the wire is extended in the same direction until it meets the minimum area.

3.5.5 Cut Spacing

The authors propose a simple change in the path search algorithm that can reduce cut-to-cut spacing in the same net vias. Whenever a layer change occurs, the strategy is to backtrack through the path, searching for the first via it finds. When it does, it checks if the distance between the first found via and the current via would respect the spacing requirement.

3.6 Dr. CU 2.0 (LI et al., 2019)

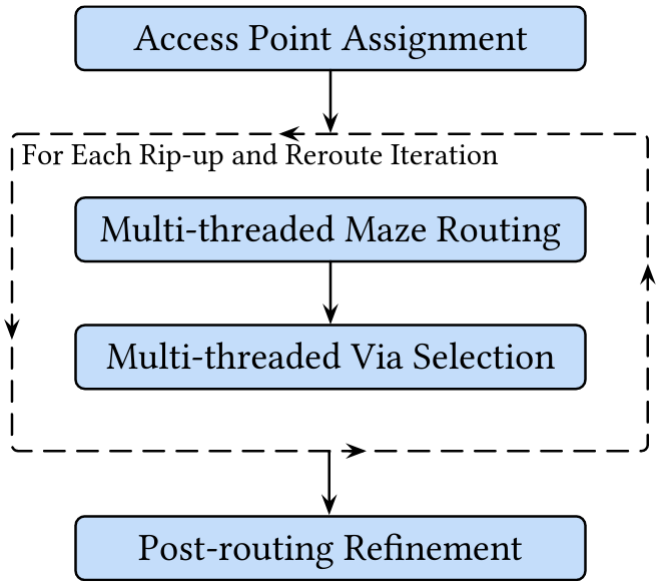
In their work, the authors present a detailed routing framework. The authors state that their major contributions are:

- A pre-processing to compute valid access points for pins with off-track via support.
- A design rule-aware maze routing that supports end-of-line spacing
- A post-processing that fixes corner-to-corner spacing

Figure 3.19 shows the overall flow of their work. There are four steps, Access Point As-

signment, Milth-threaded Maze Routing, Multi-threaded Via Selection, and Post-Routing Refinement. We will study each step further in the following subsections.

Figure 3.19: Routing flow proposed by (LI et al., 2019)



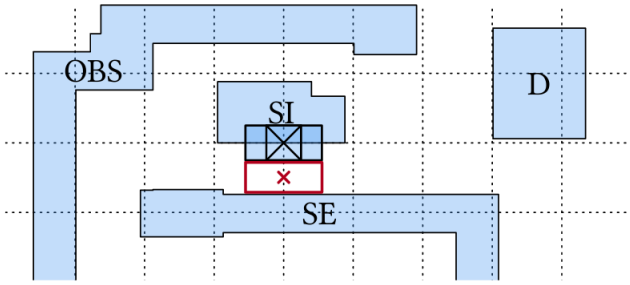
Source: (LI et al., 2019)

3.6.1 Access Point Assignment

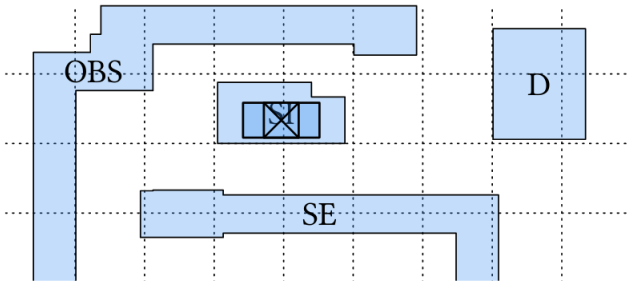
This step of the routing flow proposed by (LI et al., 2019) is about finding points in the routing grid around pins that can connect the net structure to the pins. The authors point that the number of routing tracks in a given area decreases with technology nodes' advancement. In fact, for some pins in specific configurations, there is no violation-free same-layer on-grid point that can access the pin (LI et al., 2019). Figure 3.20 shows an example configuration that is affected by this phenomenon. (a) shows that no on-grid via can access the pin labeled SI with no violations, and a highlight shows the spacing violation between the example via and the pin labeled SE. In (b), the authors show that an off-track via can access the pin labeled SI with no violations. The authors' methodology proposes that the upper layer's grid points can be used instead of the bottom to find these off-grid points to access the pins.

Pins that can only be accessed with off-track vias have the pin accessing via created before the maze routing and are not moved during the rip-up and reroute. Pins that can be accessed normally by on-grid vias have their access points stored as possible paths to connect the rest of the net's routing to the pin.

Figure 3.20: Off-track Pin Access (LI et al., 2019)



(a) Pin `SI` has no violation-free same-layer access point.



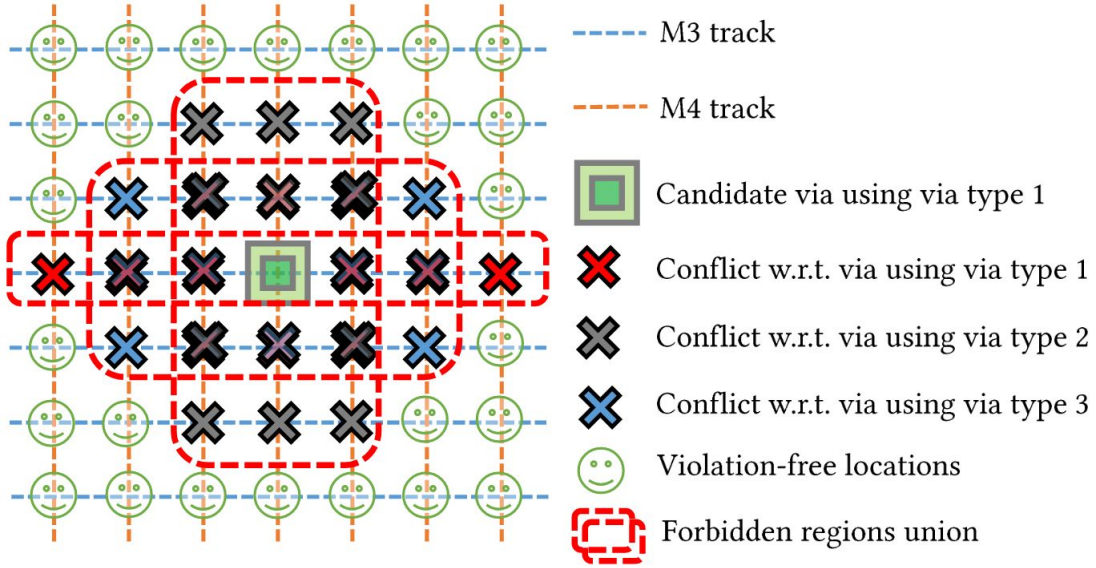
(b) Connect `SI` to upper-layer access points with an off-track via.
Source: (LI et al., 2019)

3.6.2 Multi-threaded Maze Routing and Via Selection

During these steps, a local grid graph is generated for each net based on the guide's location. The authors state that for each iteration, this local grid graph expands in both directions to provide more room for the search. When the number of violations in a part of the local grid graph exceeds a certain threshold, it extends to adjacent layers as well (LI et al., 2019). The maze router runs constrained on this local grid graph, and nets whose local grid graph do not overlap are considered independent work units for multithreaded routing. The via selection technique utilizes a construction of look-up tables (LUTs). The authors state that creating a LUT of vias can speed up the process of inserting vias by categorizing the via-related spacing violations for each via in the library. Figure 3.21 shows an example of LUT for a candidate via of type 1, with the conflict regions with respect to other types of vias color-coded. When considering a via to be placed in this candidate's vicinity, the LUT is consulted to choose the type of the second via. LUTs are also created with configurations of via-to-wire and via-to-obstacle. The authors do not specify how the configurations are obtained and how many there are.

During the maze routing, the authors consider the end-of-line (EOL) violation. There are two different constraints: EOL with parallel edges (EWP) and without parallel edges (EWOP). The EWP constraint is larger than the EWOP. However, the authors pro-

Figure 3.21: Via to via LUT (LI et al., 2019). The candidate via point in the middle would cause different violations patterns depending on the via type selected.



Source: (LI et al., 2019)

pose that only the largest one is considered by the maze router, as the added complexity of handling EWP conditions is not beneficial enough (LI et al., 2019).

3.6.3 Post-routing Refinement

The authors state that the via type selection is order-sensitive. Therefore, a round of via type selection attempts to change the vias' types after the maze routing is complete to select the via types with the lowest violation count for each location. Corner-to-corner (C2C) spacing violations are also considered for post-routing refinement. For C2C, the authors only addressed the cases of via and wires causing violations, ignoring cases of pins. They state that most pins are in metal 1 and that there are rarely any C2C rules for the bottom layer as their reasoning not to handle this case (LI et al., 2019). As for the other cases, the authors propose using the same methodology of LUT.

3.7 Summary

Table 3.4 shows the results of (ZHANG; CHU, 2011) compared with MCFRoute and WROUTE. Based on this comparison, we compared the three algorithms in terms of runtime, the number of violations produced, the number of vias inserted, and the total

Table 3.4: Results of (ZHANG; CHU, 2011), MCFRoute and WROUTE in the ISPD 2005 benchmark suite (NAM et al., 2005).

	(ZHANG; CHU, 2011)				MCFRoute				WROUTE			
Benchmark	#vio	wl$\times 10^7$	#via$\times 10^6$	cpu	#vio	wl$\times 10^7$	#via$\times 10^6$	cpu	#vio	wl$\times 10^7$	#via$\times 10^6$	cpu
adaptec1	0	8.4	1.5	622	0	12.5	2.3	22000	0	8.5	1.5	1201
adaptec2	0	10.2	1.9	558	0	15.1	2.6	27000	221	10.4	2.0	1344
adaptec3	0	21.8	3.5	1176	-	-	-	-	0	22.1	3.6	3939
adaptec4	4	19.8	3.0	1330	-	-	-	-	324	20.4	3.2	4424
adaptec5	14	46.6	6.9	2844	-	-	-	-	294	47.2	7.2	7729
newblue1	0	8.8	2.3	297	-	-	-	-	0	9.1	2.4	914
newblue5	6	46.3	7.2	2654	-	-	-	-	287	48.8	7.8	7097
newblue6	0	39.9	8.5	2445	-	-	-	-	0	41.2	9.0	6645
bigblue1	0	9.8	2.2	811	0	16.3	2.9	32000	0	9.7	2.2	1802
bigblue2	0	21.2	3.7	1177	0	24.1	5.0	43000	54	22.0	3.9	2856

Table 3.5: Results of and (KAHNG et al., 2011), (SUN et al., 2018), (GONÇALVES; JR; MARQUES, 2020) and (LI et al., 2019) in the ISPD 2018 benchmark suite, as reported by (GONÇALVES; JR; MARQUES, 2020)

	(KAHNG; WANG; XU, 2018)		(SUN et al., 2018)		(GONÇALVES; JR; MARQUES, 2020)		(LI et al., 2019)	
Benchmark	Score$\times 10^6$	runtime	Score$\times 10^6$	runtime*	Score$\times 10^6$	runtime	Score$\times 10^6$	runtime
test1	0.38	154	0.36	33.8	0.32	16	0.29	31
test2	5.62	1399	5.53	346	5.00	71	4.70	197
test3	6.98	2335	6.71	460	5.71	157	5.37	532
test4	25.8	9972	37.0	1210	16.5	314	15.6	2761
test5	21.6	3705	51.0	1451	17.6	208	16.3	1643
test6	29.0	6124	73.3	3168	23.5	286	21.6	1444
test7	50.6	10994	124	8586	43.3	429	38.4	3172
test8	49.6	9793	125	8295	45.9	435	38.5	2991
test9	41.8	9119	117	5657	39.1	405	33.1	1924
test10	56.3	16421	234	6553	72.4	717	41.9	5212
Average	28.7	7442	77.4	3576	26.9	303.8	21.5	1990

wire length created. (ZHANG; CHU, 2011) is the fastest, generates the fewest violations, and uses the least amount of vias and wire resources.

Table 3.5 shows the results of (KAHNG; WANG; XU, 2018), (SUN et al., 2018), (GONÇALVES; JR; MARQUES, 2020), and (LI et al., 2019). The times were reported by (GONÇALVES; JR; MARQUES, 2020) where the authors claim to have obtained the binaries and executed all tools in the same machine, therefore obtaining a supposedly scaled result, except (SUN et al., 2018) that was not available. The score obtained for the contest benchmarks is a weighted equation of different metrics. The table presents the score and runtime, and while these are not enough to visualize differences in specific metrics, the score gives a rough estimate of the quality of the result. According to the results, the best scoring tool in average is (LI et al., 2019), followed by (GONÇALVES; JR; MARQUES, 2020), (KAHNG et al., 2011) and finally (SUN et al., 2018). As for runtime, (GONÇALVES; JR; MARQUES, 2020) is much faster than the other tools, followed by (LI et al., 2019), and (KAHNG et al., 2011). The runtime of (SUN et al., 2018) was not obtained in the same machine; therefore, it would not be a fair point of comparison.

Table 3.6 compares the studied frameworks concerning several functionalities and data structures. (ZHANG; CHU, 2011) states in their work that the flow can perform

Table 3.6: Comparison between the studied frameworks.

	(Jia et al., 2018)	(ZHANG; CHU, 2011)	(KAHNG; WANG; XU, 2018)	(SUN et al., 2018)	(GONÇALVES; JR; MARQUES, 2020)	(LI et al., 2019)
Layer Assignment	Yes	No	Yes	Yes	Yes	No
Track Assignment	Yes	No	No	No	No	No
Out of Grid Structures	No	No	Yes	Yes	Yes	Yes
Multithreaded Data Set	Routing Region	Panel	Panel	Net	Batch	Net
Conflict Modeling	MCF	Conflict Graph	Conflict Graph	Interval Graph	Resource Sharing	LUT
Conflict Resolution	MCF	MWIS	MWIS	Negotiation and MWIS	Exclusion	-
Model Solver	ILP	Benefit Heuristic	ILP	Hsiao:1992:EAF:144207.144222	-	-

layer assignment, and (LI et al., 2019), while using the ISPD formulation that includes layer assignment, formally does not constrain the routing to the layers. Track assignment is a requirement only for (Jia et al., 2018). Both (Jia et al., 2018) and (ZHANG; CHU, 2011) need structures to be aligned to the grid, although the authors briefly talk about the necessity of off-grid vias to access pins. In modern frameworks, multithreaded capable algorithms can reach better runtimes. The data set that the multithreaded algorithms use varies. (Jia et al., 2018) creates four sets of routing regions so that each set contains no neighbor routing regions within the same set. (ZHANG; CHU, 2011) and (KAHNG; WANG; XU, 2018) use the same panel strategy for global segments. (GONÇALVES; JR; MARQUES, 2020), and (LI et al., 2019) use their respective modeling of the net topology as a basis to determine which nets can be routed in parallel. (SUN et al., 2018) uses a net as a data set and handles the data dependency afterward. The conflict modeling of (Jia et al., 2018) bases itself on MCF theory. Both (ZHANG; CHU, 2011) and (KAHNG; WANG; XU, 2018) use conflict graph, while (SUN et al., 2018) uses an interval graph to model the conflicts. To solve the conflict, MCF uses the MCF theory. (ZHANG; CHU, 2011) and (KAHNG; WANG; XU, 2018) extract the MWIS from the conflict graph, and (SUN et al., 2018) uses a combination of both MWIS and negotiation. As for solving the model, (Jia et al., 2018) and (KAHNG; WANG; XU, 2018) use an ILP formulation to solve the MCF and MWIS, respectively. (ZHANG; CHU, 2011) solves the MWIS problem using the benefit heuristic, and (SUN et al., 2018) solves the instances of MWIS using (HSIAO; TANG; CHANG, 1992). (GONÇALVES; JR; MARQUES, 2020) uses resource sharing (RS) as a conflict model on pin access and solves the conflicts by simply selecting one and excluding the others, with no engine. (LI et al., 2019) uses a look-up table-based scheme to consult on possible conflicts, specially vias, and does not use an engine.

4 MIXED ROUTING FRAMEWORKS

Some of the literature works propose either performing global and detailed routing together in a flow while exchanging information, and some others propose performing full circuit routing without dividing the routing hierarchically. We will address these specific cases in the following sections, although the latter is inefficient and struggles to meet timing constraints while minimizing violations.

4.1 Qrouter (QROUTER, 2017)

Qrouter is a routing tool part of the Qflow toolchain (QROUTER, 2017). It bases itself on Lee's algorithm (LEE, 1961), also known as maze router. The algorithm does not explicitly perform global routing and detailed routing; instead, it performs full circuit routing from the start. The employed algorithm expands on Lee's algorithm because it can handle multiple sources and multiple terminals for a connection. One pin of each net becomes a source, and the others become targets. When the algorithm finds a path between the source and one target, the path is committed and becomes part of the source for new searches to the other targets.

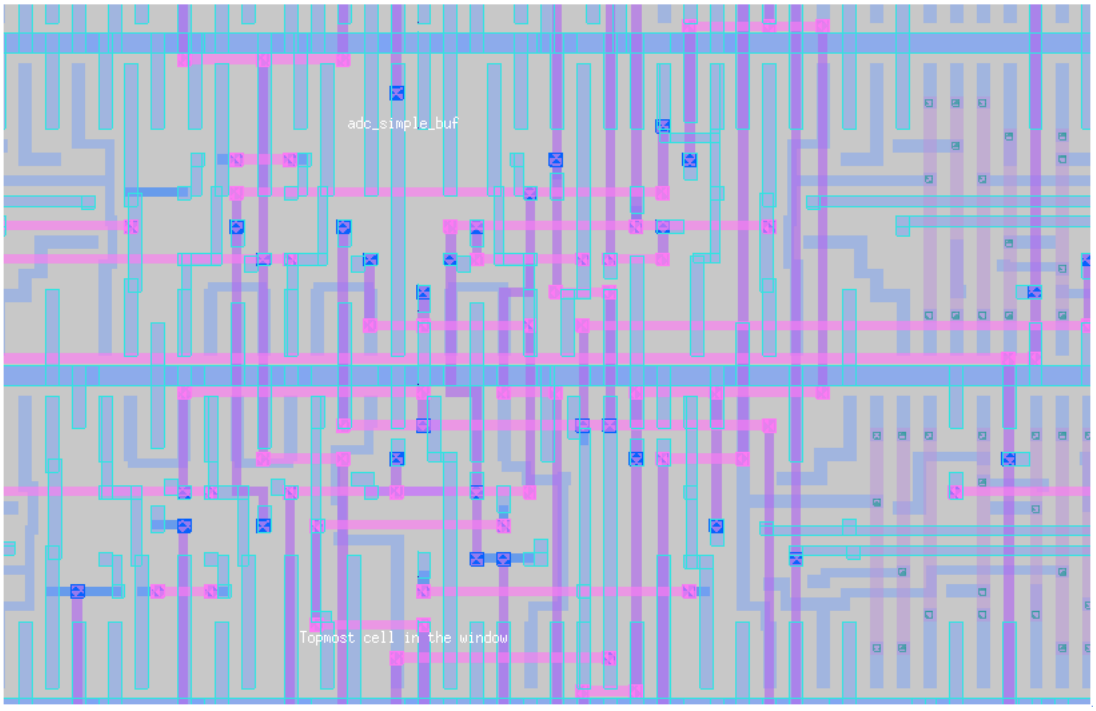
Because the maze router is too slow for full circuit routing, the authors developed a hybrid technique. It first computes an optimal trunk-and-branches solution. This solution is used as an initial routing for each net, and the terminals of each net are attached to the trunk or branches in an ad-hoc fashion. After the initial solution, the flow applies rip-up and reroute of nets that have a short. After a certain predefined number of iterations, the nets that are still unrouted are routed using the maze router.

Figure 4.1 shows the snippet of a routing solution produced by Qrouter. It is worth noting that the routing produced is very dense but seems to be using only three metal layers based on the color-coding. The authors themselves state that the solution relies on the placement quality, and it does not alter placement to improve routability.

4.2 GDRouter (Zhang; Chu, 2012)

GDRouter is a framework for complete routing by interleaving global routing, and detailed routing in a flow (Zhang; Chu, 2012). The flow employed is in figure 4.2. In the

Figure 4.1: Example circuit routed by Qrouter (QROUTER, 2017)

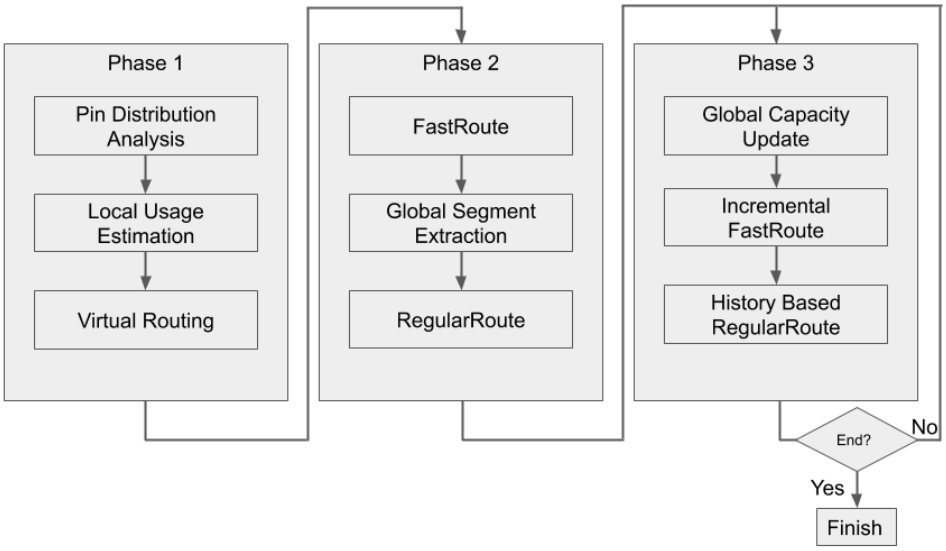


Source: (QROUTER, 2017)

first phase, called "Initial Capacity and Routing Weight Estimation," the pin distribution analysis is done using the gridded Voronoi diagram. The calculation uses the pin density of the G-Cell weight. The resulting weight is useful in global routing. Then, the flow attempts to estimate local G-Cell utilization. Two cases require attention: first, when a net is entirely in one G-Cell, it is necessary to estimate the resource usage. Second, a global net that spans multiple gcells will utilize resources within the gcells where its pins reside. The flow uses the spine router to estimate these two resource usages. It strongly resembles the vertical trunk router. When a trunk is close to the G-Cell boundary, the G-Cell has its capacity decremented. When the spine router uses more than 50% of the track resources, the capacity is updated. Finally, virtual routing takes place. This virtual routing merely is a fast implementation of both FastRoute, and RegularRoute (ZHANG; CHU, 2011) in sequence. The simplified implementation of FastRoute only uses Z- and L-shape patterns. The simplified implementation of RegularRoute only performs track assignment and has a reduced number of global segment choices. The intention is to very quickly produce an initial solution from which one can infer routablity statistics.

Phase two starts with a complete FastRoute execution utilizing the routability information from phase 1. Then, global segment extraction generates a global routing solution. A complete RegularRoute implementation uses this global routing solution, which produces a full routing solution. This solution is better than the previously provided by

Figure 4.2: GDRouter overall flow (Zhang; Chu, 2012).



Source: (Zhang; Chu, 2012)

just chaining FastRoute with RegularRoute because of the information generated in the first phase.

Phase three is called "iterative test routing." It consists of updating the usage estimations, incrementally using FastRoute to reduce the congestion by rerouting problematic nets, and then running a history-based RegularRoute. After an iteration, if the result has stopped improving or a certain number of iterations have already run, the algorithm finishes. Otherwise, this chain of an update, global routing, and detailed routing continues, improving the solution further.

5 PROPOSED INITIAL DETAILED ROUTING FLOW

5.1 Introduction

Figure 5.1 shows an overview of the proposed initial detailed routing flow. There are four main phases: Initialization Phase, Parallel Workload Creation Phase, Parallel Routing Phase, and Sequential Routing Phase. Section 5.2 goes into detail about each of the phases. The goal of the initial detailed routing flow differs slightly from the definition of the initial detailed routing. The reason is that we designed the flow to comply with the ISPD Initial Detailed Routing Contest suite (POSSER et al., 2018). There are a few specific details about the contest formulation that caused changes to the flow. The main difference is the greedy router process. Because any open in any net causes the entire solution to be invalid, this stage applies a greedy sequential shortest-path routing algorithm on any nets that the main flow could not route to prevent the solution's invalidation. Even though this is what the contest proposed, we believe that it is better to create an entire routing solution without resorting to algorithms that are mostly unaware of the bigger picture. While proposing such a flow is out of this work's scope, it is essential to remind ourselves that all stages of a flow share the goal and responsibility of achieving a final physical design.

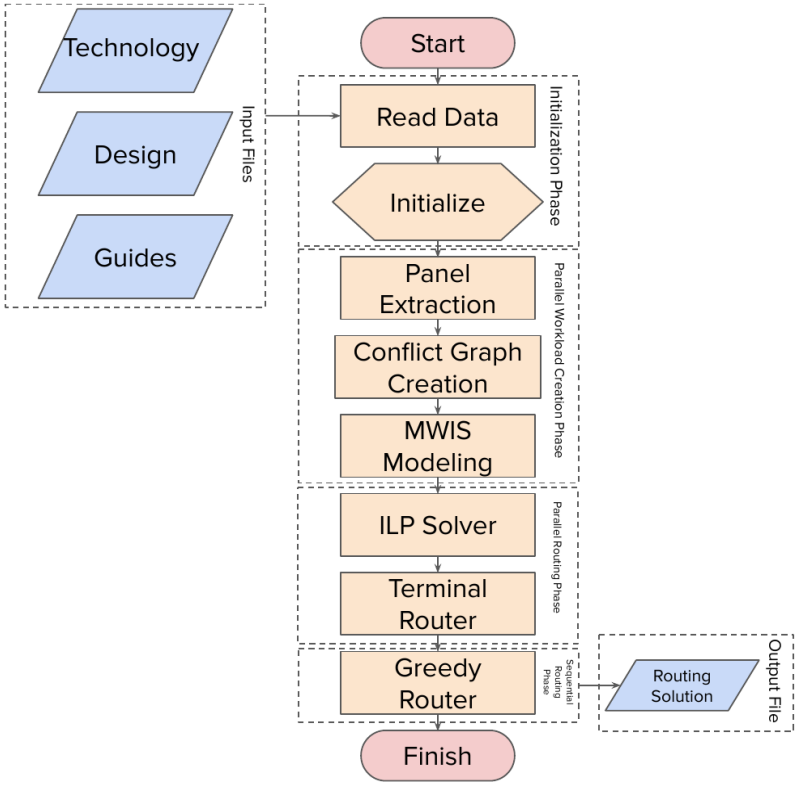
5.2 Initial Detailed Routing Flow Phases

The proposed initial detailed routing flow has four main phases. The following items contain a brief description of each phase.

- Initialization Phase: Reads the input data and initializes the routing problem instance.
- Parallel Workload Creation Phase: Aims to create independent workloads that can be addressed in parallel.
- Parallel Routing Phase: Attempts to solve the routing problem formulated by the previous phase.
- Sequential Routing Phase: Sequentially routes all nets left incomplete by the previous phase.

Each of the phases has algorithms that provide a solution to its problem. Note

Figure 5.1: Overview of the proposed initial detailed routing flow.



Source: From the author.

that this does not guarantee a valid, complete, or optimal solution to their problem. For example, a very congested routing problem could have an incomplete parallel routing solution.

The ideal initial detailed routing flow would be slightly different. Whenever the parallel routing phase failed to route large nets - large in terms of spanning through many GCells, and not in terms of the number of pins - or a large number of nets, it indicates a congestion problem that could have originated in global routing or even placement. A design with a large number of incomplete nets generally requires a sequential greedy routing of the incomplete nets. The routing of the incomplete nets would also have to navigate through a potentially large number of already routed nets in the vicinity, which, combined with special routing rules, increases routing difficulty even further.

The main routing phase, implemented in both the Parallel Workload Creation and Parallel Routing Phase, attempts to create a routing solution mostly correct by construction. When a solution of these phases is incomplete for a large number of nets, a cleverly implemented initial detailed routing flow could make use of the data provided by the incomplete routing to backtrack to global routing or even placement to fine-tune their solution considering the new, previously unavailable routing information. In this sense, the initial detailed routing could also act as a very accurate routability verification tool. A

complete physical design flow would very likely already contain a congestion estimation step - or perhaps even more than one. Remember, however, that what these congestion estimation algorithms essentially try to predict the result of the routing, and the information given by failure in a step of the initial detailed routing is the de facto information that the estimators merely attempt to predict.

5.2.1 Initialization Phase

5.2.1.1 Read Data

This step is included for completeness but depends entirely on the specific implementation of the files. Regarding the implementation of our proposed flow, chapter 6 contains information about the data formats used in this specific implementation.

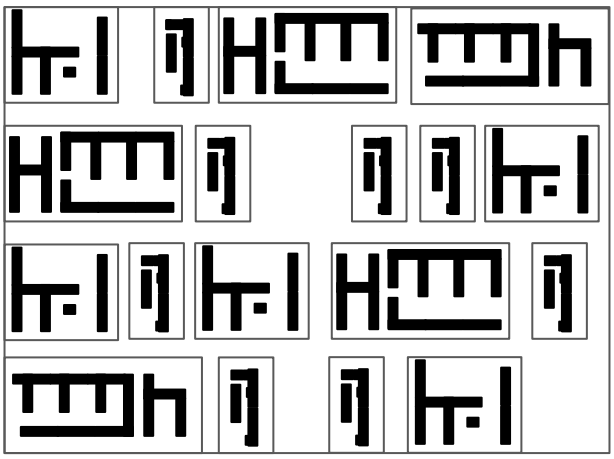
5.2.1.2 Initialize Data Structures

The second initialization step, labeled "Initialize" in the flow chart shown in figure 5.1, must implement some of the most important data structures in the entire flow: the routing grid and the geometry tree. The routing grid is a data structure representing the routing region in terms of a grid of a fixed granularity defined by the technology. The geometry tree is a data structure that stores the wires, vias, and blockages. In an exemplary implementation, both data structures would consume almost no space in memory while being instantaneous to access. However, the reality requires the trade-off between speed, memory footprint, utility, and complexity of implementation.

The geometry tree provides the flow with an interface to obtain information about structures in the vicinity of a given point. We propose adopting the R-Tree as the data structure for the geometry tree (GUTMAN, 1984). Using an open-source implementation of the tree data structure makes creating the geometry tree trivial.

In the routing grid, we propose a simple matrix for each layer. For each point, the grid can provide information such as if the node is occupied, if it contains a via to another layer, what is the net that the node belongs to (if any), and if the node is an access point to any pin of any cell. We recommend implementing each layer with its grid, mainly because it can have a different pitch and routing rules.

Figure 5.2: Example of a placed circuit.



Source: From the author.

5.2.2 Parallel Workload Creation Phase

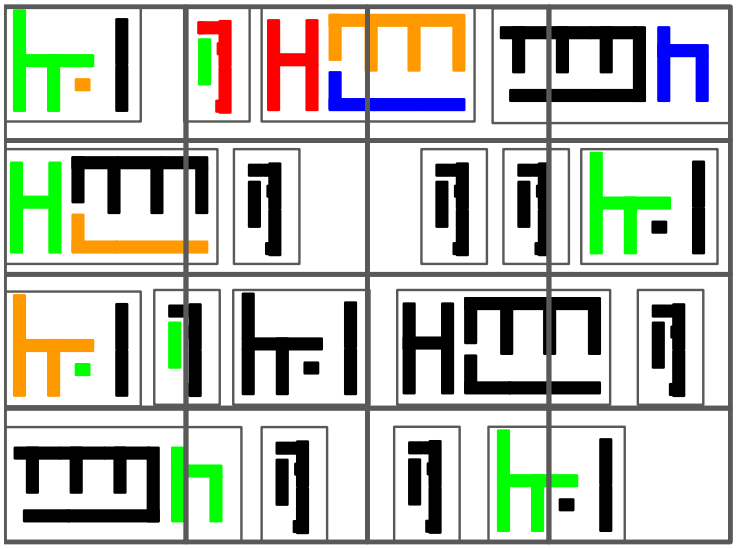
In the parallel workload creation phase, the algorithm employs techniques to divide the routing problem into independent smaller problems. We propose an approach similar to what was first proposed in (ZHANG; CHU, 2011), to divide the routing region into panels for each layer and work with panels as independent. The strategy consists of dividing the entire routing problem into panels. To better explain the creation, consider the example in figure 5.2. In this example, cells with their outlines are visible, and the pins are in full black. Please note that this example is not necessarily realistic or to scale.

The input files provide the information of which pins are logically connected, and the router should physically connect using wires and vias. Figure 5.3 contains the same problem, but some of the pins were color-coded to help visualize the logical connections. Furthermore, this version of the figure contains a GCell grid laid on top of the original problem.

According to the problem definition, the input should also specify routing guides for each of the nets. Figure 5.4 shows a hypothetical set of guides for each of the color-coded nets. Note that in 5.4e the figure contains the combination of the guides of all nets in the example, and where the guides overlap, the figure uses thinner rectangles to represent the guides - but they still contain the entire GCells and are unchanged in dimensions. This is to reduce visual pollution if all guides were to overlap.

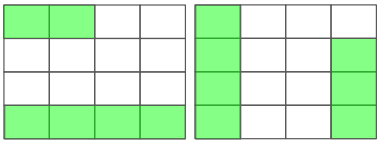
Finally, figure 5.5 shows the same configuration achieved in 5.4, but now the panels are separated clearly. Notice that most panels have trivial solutions, having either zero

Figure 5.3: Example placed circuit with an overlay GCell grid and some of the pins color coded.

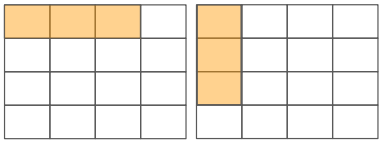


Source: From the author.

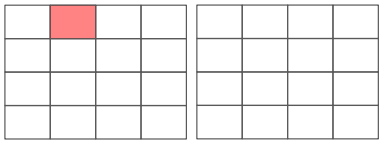
Figure 5.4: Guides of the nets previously color coded in figure 5.3



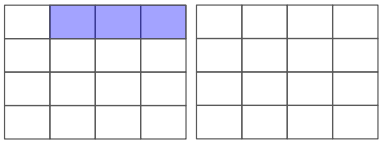
(a) Guides of the green coded net.



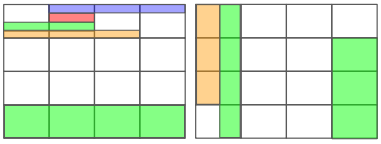
(b) Guides of the orange coded net.



(c) Guides of the red coded net.



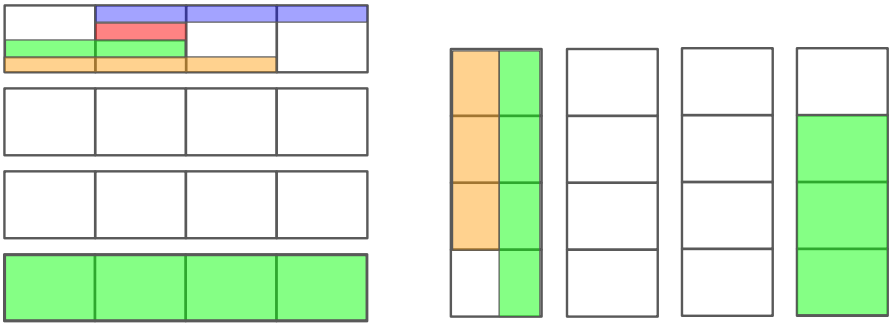
(d) Guides of the blue coded net.



(e) Guides of all nets combined.

Source: From the author.

Figure 5.5: Explicit panels obtained from the configuration in figure 5.4e.



(a) Horizontal panels. (b) Vertical panels.

Source: From the author.

or one guide spanning through it. The most interesting panel is the first horizontal panel, where all four exemplified nets overlap.

In this instance, the independent units of work are the panels, but not merely all panels in any order. Note that while the panels are mostly independent, adjacent panels share a boundary, where routing and spacing rules apply. Thus, this phase outputs four independent sets: odd-indexed vertical panels (OV), even-indexed vertical panels (EV), odd-indexed horizontal panels (OH), and even-indexed horizontal panels (EH).

Algorithm 1 shows a simplified initialization of the horizontal panels. Note that in the algorithm there is a function called *findGuidesInPosition()*. Considering that the guides were previously stored in an R-Tree, as discussed before, the average complexity of searching a guide given a position is $O(log(n))$ (GUTMAN, 1984). The creation of the vertical workload is analogous.

5.2.3 Parallel Routing Phase

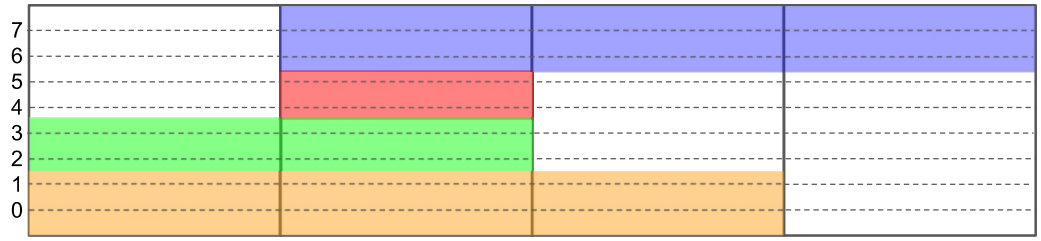
This phase's input includes the panels obtained previously. The objective is to assign one of the available tracks to each guide for every panel. Consider the configuration in figure 5.6, where the topmost horizontal panel of figure 5.4e is in detail. Once again, the guides that are part of the panel have a reduced height, but this is not the case in the real instance. The guides are shorter in the figure because otherwise, they would overlap, and the figure would be too polluted - they span the entire height of the panel.

The solution of this stage of the flow is to assign a track index to each guide in the panel. For example, the solution $S = \{\{Orange, 0\}, \{Green, 1\}, \{Red, 2\}, \{Blue, 3\}\}$ is a valid output, where the track assignment simply assigned the bottom-most free track

```
Data: Width and height of GCell Grid w and h respectively.
Result: Odd- and even- indexed horizontal panels OH and EH respectively.
EH ← {};
OH ← {};
for i ← 0 to w do
    panel ← {};
    for j ← 0 to h do
        guides = findGuidesInPosition(i, j);
        foreach guide g in guides do
            panel.insert(guide);
        end
    end
    if isOdd(i) then
        OH.insert(panel);
    else
        EH.insert(panel);
    end
end
return OH and EH
```

Algorithm 1: Initialization of horizontal panels.

Figure 5.6: Example of a horizontal panel with color coded guides, based on the example in figure 5.4e. The panel has eight tracks explicitly shown by the dashed lines and indexed on the left from zero to seven. Note that this illustration shows the guides with a height of only a quarter of the panel height, but they span the entire height of the panel.



Source: From the author.

to any of the guides. This trivial algorithm generates a solution that does not consider any information of the nets or any other objects in the routing problem and therefore is not ideal. What we propose is an approach that follows these guidelines:

- Model the panel as a conflict graph
- Prepare an ILP formulation that solves the maximum-weighted independent set in the conflict graph (MWIS)
- Solve the MWIS problem in the conflict graph
- Return the track assignment

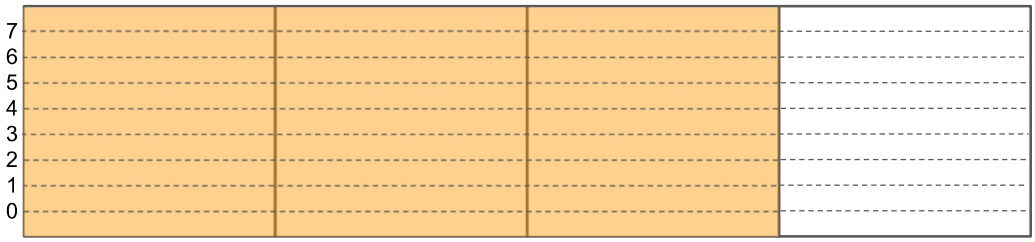
In this context, a conflict graph is a node-weighted in-directed graph where each node represents one possible track assignment for one guide, and edges between nodes represent a conflict between the connected nodes. Figure 5.7a shows the entire horizontal panel, but with only the orange guide, with its true height. Figure 5.7b shows a graph with eight nodes, each node is associated with a track of the panel, and the index of the track is indicated inside the node. Each node in the graph is called a choice because it represents one possible choice of track to assign to the guide. This completes the nodes' construction relevant to the orange guide, but there are no edges yet. This is where we use the concept of conflict by redundancy.

Note that when the algorithm assigns any of the eight tracks to a guide, there is no point in assigning another track because the first track already routes the guide as desired. Therefore, every node of the orange guide conflicts with every other node of the same guide by redundancy. In the graph, the conflict is represented by an edge between two nodes. Because in this case, the eight nodes conflict between each other, this eight-way conflict is represented by a clique, illustrated in figure 5.7c.

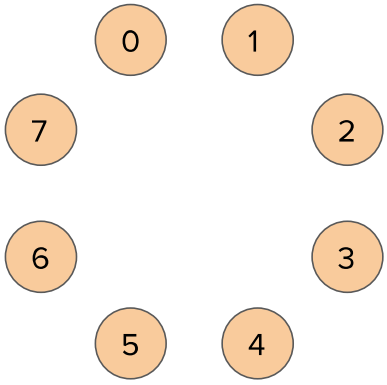
The same construction of a clique happens for every other guide in the panel. After creating the cliques, the algorithm achieves the configuration in figure 5.8.

The next step is to analyze the panel's guides to identify conflicts between guides, called conflict by overlap. This conflict happens when two guides overlap, and therefore the nodes of the graph corresponding to tracks of the same index conflict. Because the example with eight tracks would be too impractical to show in a figure, we resort to a simpler - but analogous - example with only three guides. Consider the simpler problem illustrated by figure 5.9. Figure 5.9a shows the relevant horizontal panel. Note that, unlike the previous example, this panel contains two blockages. When building the cliques representing each guide's track options, some tracks are blocked and unavailable to some of

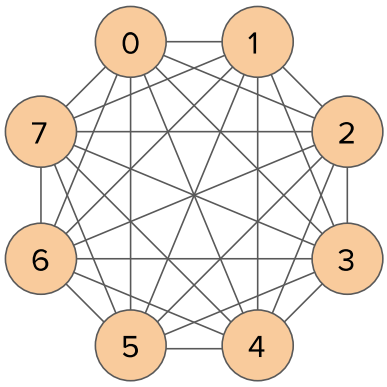
Figure 5.7: Creation of a part of the conflict graph that represents the orange nodes and the concept of conflict by redundancy.



(a) Example problem originally in figure 5.4e, but with only the orange guide and showing the real height of the orange guide.



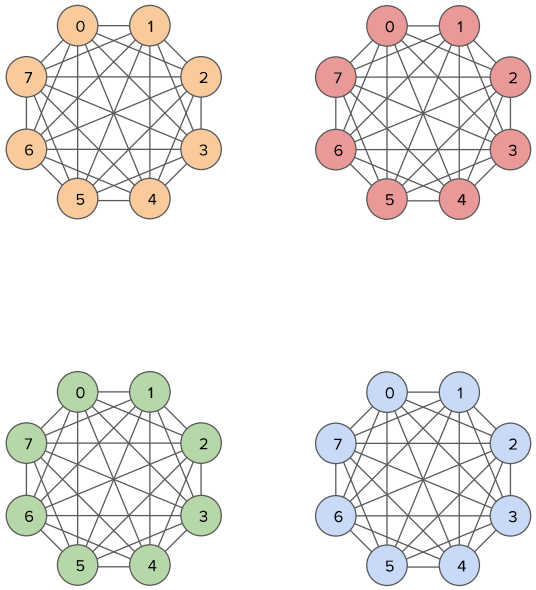
(b) Nodes in a graph representing the possible tracks for the orange guide. Note that the number inside the nodes are the track index, and not the weight of the nodes.



(c) A clique between all nodes of the graph of 5.7b, representing an eight-way conflict between all nodes.

Source: From the author.

Figure 5.8: Configuration obtained by constructing the four eight-way cliques, each representing the tracks for the guide of same color.



Source: From the author.

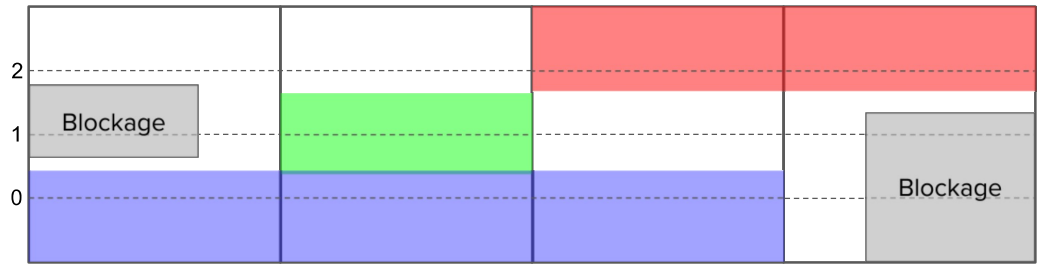
the guides. For example, note that the red guide cannot route through the tracks of index 0 or 1 because both are partially blocked in the rightmost G-Cell. Figure 5.9b shows the construction of the cliques representing each guide. The nodes are properly color-coded, and the number inside each node is the index of the track they represent. Note that the blue guide cannot use the track of index 1, and the red guide cannot use the tracks of index 0 or 1. The cliques do not contain the nodes associated with that choice. Finally, considering the guides' overlaps, figure 5.9c shows the complete conflict graph for this instance. The three nodes 2-indexed are conflicting by overlap, and the two 0-indexed nodes are why the bolder edges connect them.

With the conflict graph built for the panel, the next step is to model the maximum weighted independent set problem (MWIS). Before modeling, we first describe the MWIS problem and why solving it in the conflict graph produces a track assignment. The following definitions are formal descriptions of essential concepts building up to the MWIS.

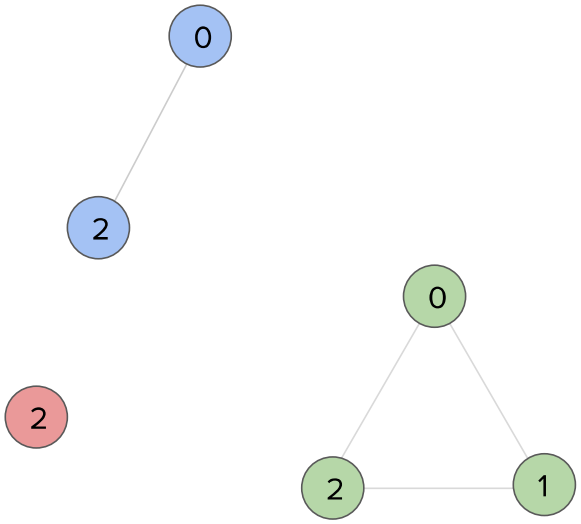
Definition 5.2.1 (Independent Set)**.** *"An independent set of a graph G is a subset S of nodes in G such that no two vertices of S represent an edge in G."*(PEMMARAJU; SKIENA, 2003)

Definition 5.2.2 (Maximum Independent Set)**.** The maximum independent set (MIS) of a graph G is the largest independent set S of G.(PEMMARAJU; SKIENA, 2003).

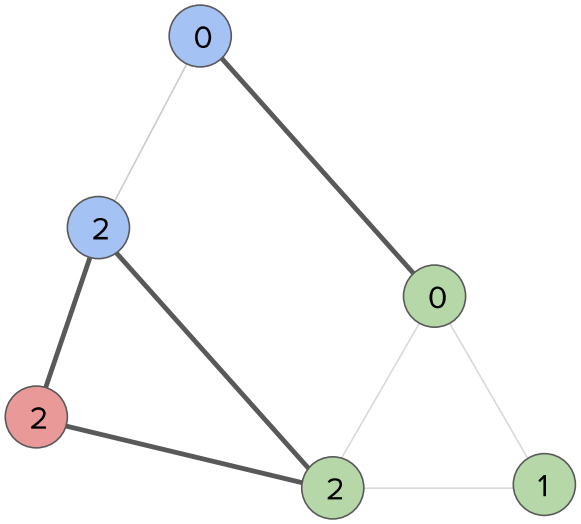
Figure 5.9: Step-by-step creation of a complete conflict graph for a given panel.



(a) Example horizontal panel with three horizontal tracks, three guides, and two blockages interfering with the tracks availability. The three horizontal tracks are indexed from 0 to 2.



(b) Graph containing the nodes generated from the panel in 5.9a and with the edges representing conflicts by redundancy. The index of the nodes correspond to the index of the track the choice represents - for example, the red and blue nodes indexed with 2 represent the choice that would route through track of index 2.



(c) Complete conflict graph representing the panel 5.9a. There is a connection between the green and blue nodes indexed 0 because both represent the intent of routing through track of index 0.

Source: From the author.

Definition 5.2.3 (Exact Weighted Independent Set)**.** The exact *weighted* independent set (EWIS) of a node-weighted graph G is an independent set S of G whose sum of component nodes weights is exactly a given value(MILANIC; MONNOT, 2007).

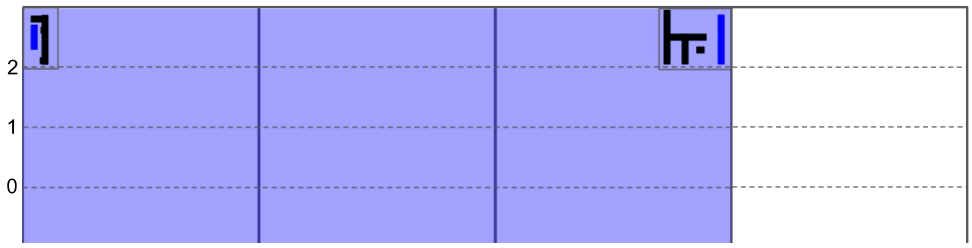
Definition 5.2.4 (Maximum Weighted Independent Set)**.** The maximum *weighted* independent set (MWIS) of a node-weighted graph G is the independent set S of G whose sum of component nodes weights is the largest possible.

It is instrumental in understanding why an independent set in the conflict graph generates a panel's track assignment. Consider the conflict graph in 5.9c. Now consider a set of nodes that is **not** independent, $S = \{Blue_2, Red_2, Green_1\}$. The track assignment represented by this solution would require a wire in track 1 for the green guide, a wire in track 2 for the blue wire, and a wire in track 2 for the red wire. This is impossible because track 2 cannot support two wires. This is why the nodes were connected in the first place because they *conflict by overlap*. Now consider an independent set $S = \{Blue_0, Red_2, Green_1\}$. The track assignment for this set is indeed possible because it requires a wire in track 0 for the blue guide, a wire in track 2 for the red guide, and a wire in track 1 for the green guide, which satisfies the physical requirement of one wire per track and no overlaps.

We are not, however, merely interested in a feasible track assignment. The quality of the wires generated is critical to prioritize specific parameters. To better model that, we propose a weight assignment to each node in the conflict graph. By assigning weights to each node and having the weight correlate to the metrics we wish to optimize, solving the MWIS problem in the conflict graph optimizes our metrics. Before discussing algorithms to solve the MWIS, we propose a strategy to calculate weights for the choices.

As described previously, each guide has a set of choices, one for each available track. In practice, some choices can be better than others during routing, even for the same guide. Consider the trivial case in figure 5.10. In this example, there is a straightforward two-pin net connecting the two blue pins. There are three available tracks to choose from for this guide, but it is evident that the track of index 2 is the closest to both pins. Choosing the tracks of index 1 or 0 would require adding longer terminal connections and, therefore, worse. To capture this phenomenon, we consider the estimated terminal connection cost and subtract it from the choice's weight. This way, the tracks with the smallest terminal connections are prioritized. Note that in the example provided, the terminals were the actual pins of the cell. When routing in higher layers, the terminals are the possible points where the bottom layer could contain a via.

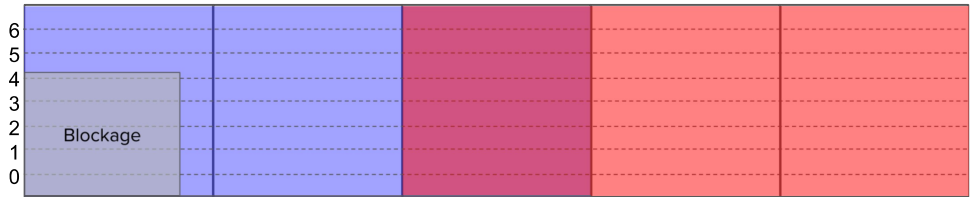
Figure 5.10: Example guide showing three possible choices to route a simple net.



Source: From the author.

Another important parameter to consider when calculating the weights of choices is the number of choices for each guide. Consider the example in figure 5.11. Note that there are two guides in this panel, one blue and one red. Both overlap in the middle G-Cell, and therefore their choices will conflict by overlap in the conflict graph. Now, notice that there is a blockage spanning the bottom five tracks in the leftmost G-Cell. This blockage causes the blue guide to only have two choices instead of seven for the red guide. This means the blue guide is *less flexible* when it comes to choices, and therefore harder to route. This incurs a flexibility weight to the blue greater than the red guide.

Figure 5.11: Example panel with two guides and a blockage.



Source: From the author.

Guides of hard to route nets should have routing priority. To model that, we consider three parameters:

- The number of pins of the net: This is trivial; nets with a higher number of pins are generally harder to route.
- The number of G-Cells the net spans through. Some global nets span through the entire routing area. Increasing the weights of their guides increase the priority of routing these large nets instead of smaller nets, some of which may be trivially routable.
- The panels' boundary density: In the same panel, some G-Cells have a higher number of guides passing through them. The number of guides passing through a G-Cell is called the boundary density. Choices that want to route through high boundary density regions are harder to route because they compete for the tracks with multiple other nets.

Table 5.1: Notation of the terms used for choice weight calculation.

Symbol	Term
W_C	Total weight of choice C
$\|N\|$	Number of pins of net N
F_G	Flexibility of guide G
D_{GC}	Boundary density of G-Cell GC
S_N	Number of G-Cells of net N
T_C	Estimated cost of terminal connections of choice C

Table 5.2: Notation used to model the MWIS into an ILP formulation (KAHNG; WANG; XU, 2018)

Notation	Rationale
$G(V, E)$	Conflict graph containing set of vertices V and set of edges E.
$v_{i,j}$	A vertex in the conflict graph representing the j^{th} choice of the i^{th} segment.
$e_{i,j}^{i',j'}$	An edge in the conflict graph indicating conflict between $v_{i,j}$ and $v_{i',j'}$.
$b_{i,j}$	Binary indicator of whether $v_{i,j}$ was used or not in the routing.
$w_{i,j}$	Weight of $v_{i,j}$.

Adopting the naming scheme in table 5.1, equation 5.1 shows the basic weight calculation equation. Note that all terms are multiplied by a constant. This allows for the normalization and weighting of the terms.

$$W_C = \alpha_1 * |N| + \alpha_2 * F_G + \alpha_3 * D_{GC} + \alpha_4 * S_N - \alpha_5 * T_C \tag{5.1}$$

The values for the alphas were found empirically. We have observed that our implementation of A* is particularly slow for nets spanning a large area. This is the main objective of the weights, increasing the chance that a net that A* would take long to route is instead routed fully by the track assignment flow.

With every node having its weight calculated, the weighted conflict graph is complete. The MWIS problem can be solved on the graph to produce the track assignment.

The proposed technique to solve the MWIS problem on the conflict graph is an ILP based algorithm. This technique involves modeling the problem as an ILP formulation, sending it to the ILP engine, and processing the result. The modeling is analogous to the one proposed by (KAHNG; WANG; XU, 2018), and we use the same notation, specified in table 5.2.

Also analogous to (KAHNG; WANG; XU, 2018), the ILP formulation is in equation 5.2. The formulation reads, "maximize the total weight of the selected vertices while

never selecting two adjacent vertices." This is the definition of the MWIS problem.

$$\begin{aligned} &\textbf{Maximize:} && \sum w_{i,j} * b_{i,j} \\ &\textbf{Subject to:} && b_{i,j} + bi', j' <= 1, \forall e_{i,j}^{i',j'} \in E \end{aligned} \tag{5.2}$$

The ILP solver's output is the track assignment that can be mapped back to the tracks for each panel. Of course, routing each guide through its given track is not sufficient to produce a complete routing. Subsection 5.2.4 will describe our proposed method to finish routing.

5.2.4 Sequential Routing Phase

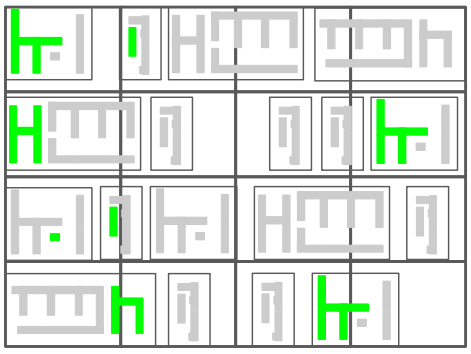
The input of this phase is the result of the track assignment. Therefore, most of the guides have a track assigned to them. First, we propose finishing routing of the nets that have a track assigned to every guide. Figure 5.12 shows the routing of a net given the track assignment solution. First, in figure 5.12b, the assignment is applied, and each guide is routed through its entire length. The obtained structure is a rough routing for the net, but the wires are not connected because they are in different layers. To connect the wires, the next stage inserts vias from the bottom layers to the top. Figure 5.12c shows the position of the vias connecting the pins (in metal 1) to the blue wires (in metal 2). Notice that the figure also has a red dashed box highlighting a via. In this track assignment, the pin is not directly under the wire, and a short wire is needed to connect the via to the track. This wire is called a terminal connection. In our proposed approach, the terminal connections are routed using A*. It is desirable to have as few of these as possible because they often (as in this case) span in the non-preferred direction and use more total wires than if not necessary. The vias connecting the metal 2 and metal 3 layers are added in 5.12d, and accomplish a fully routed structure, with all pins electrically connected. Notice, however, that some of the initially routed track assignment wires are longer than needed. The trimming of these wires produces the final result, shown in figure 5.12e.

Sequentially applying this to all nets that have complete track assignment produces a complete routing for many of the circuit's nets. However, some nets may not have a complete track assignment, where one or more of the net guides failed to secure itself a track. This can happen where there is considerable congestion of many nets competing for a portion of the track, and a less prioritized net may not have any track assigned. Sometimes, the tracks cannot be connected using vias, or even the pins cannot be accessed

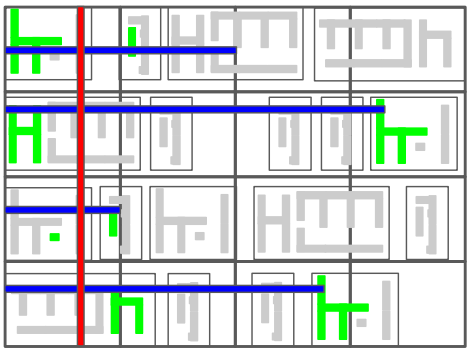
because of congestion. In all of these cases, our proposed approach discards any wire or via associated with these nets and attempt to route it using A* entirely.

We propose that the A* used to finish the routing have three different flavors. First, an A* confined to route within the routing guides, with increased cost for routing in non-preferred direction, and that has increased cost for using vias. Second, a similar implementation but with a high cost for routing outside the guides instead of impossibility, relaxing the routing overall. Finally, a third option that has a high cost for causing shorts, but not the impossibility. This allows for many nets to be routed entirely, albeit causing shorts. This third A* would not, in most cases, produce a valid routing, as having shorts invalidate the functionality of the circuit. However, producing the routing with the most possible routed nets can help identify congestions and possible improvements in the used algorithms. For example, regions of the circuit where nets were left unrouted or with shorts could have a cost to use when doing the global routing.

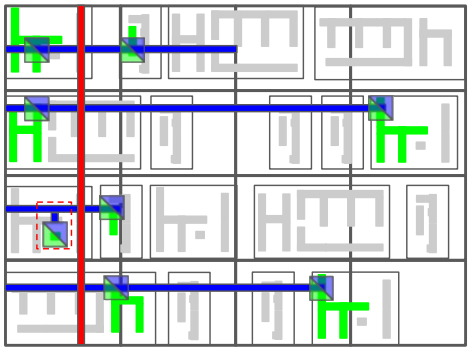
Figure 5.12: Implementation of the routing of a net given its track assignment.



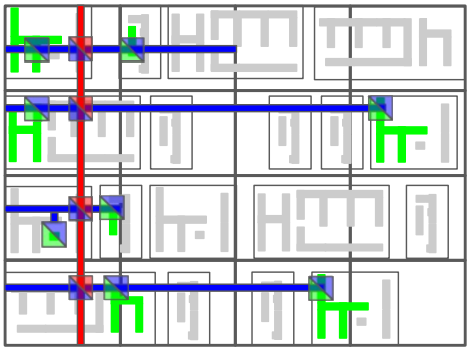
(a) Example with the green pins in detail.



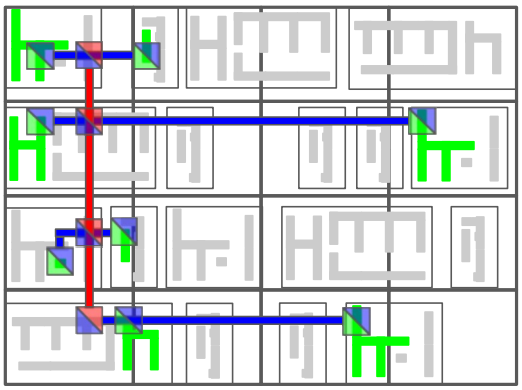
(b) Example track assignment for the green net.



(c) Vias between metal-1 and -2, with a terminal connection highlighted by the red dashed box.



(d) Vias between metal-2 and -3.



(e) Trimming process of the tracks.

Source: From the author.

6 IMPLEMENTATION OF THE PROPOSED ROUTING FLOW

This chapter presents an implementation of the flow proposed in Chapter 5. The implementation follows the specifications of the ISPD 2019 Contest (POSSER et al., 2018). Each test case consists of the following files:

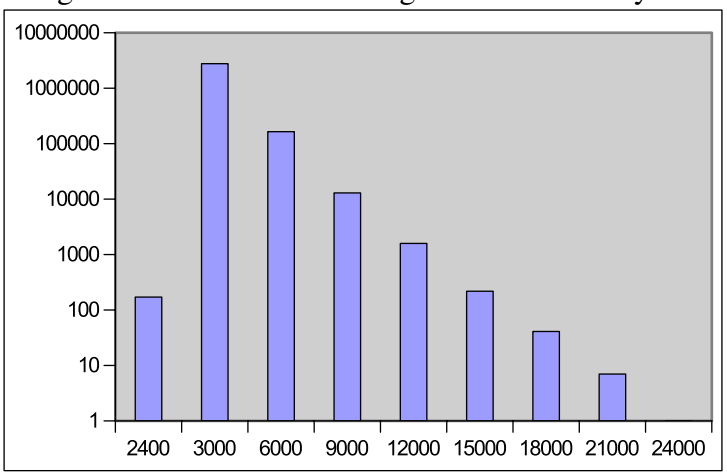
- LEF file: Formatted according to (INC., 2009). It contains *"the routing rules and the design information that need to be considered in the contest"* (POSSER et al., 2018). In practice, it contains essential information such as the grid width, size of the design, and exact values for the design rules in every layer. In the flow chart shown in 5.1, "Technology" represents the LEF file.
- DEF file: Also formatted according to (INC., 2009). It contains *"the placement information, pins, and blockages of the design"* (POSSER et al., 2018). Loading the DEF and LEF files is enough to generate the placement solution. In the flow chart shown in 5.1, "Design" represents the DEF file.
- Guides file: This file contains the global routing solution, with a layer assignment. Each net in the design must have a set of at least one rectangle in this file. Furthermore, the rectangles' coordinates must be aligned to the GCell grid, and each rectangle has a layer assigned to it. The format of the file is custom. In the flow chart shown in 5.1, "Guides" represents this file.

The guides file, containing the global routing solution, essentially contains a coarse routing solution for each net. As specified before, this file contains rectangles that should respect these rules:

- Alignment to the GCell grid: The coarseness of the global routing is the GCell grid. Therefore, the guides must be aligned to the G-Cell grid.
- Prefered-direction aware: Guides in vertical/horizontal layers must all span in the same direction and have the same width/height.
- The guide structure for a given net must not be disjoint. This means that a valid global routing solution does not generate two or more wholly disconnected groups of guides.

In the ISPD contest test suite, sometimes some of these rules are not respected, which increases the complexity of detailed routing. As an example, figure 6.1 shows the distribution of widths of the guides in test7 of the test suite. Notice that there are many different widths, indicating that the preferred direction is not respected. Perhaps more

Figure 6.1: Test7 width histogram of vertical layers.



Source: From the author.

importantly, the width is not always a multiple of the G-Cell width, in this case, 3000. There are some guides with widths of 2400, which violates alignment to the G-Cell grid.

The implementation of the proposed initial detailed routing flow was conducted based on Rsyn (FLACH et al., 2017), which provided the parsers for the files and the data structures for physical design, along with a framework on top of which the initial detailed routing flow was developed.

The implementation fails to route all nets in the benchmarks because of pin accessibility problems. To produce the results, a commercial tool was used to route the remaining unrouted nets. The worst case is test4, where 3% of the nets had pin access problems, while for most of the others it was close to 1%.

6.1 Results

This section analyses the results obtained by running the implemented proposed flow in the ISPD 2019 benchmarks. Table 6.1 shows the results in terms of score in the ISPD 2019 contest test suite, as defined in (POSSER et al., 2018). The table contains the results of the top four tools in the contest. Rsyn is a physical design framework developed in UFRGS, and it has been used for multiple different research projects in the last years.

Table 6.1: Results in terms of score of the implementation of the proposed flow in the ISPD 2019 contest test suite, alongside the four highest scoring tools in the contest.

	Dr. CU	NTUidRoute	TripleZ	Kim & Lee	SmartDR	Ours
test1	6	63	14	408	48	245
test2	219	1656	464	10264	796	6327
test3	10	150	31	815	61	414
test4	228	3043	585	11689	1182	7182
test5	33	506	-	-	-	838
test6	479	4244	873	28698	1604	13574
test7	992	8531	-	-	-	-
test8	1372	-	-	-	-	-
test9	2168	24415	-	-	-	-
test10	2163	23663	-	-	-	-

Table 6.2: Notation used in equation 6.1

Symbol	Meaning
V_i	The value reported by the evaluator for the parameter i
M_i	The multiplier used for metric i
W_i	The weight of metric i

6.1.1 Analysis of the Result

The score defined by the contest can be calculated as specified in equation 6.1, where the terms are as specified in table 6.2. The parameters are indexed by i in the notation. Table 6.3 list the parameters with the multiplier and weight utilized.

$$Score = \sum_{i=0}^{i=15} V_i * M_i * W_i \tag{6.1}$$

An interesting point of study of the proposed flow is its performance in the test suite when it comes to each parameter. Table 6.4 shows the scores obtained for the benchmarks test2, test4, and test6, and figure 6.3 graphs the impact of each parameter in the score obtained. The three graphs are very similar, with PRL, EOL, number of shorts, shorts area, and number of minimum area dominating the score. The number of cut spacing appear in both test2 and test6, but are much less common in test4, indicating that test4 has easier pin access and via insertion overall, perhaps as a result of the benchmark using 65nm technology - as opposed to 32nm for the other two - as well as a larger die size.

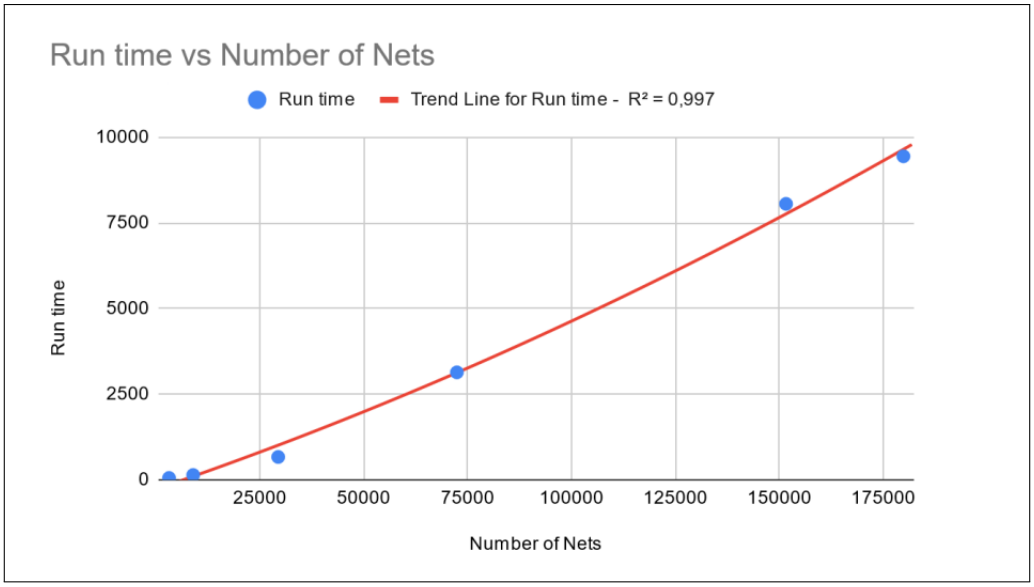
A regression analysis of the data obtained from the runtimes for test cases 1 to 6 show that the runtime in terms of number of nets is of a polynomial nature. Figure 6.2 shows that fitting a polynomial of second degree to the data achieves an R-squared of 0.997, indicating that the polynomial achieves a very close approximation. We believe

Table 6.3: Parameters used to calculate score, alongside their index, multiplier and weight.

Index	Parameter	Multiplier	Weight
0	Total wire length	0.005	0.5
1	Single-cut via count	1	4
2	Multi-cut via count	1	2
3	Out-of-guide wires	0.005	1
4	Out-of-guide vias	1	1
5	Off-track wires	0.005	0.5
6	Off-track vias	1	1
7	Wrong-way wires	0.005	1
8	Number of shorts	1	500
9	Area of shorts	0.000025	500
10	Number of minimum area violations	1	500
11	Number of parallel-run-length violations	1	500
12	Number of end-of-line violations	1	500
13	Number of cut spacing violations	1	500
14	Number of adjacent cut violations	1	500
15	Number of corner spacing violations	1	500

this is due to the fact that the larger test cases relied more on A* to route, which is slower than the track assignment flow.

Figure 6.2: Scatter plot of the runtime versus number of nets for test cases 1 through 6, with an approximation obtained by a second degree polynomial curve fitting achieving an R-squared of 0.997.

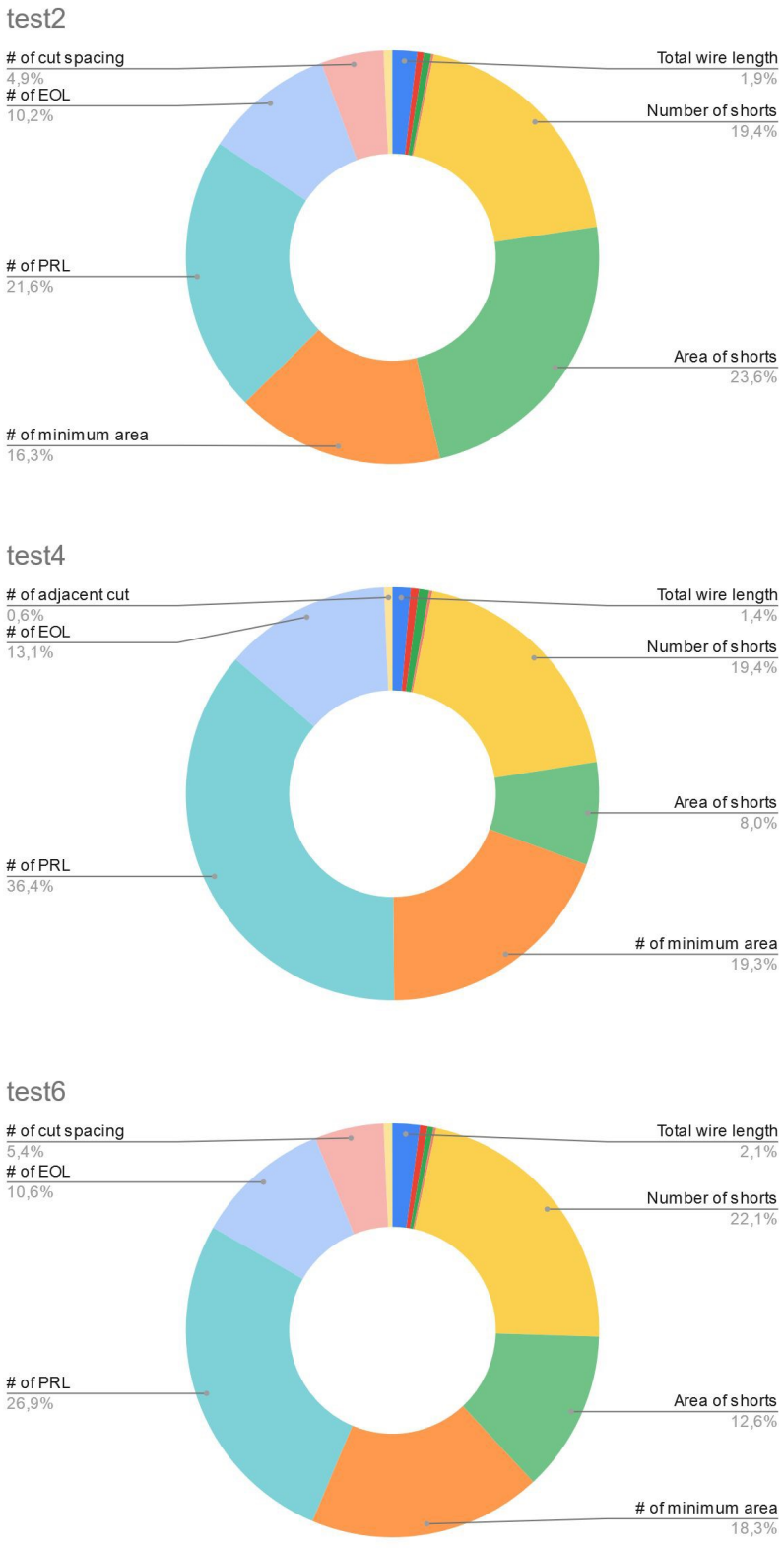


Source: From the author.

Table 6.4: Values obtained for each parameter by the implementation of the proposed flow for test2, test4 and test6.

Parameter	test2	test4	test6
Total wire length	$4,96 * 10^9$	$4,42 * 10^9$	$1,19 * 10^{10}$
Single-cut via count	852330	1264002	2077281
Multi-cut via count	0	0	0
Out-of-guide wire	$7,44 * 10^8$	$1,26 * 10^9$	$1,29 * 10^9$
Out-of-guide vias	118092	232925	229739
Off-track wires	$1,03 * 10^7$	$1,74 * 10^5$	$2,66 * 10^7$
Off-track vias	0	0	0
Wrong-way wires	$2,08 * 10^8$	$3,34 * 10^8$	$4,99 * 10^8$
Number of shorts	251936	304895	617545
Area of shorts	$1,23 * 10^{10}$	$5,05 * 10^9$	$1,41 * 10^{10}$
Number of minimum area violations	211657	302784	510180
Number of parallel-run-length violations	279920	571131	752478
Number of end-of-line violations	131654	204807	297227
Number of cut spacing violations	63589	330	151135
Number of adjacent cut violations	8064	10130	17897
Number of corner spacing violations	1020	0	1493

Figure 6.3: Distribution of score per parameter for tests 2, 4 and 6.



Source: From the author.

7 CONCLUSIONS AND FUTURE WORK

In this work, we have studied the initial detailed routing problem in the context of the VLSI physical design flow. Multiple fundamental algorithms for general routing were presented. These algorithms incorporate key concepts that are useful for routing in general and shape the underlying structures of modern frameworks. In special, A*, being a very efficient generic path-finding algorithm, has proven to be an essential building block of many state of the art frameworks. In some cases, the fundamental algorithms handle some constraints entirely. For example, we have observed that in (ZHANG; CHU, 2011) and (KAHNG; WANG; XU, 2018) shortest path search algorithms generate the terminal connections. Another frequent application of shortest path search is as a back-up for when the main algorithm fails to route some nets.

The study of the state of the art initial detailed routing frameworks reveals an important trend; the algorithms depend heavily on two concepts, conflict resolution and parallelization schemes. Modeling the conflicts as a conflict graph is the dominant technique, being employed by three of the state of the art frameworks with good results. The technique that proved the best for solving the conflicts is obtaining the maximum weighted independent set of the conflict graph. Also, the frameworks implicitly handle the net ordering problem by modeling the routing in such a way that allows simultaneous routing of multiple nets. For example, both (ZHANG; CHU, 2011) and (KAHNG; WANG; XU, 2018) achieve the simultaneous routing through the track assignment of all segments in a panel at the same time, which allows the algorithm to prioritize the more important segments.

7.1 Future Works

The implementation of the proposed initial detailed routing flow has shown to be a good base on which to develop more algorithms. The design rules such as PRL and, minimum area and EOL incur a huge penalty in the score, and a way to introduce algorithms that support them is a very promising subject. The inclusion of a pin access generation and a via selection algorithms in the complete flow is another topic of future work.

The main line of future work, however, is evaluating the viability of a track assignment step in detailed routing. The state-of-the-art shows that initial detailed routing flows that do not include a track assignment step are very competitive, sometimes even better than those with track assignment. We believe the track assignment step can accelerate the detailed routing if appropriately implemented into the flow. One of our proposition's mistakes is the attempt to maximize the track assignment success and focus on working with its result. Perhaps a fast, simplified, and lightweight track assignment step could be implemented as a way to accelerate the routing of the largest nets only.

REFERENCES

AKERS, S. B. A modification of lee's path connection algorithm. **IEEE Transactions on Electronic Computers**, IEEE, n. 1, p. 97–98, 1967.

ALPERT, C. J. et al. What makes a design difficult to route. In: ACM. **Proceedings of the 19th international symposium on Physical design**. [S.l.], 2010. p. 7–12.

ASANO, T. Parametric pattern router. In: IEEE. **Design Automation, 1982. 19th Conference on**. [S.l.], 1982. p. 411–417.

CHAN, W.-T. J. et al. Routability optimization for industrial designs at sub-14nm process nodes using machine learning. In: **Proceedings of the 2017 ACM on International Symposium on Physical Design**. New York, NY, USA: ACM, 2017. (ISPD '17), p. 15–21. ISBN 978-1-4503-4696-2. Available from Internet: <http://doi.acm.org/10.1145/3036669.3036681>.

Chang, Y.; Lee, Y.; Wang, T. Nthu-route 2.0: A fast and stable global router. In: **2008 IEEE/ACM International Conference on Computer-Aided Design**. [S.l.: s.n.], 2008. p. 338–343. ISSN 1092-3152.

CHANG, Y.-T. et al. Obstacle-avoiding rectilinear steiner minimal tree construction. In: IEEE. **VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on**. [S.l.], 2008. p. 35–38.

CHO, M. et al. Boxrouter 2.0: Architecture and implementation of a hybrid and robust global router. In: IEEE. **2007 IEEE/ACM International Conference on Computer-Aided Design**. [S.l.], 2007. p. 503–508.

CHU, C.; WONG, Y.-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**, IEEE, v. 27, n. 1, p. 70–83, 2008.

CLOW, G. W. A global routing algorithm for general cells. In: IEEE. **Design Automation, 1984. 21st Conference on**. [S.l.], 1984. p. 45–51.

DANIGNO, M. et al. Proposal and evaluation of pin access algorithms for detailed routing. In: IEEE. **2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. [S.l.], 2019. p. 602–605.

DAS, S.; KHATRI, S. P. A regularity-driven fast gridless detailed router for high frequency datapath designs. In: ACM. **Proceedings of the 2001 international symposium on Physical design**. [S.l.], 2001. p. 130–135.

FLACH, G. et al. Rsyn: An extensible physical synthesis framework. In: **Proceedings of the 2017 ACM on International Symposium on Physical Design**. New York, NY, USA: Association for Computing Machinery, 2017. (ISPD '17), p. 33–40. ISBN 9781450346962. Available from Internet: <https://doi.org/10.1145/3036669.3038249>.

FOGAÇA, M. et al. Contributions to openroad from abroad: experiences and learnings. In: IEEE. **2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)**. [S.l.], 2020. p. 1–8.

GANLEY, J. L.; COHOON, J. P. Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles. In: IEEE. **Circuits and Systems, 1994. ISCAS'94., 1994 IEEE International Symposium on**. [S.l.], 1994. v. 1, p. 113–116.

GESTER, M. et al. Bonnroute: Algorithms and data structures for fast and good vlsi routing. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, ACM, v. 18, n. 2, p. 32, 2013.

GILBERT, E. N.; POLLAK, H. O. Steiner minimal trees. **SIAM Journal on Applied Mathematics**, SIAM, v. 16, n. 1, p. 1–29, 1968.

GONÇALVES, S. M.; JR, L. S. d. R.; MARQUES, F. S. Smartdr: algorithms and techniques for fast detailed routing with good design rule handling. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, ACM New York, NY, USA, v. 26, n. 2, p. 1–38, 2020.

GUTMAN, A. R-trees: A dynamic index structure for spatial searching in proc. In: **ACM SIGMOD**. [S.l.: s.n.], 1984. p. 45–57.

HANAN, M. On steiner's problem with rectilinear distance. **SIAM Journal on Applied Mathematics**, SIAM, v. 14, n. 2, p. 255–265, 1966.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE transactions on Systems Science and Cybernetics**, IEEE, v. 4, n. 2, p. 100–107, 1968.

HENTSCHKE, R. et al. Maze routing steiner trees with delay versus wire length tradeoff. **IEEE transactions on very large scale integration (VLSI) systems**, IEEE, v. 17, n. 8, p. 1073–1086, 2009.

HENTSCHKE, R. F. et al. Maze routing steiner trees with effective critical sink optimization. In: **Proceedings of the 2007 international symposium on Physical design**. [S.l.: s.n.], 2007. p. 135–142.

HSIAO, J. Y.; TANG, C. Y.; CHANG, R. S. An efficient algorithm for finding a maximum weight 2-independent set on interval graphs. **Inf. Process. Lett.**, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 43, n. 5, p. 229–235, oct. 1992. ISSN 0020-0190. Available from Internet: <http://dx.doi.org/10.1016/0020-0190(92)90216-I>.

HU, Y. et al. An-oarsman: Obstacle-avoiding routing tree construction with good length performance. In: ACM. **Proceedings of the 2005 Asia and South Pacific Design Automation Conference**. [S.l.], 2005. p. 7–12.

INC., C. D. S. **LEF/DEF Language Reference**. 2009. <https://web.archive.org/web/20181031162337/http://www.ispd.cc/contests/18/lefdefref.pdf>. [Online; accessed 26-July-2020; Archived from the original on 31-October-2018].

JIA, X. et al. Mcfroute: a detailed router based on multi-commodity flow method. In: IEEE. **Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on**. [S.l.], 2014. p. 397–404.

Jia, X. et al. A multicommodity flow-based detailed router with efficient acceleration techniques. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 37, n. 1, p. 217–230, Jan 2018. ISSN 0278-0070.

JOHANN, M.; REIS, R. Techniques and results of the marte routing system. In: **Proceedings of 8th SBCCI**. [S.l.: s.n.], 1994. v. 29.

JOHANN, M. d.; KINDEL, M.; REIS, R. daL. Layout synthesis using transparent cells and fotc routing. In: IEEE. **38th Midwest Symposium on Circuits and Systems. Proceedings**. [S.l.], 1995. v. 2, p. 787–790.

JOHANN, M. d. O. Novos algoritmos para roteamento de circuitos vlsi. 2001.

JOHANN, M. D. O.; REIS, R. D. L. Legal: an algorithm for simultaneous net routing. In: IEEE. **Symposium on Integrated Circuits and Systems Design**. [S.l.], 2001. p. 180–185.

JOHANN, M. d. O.; REIS, R. da L. A full over-the-cell routing model. In: IEEE. **Proceedings of ASP-DAC'95/CHDL'95/VLSI'95 with EDA Technofair**. [S.l.], 1995. p. 845–850.

JOHANN, M. de O.; SANTOS, G. B. V. dos; REIS, R. A. da L. A legal algorithm following global routing. In: IEEE. **Proceedings. 15th Symposium on Integrated Circuits and Systems Design**. [S.l.], 2002. p. 271–276.

KAHNG, A. B. et al. **VLSI physical design: from graph partitioning to timing closure**. [S.l.]: Springer Science & Business Media, 2011.

KAHNG, A. B.; WANG, L.; XU, B. Tritonroute: an initial detailed router for advanced vlsi technologies. In: ACM. **Proceedings of the International Conference on Computer-Aided Design**. [S.l.], 2018. p. 81.

KWA, J. B. Bs*: An admissible bidirectional staged heuristic search algorithm. **Artificial Intelligence**, Elsevier, v. 38, n. 1, p. 95–109, 1989.

LEE, C. Y. An algorithm for path connections and its applications. **IRE transactions on electronic computers**, IEEE, n. 3, p. 346–365, 1961.

LEUNG, H. K.-S. Advanced routing in changing technology landscape. In: ACM. **Proceedings of the 2003 international symposium on Physical design**. [S.l.], 2003. p. 118–121.

LI, H. et al. Dr. cu 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction. In: IEEE. **2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.], 2019. p. 1–7.

LIN, C.-W. et al. Obstacle-avoiding rectilinear steiner tree construction based on spanning graphs. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 27, n. 4, p. 643–653, 2008.

MILANIC, M.; MONNOT, J. On the complexity of the exact weighted independent set problem. 2007.

MOORE, E. F. The shortest path through a maze. In: **Proc. Int. Symp. Switching Theory, 1959**. [S.l.: s.n.], 1959. p. 285–292.

NAM, G.-J. et al. The ispd2005 placement contest and benchmark suite. In: ACM. **Proceedings of the 2005 international symposium on Physical design**. [S.l.], 2005. p. 216–220.

NAM, G.-J.; SZE, C.; YILDIZ, M. The ispd global routing benchmark suite. In: ACM. **Proceedings of the 2008 international symposium on Physical design**. [S.l.], 2008. p. 156–159.

NAM, G.-J. et al. Ispd placement contest updates and ispd 2007 global routing contest. In: ACM. **Proceedings of the 2007 international symposium on Physical design**. [S.l.], 2007. p. 167–167.

NUNES, L.; REIMANN, T.; REIS, R. Gr-pa: A cost pre-allocation model for global routing. In: IEEE. **2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)**. [S.l.], 2013. p. 134–137.

NUNES, L.; REIS, R. Global routing congestion reduction with cost allocation look-ahead. In: IEEE. **2013 26th Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.], 2013. p. 1–5.

PATEL, A. **Amit's A* Pages**. 2018. <https://theory.stanford.edu/~amitp/GameProgramming/index.html>. Accessed: 2018-12-20.

PEARL, J. **Heuristics: Intelligent Search Strategies for Computer Problem Solving**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984. ISBN 0-201-05594-5.

PEMMARAJU, S.; SKIENA, S. **Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica®**. [S.l.]: Cambridge university press, 2003. 318 p.

POHL, I. Bi-directional search. **Machine intelligence**, v. 6, p. 127–140, 1971.

POSSER, G. et al. **Introduction of ISPD19 Contest Problem**. 2018. <http://www.ispd.cc/contests/19/Introduction_of_ISPD19_Contest_Problem.pdf>. Accessed: 2019-01-03.

Qi, Z.; Cai, Y.; Zhou, Q. Accurate prediction of detailed routing congestion using supervised data learning. In: **2014 IEEE 32nd International Conference on Computer Design (ICCD)**. [S.l.: s.n.], 2014. p. 97–103. ISSN 1063-6404.

QROUTER. 2017. Available from Internet: <http://opencircuitdesign.com/qrouter/>.

REIMANN, T. J.; SANTOS, G. B.; REIS, R. A. Routing algorithms performance in different routing scopes. In: IEEE. **2010 17th IEEE International Conference on Electronics, Circuits and Systems**. [S.l.], 2010. p. 643–646.

REIS, R.; GOMES, R.; LUBASZEWSKI, M. An efficient design methodology for standard cell circuits. In: IEEE. **1988., IEEE International Symposium on Circuits and Systems**. [S.l.], 1988. p. 1213–1216.

SANTOS, G. et al. The fidelity property of the elmore delay model in actual comparison of routing algorithms. In: IEEE. **2010 IEEE International Conference on Computer Design**. [S.l.], 2010. p. 195–202.

SANTOS, G. B. V. dos; JOHANN, M. de O.; REIS, R. A. da L. Channel based routing in channel-less circuits. In: IEEE. **2006 IEEE International Symposium on Circuits and Systems**. [S.l.], 2006. p. 4–pp.

SHIRAKAWA, I.; FUTAGAMI, S. A rerouting scheme for single-layer printed wiring boards. **IEEE transactions on computer-aided design of integrated circuits and systems**, IEEE, v. 2, n. 4, p. 267–271, 1983.

SHOJAEI, H.; DAVOODI, A.; LINDEROTH, J. T. Congestion analysis for global routing via integer programming. In: **Proceedings of the International Conference on Computer-Aided Design**. Piscataway, NJ, USA: IEEE Press, 2011. (ICCAD '11), p. 256–262. ISBN 978-1-4577-1398-9. Available from Internet: <http://dl.acm.org/citation.cfm?id=2132325.2132386>.

SOUKUP, J. Fast maze router. In: IEEE PRESS. **Proceedings of the 15th Design Automation Conference**. [S.l.], 1978. p. 100–102.

SUN, F.-K. et al. A multithreaded initial detailed routing algorithm considering global routing guides. In: ACM. **Proceedings of the International Conference on Computer-Aided Design**. [S.l.], 2018. p. 82.

Tabrizi, A. F. et al. Detailed routing violation prediction during placement using machine learning. In: **2017 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)**. [S.l.: s.n.], 2017. p. 1–4. ISSN 2472-9124.

Tabrizi, A. F. et al. A machine learning framework to identify detailed routing short violations from a placed netlist. In: **2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2018. p. 1–6.

TUMELERO, D.; BONTORIN, G.; REIS, R. Overhead for independent net approach for global routing. In: IEEE. **2015 IEEE 6th Latin American Symposium on Circuits & Systems (LASCAS)**. [S.l.], 2015. p. 1–4.

WESTRA, J.; GROENEVELD, P. Is probabilistic congestion estimation worthwhile? In: ACM. **Proceedings of the 2005 international workshop on System level interconnect prediction**. [S.l.], 2005. p. 99–106.

WU, Y.-F. et al. Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles. **IEEE Transactions on Computers**, IEEE, v. 100, n. 3, p. 321–331, 1987.

XU, X. et al. Self-aligned double patterning aware pin access and standard cell layout co-optimization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 34, n. 5, p. 699–712, 2015.

XU, Y.; ZHANG, Y.; CHU, C. Fastroute 4.0: Global router with efficient via minimization. In: **Proceedings of the 2009 Asia and South Pacific Design**

Automation Conference. Piscataway, NJ, USA: IEEE Press, 2009. (ASP-DAC '09), p. 576–581. ISBN 978-1-4244-2748-2. Available from Internet: <http://dl.acm.org/citation.cfm?id=1509633.1509768>.

ZHANG, Y.; CHU, C. Regularroute: An efficient detailed router with regular routing patterns. In: ACM. **Proceedings of the 2011 international symposium on Physical design**. [S.l.], 2011. p. 45–52.

Zhang, Y.; Chu, C. Gdrouter: Interleaved global routing and detailed routing for ultimate routability. In: **DAC Design Automation Conference 2012**. [S.l.: s.n.], 2012. p. 597–602. ISSN 0738-100X.

APPENDIX — APPENDIX A

List of published research:

Title: Clip Clustering for Early Litographic Hotspot Classification
Authors: Oliveira, Andre; Puget, Julia; Metzler, Carolina; Reis, Ricardo.
Event: 2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS)
DOI: 10.1109/LASCAS.2019.8667548

Title: Algorithms for Access Point Selection at Pre-Routing Stage
Authors: Danigno, Marcelo; Fogaca, Mateus; Monteiro, Eder; Ferreira, Jorge; Oliveira, Andre; Reis, Ricardo; Butzen, Paulo.
Journal: Journal of Integrated Circuits and Systems (JICS)
Accepted for publication

Title: Proposal and Evaluation of Pin Access Algorithms for Detailed Routing
Authors: Danigno, Marcelo; Butzen, Paulo; Ferreira, Jorge; Oliveira, Andre; Monteiro, Eder; Fogaca, Mateus; Reis, Ricardo.
Event: 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)
DOI: 10.1109/ICECS46596.2019.8965194

Title: UFRGSPlace: Routability Driven FPGA Placement Algorithm for Heterogeneous FPGAs
Authors: Puget, Julia; Oliveira, Andre; Seclen, Jorge; Reis, Ricardo.
Event: 2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)
DOI: 10.1109/ICECS.2017.8292069