Reducing the Mismatch Between Global and Detailed Routing of Integrated Circuits

by

Daohang Shi

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2016

Date of final oral examination: 12/06/2016

The dissertation is approved by the following members of the Final Oral Committee:

Azadeh Davoodi, Associate Professor, Electrical and Computer Engineering, Chair
Yu-Hen Hu, Professor, Electrical and Computer Engineering
Younghyun Kim, Assistant Professor, Electrical and Computer Engineering
Dan Negrut, Associate Professor, Mechanical Engineering
Parameswaran Ramanathan, Professor, Electrical and Computer Engineering

ACKNOWLEDGMENTS

I would like to give my thanks to my advisor, Professor Davoodi, for her guidance and support during my years in Madison. I would also like to thank the committee members for their time and suggestions.
Finally, to my wife Ying and to my daughter Maddie.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Electronic Design Automation (EDA) of Integrated Circuits (ICs) is facing many challenges in the dusk of Moore's law. Among various EDA tools, routing is a key contributor to the overall quality of an IC and has become significantly complicated by technology scaling due to the increasing number of design rules and fabrication constraints. Among different steps of routing, the 'detailed-routing' step is particularly time-consuming because at this step every design rule needs to be satisfied for every routed net. To reduce the runtime spent on detailed routing, the preceding 'global routing' step uses approximations to quickly generate a routing solution in order to provide a better starting point for detailed routing. However the utility of global routing has been continuously diminishing over the past years because existing global routing procedures are inherently ignorant of design rules and fabrication constraints. As a result, the gap between global and detailed routing has been growing.

This dissertation aims to bridge the gap between global and detailed routing by making global routing aware of relevant factors which used to only be considered at the detailed routing stage. We build models to capture these factors by identifying the most significant sources of mismatch between the global and detailed routing stages. These include models to capture variation in the size of inter-layer vias caused by metal layer heterogeneity in modern fabrication technologies, as well as models to estimate the locations of these vias to better estimate routing resource usage at the global routing stage. We also utilize an emerging integration technology for global routing to address the pin accessibility challenge that is seen at the detailed routing stage. Finally we introduce a new metric and optimization strategy for more effective evaluation of 'routability' at the global routing stage. To validate our

models and techniques, we have built a high-end validation framework featuring 45nm and 22nm designs released by industry, and the Olympus SoC tool-set by Mentor Graphics.

1.1 Motivation and Preliminaries

Physical design is a step after circuit design and logic synthesis in the VLSI design cycle. During this step physical information of the circuit including component placement and interconnect routing are determined. A complete physical design flow usually contains several stages, which are circuit partitioning, floor-planning, placement, clock network synthesis, routing, and timing closure.

The routing stage connects all the pins for each net and is usually split into global routing (GR) and detailed routing (DR). At the global routing stage, the entire layout region is first divided into tiles (commonly known as *global-cells* or *g-cells*). An example is illustrated in Figure 1.1. The boundaries between adjacent g-cells are called *edges*. Instead of finding routes to connect the pins, global router finds routes to connect those g-cells in which the pins are located. The goal is to connect the pins in order to prevent wiring demands from exceeding edge capacities.

At the detailed routing stage, the exact physical information about detailed wirings is determined using the global routing results as a starting point. This means that every wire from the global routing stage will be assigned to a routing track. For example, in Figure 1.1, the global routing solution of `net1` is the L-shape routing which consists of three wires. The wire in the bottom will be assigned to one routing track between the two highlighted g-cells (in light green). In addition, the detailed location of the vias, which are used to connect wires on adjacent metal layers, will also be determined inside each g-cell after every routing segment is assigned to a routing track.

The gateway to transition from a global routing solution to a detailed

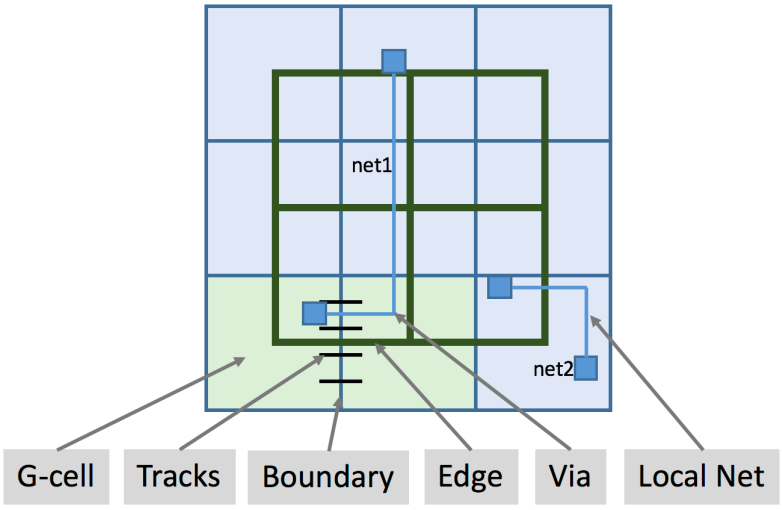


Figure 1.1: Example of global cells (g-cells). The entire region is partitioned into 3 $\times$ 3 g-cells. There are 4 routing tracks on the boundary between the two highlighted g-cells.

one is through a step called 'track assignment'. During this step, the flat segments of each net routed during the GR stage are assigned to individual routing tracks. Specifically, this assignment needs to follow the global routing solution, i.e., it needs to ensure that the assigned track to each flat segment is chosen from those g-cells in which the net was routed at the GR stage. Once the track for each flat segment is determined, the DR stage initiates which may then change many of the track assignments in order to satisfy various design rules and performance requirements. This actual DR step is significantly more time-consuming than the initial track assignment.

There are many design rules at the DR stage which need to be satisfied. This makes detailed routing extremely difficult and time consuming. The increasing complexity of design rules in advanced technology nodes also results in mismatch between global and detailed routing. The mismatch is mainly resulted from the fact that many of these de-

sign rules are ignored during global routing. Meanwhile, detailed routing utilizes the routing solution generated at the global routing stage as its starting point. Therefore, detailed routing can converge faster to a feasible routing solution that is free of design-rule violations if its mismatch with the global routing is reduced. Modern VLSI design flow typically requires multiple rounds of global and detailed routing so reducing this mismatch can significantly accelerate the design cycle and reduce time-to-market of the chip Graphics; Synopsys; Cadence.

1.2 Sources of Mismatch between Global and Detailed Routing

In this section we categorize and discuss key sources of mismatch between global and detailed routing.

- **Local Nets**:

  Local nets are those nets whose pins are located in the same g-cell. For example, in Figure 1.1, `net2` is a local net. The wirings of a local net are usually within the same g-cell without traversing any of the boundaries of the g-cell, so local nets are commonly ignored at the global routing stage. However many nets may be local for example, 31.20% of the nets in the ISPD'11 Viswanathan et al. (2011) benchmarks are local nets Shojaei et al. (2013). Ignoring the routing resources consumed by local net routing during global routing results in the mismatch between GR and DR.

  A few prior works studied the local net routing during global routing Wei et al. (2012); Shojaei et al. (2013, 2011). In LCGRIP Shojaei et al. (2013, 2011), the area required to route a local net during global routing is approximated by multiplying the wire

width by the half-perimeter of the net's bounding box. Furthermore it introduces a vertex capacity for each g-cell to be used during global routing. If a g-cell is in the lowest two routing layers, its vertex capacity will be deducted by the required area to route the local nets inside the g-cell. In another recent work called GLARE Wei et al. (2012), the evaluation of local net routing is based on the pin density. GLARE blocks $k \times n$ routing tracks which it assumes are taken by the local nets in the lowest two routing metal layers, where k is a tuned parameter and n refers to the number of pins inside the g-cell. It reports improved results using this estimation technique. the estimation of local net modeling has not been verified by a commercial detailed router in any previous work.

- **Inter-layer Vias:**

  At the global routing stage, vias are traditionally modeled as connections between adjacent layers. A via model includes an estimate of its routing resource usage of a g-cell Hsu et al. (2008). However this model ignores many factors seen in modern technology nodes and is no longer effective during global routing.

  First, the traditional via model does not capture the significant change in the metal widths of adjacent routing layers. For example, at the 32nm technology node, there are up to 6 different metal widths ranging from 1x to 20x across 12 metal layers Alpert et al. (2010). Vias between two adjacent layers with different metal widths will consume additional routing resource on the lower one. This issue was never considered in any previous work.

  Second, to more accurately estimate the routing resource usage by vias, it is desirable to introduce 'via capacity', similar to edge capacity, at the global routing stage. In Hsu et al. (2008), a *via capacity model* is proposed to estimate the routing resource usage

within a g-cell. In this model, the via capacity of a g-cell is determined by computing the available area inside a g-cell, divided by the area of one via. This model captures the maximum number of vias that can be included in a g-cell. However it can be too optimistic in practice because the routing resources inside a g-cell are mostly consumed by global routes and not vias. Furthermore, this model does not take into account significant change in the widths of wires in adjacent metal layers.

Third, various types of vias with squares or rectangular shapes may be instantiated at the DR stage to trade off between manufacturability and routability. Typically square vias are preferred to achieve better manufacturability. However square vias cause more blockage of routing tracks because they need to be larger than rectangular-shaped ones; indeed for better routability, square vias can be replaced by smaller rectangular vias in certain cases Han et al. (2015). This determination of the via shape further complicates the detailed routing stage and increases the mismatch between GR and DR.

Fourth, the location of vias may have a big impact on local congestions inside the g-cells. Specifically, during global routing, the center of the g-cells are assumed to be the end-points of each flat segment of a global net. Global router has no awareness of where the vias are located in the g-cells. However the locations of the vias determine how far each flat wire segment enters the g-cells. Thus the locations of vias in the g-cells can help to more accurately estimate the routing resource usage inside the g-cells. This lack of knowledge about the via locations is another main source of mismatch between global and detailed routing.

- **Pin Access:**

Traditionally the detailed router is responsible to make sure that all the pins of standard cells get successfully connected without causing shorts or overlaps among different nets. However, this is becoming a major challenge due to the 'pin access' issue seen in modern technology nodes when the detailed router is increasingly finding it challenging to access and connect to the pins in some standard cells without causing violation.

Due to the technology scaling and increasing complexity, pin access problem is becoming even more challenging for several reasons Xu et al. (2014, 2015b, 2016); Paper; Alpert et al. (2013). First, the number of routing tracks which are available for the pins of a standard cell is reducing with technology scaling Xu et al. (2014, 2015b, 2016). This results in fewer tapping points (i.e. access points to a pin) to connect to its detailed route. Second, the more intricate structures of standard cells and the power-grid network cause new types of blockage which further complicates pin access Paper. Third, some subtle design rules such as end-of-line (EOL) and min-length rules should be checked for the routes that connect to the pins which create more constraints for how a pin can be accessed Paper; lef.

The issue may be alleviated if pin access is considered during global routing because the locations of the pins of standard cells inside the g-cells are known at this stage. However none of prior work have considered modeling pin access at the global routing stage, and as a result pin access is another major source of mismatch between global and detailed routing.

With the dusk of Moore's law, some novel device structures Qiu and Sadowska (2013); Arabi et al. (2015); Billoint et al. (2015); Lee et al. (2013); Acharya et al. (2016); Panth et al. (2015) are recently developed and can be utilized to improve pin access issue. These

structures are believed to be a promising path to alleviate the pin access issue by redesigning the standard cells.

- **Manufacturing Rules:**

  Some manufacturing rules cause mismatch between global and detailed routing. These issues are not considered in global routing at all but the design rules corresponding to them are required to be satisfied at the detailed routing stage. Below we introduce three examples of most-relevant manufacturing rules.

  - Spacing rules
    Spacing rules ensure the minimum distance between two routes on the same metal layer as allowed by the fabrication technology lef. At the global routing stage, the minimum spacing rule is required only between parallel routing tracks. However there are many more spacing rules that are considered during detailed routing, which are ignored during global routing. For example, there are various types of spacing rules in 45/28 nm technology, including metal spacing rule, end-of-line (EOL) rule, cut-to-cut spacing rule, etc. Jia et al. (2014). A few recent works have done some modeling of these spacing rules, but these are limited to the detailed routing stage Jia et al. (2014); Han et al. (2015); Zhang and Chu (2011, 2012, 2013).
  - Min-area rule
    Min-area rule is another important design rule which is accounted for during detailed routing. It requires a minimum area for every polygon on the metal layer lef. Due to the fixed width of routing wires, the min-area rule is converted to a min-length rule which limits the minimum length of each

routing segment Jia et al. (2014); Xu et al. (2015b). Min-area rule has not been considered during global routing either.

- Self-Aligned Double Patterning-aware rules
  Self-aligned double patterning (SADP) is considered as a promising lithography technique at the 10nm technology node Xu et al. (2014, 2015a). The process of double patterning itself results in more restrictive design rules which need to be satisfied during physical design. Those additional SADP-aware design rules are mainly around the end-of-line of routing segments Xu et al. (2014, 2015a); Han et al. (2015). The most recent work is about the pin access planning for SADP-based manufacturing Xu et al. (2015b); Hsu et al. (2014). However this work is only targeted at the detailed routing stage.

- **Routability Evaluation:**

  Routability is becoming an increasingly important issue in nano-scale VLSI physical design Wei et al. (2012). Traditionally overflow-based metrics are used which include estimating total overflow and maximum overflow of routing resource usage during global routing. However these metrics increasingly fail to give a clear picture of the routability of design for modern technology nodes and hence are another source of mismatch with the detailed routing stage. To reduce the mismatch in evaluating routability between GR and DR, new congestion-based metrics are proposed in Wei et al. (2012); Liu et al. (2013c) to capture the profile of routability of the design. Net congestion-based metrics include $ACN(x)$ (the average congestion of the top $x\%$ congested nets) and $WCI(y)$ (the number of nets with congestion greater than or equal to $y\%$). Edge congestion-based metrics include the values of $ACE(x)$ (the average congestion of the top $x\%$ congested edges). While these met-

rics improve upon the traditional overflow-based metrics, none of them have been evaluated at the detailed routing stage.

The above are the most significant sources of mismatch between the GR and DR stages. In this dissertation we show that it is possible to model some of these issues at the GR stage in order to provide a better starting point to the detailed router. This in turn can reduce the runtime spent at the DR stage. This is because the detailed router is more likely to converge to a solution with fewer violations by following the global routing solution, if the global router has been made aware of the above-mentioned sources of mismatch.

2 OUR CONTRIBUTIONS

This dissertation introduces techniques to bring awareness of various detailed routing issues to the global routing stage. The summary of our contributions is listed as follows.

- **Via-size aware global routing** Shi et al. (2016b)

  To capture the impact of varying-size vias at the global routing stage, we propose "via-aware edge overflow" and "edge-aware via overflow" metrics. We then introduce a fast two-stage layer assignment algorithm which optimizes these metrics. This is followed by a proposed linear programming formulation to further optimize these metrics. In our experiments, first we show significant improvement at the global routing stage is possible in our proposed metrics *without* compromising the traditional metrics of total edge overflow and via count. We also offer a multi-threaded version of our algorithm to further improve its runtime. Furthermore, we show optimizing our proposed metrics at the global routing stage reduces the number of violations seen at the detailed routing stage using the detailed routing tool of Olympus SoC by Mentor Graphics Graphics.

  Our via-size aware global routing technique is discussed in Chapter 3.

- **Via-aware track planning**

  We propose a framework to quickly predict congestion of individual routing tracks inside each g-cell at the global routing stage. A distinguishing feature of our framework compared to prior work is estimating the locations of vias and partial track utilization by a segment of a global net inside each g-cell. We integrate this model

with a novel algorithm for track assignment. There are two ways that our framework can be used. First, it can be used as a 'via analyzer' to determine the via locations inside the g-cells for an existing track assignment solution. In this case we show in our experiments that our 'via analyzer' can improve the prediction accuracy of two recent track assignment techniques. Second, our model can be used to directly analyze an existing global routing solution. In this case our analyzer internally performs track assignment while considering via locations to minimize track overlaps more effectively. A major strength of this work is to measure the error of our framework with respect to an accurate congestion map generated by a commercial detailed router. To do so, we developed a modern design flow which includes a close interaction with the commercial Olympus SoC tool of Mentor Graphics Graphics. We also used one of the latest benchmark suites Bustany et al. (2015) in industry LEF/DEF formats. Our design flow included Tcl scripts to build connections between our core C++ codes and Olympus DB.

Our via-aware track planning procedure is discussed in Chapter 4.

- **Improving pin access and detailed routability with 3D monolithic standard cells** Shi and Davoodi (2017)

  To improve pin access issue, we propose a framework to utilize 3D monolithic (3DM) standard cells. 3D monolithic integration is an emerging technology which has recently been developed and considered as a promising path in the dusk of Moore's law. Here we propose a design flow which transforms traditional standard rows, implemented as 'single-tier 2D' structures, into rows of standard 3DM cells which are folded into two tiers. The main feature of the transformation is that the 3DM cells contain redundant pins to facilitate the pin access problem compared to their 2D counter-

parts. This transformation also frees routing tracks in the two tiers used by the 3DM cells which can be used to further improve pin access and detailed routability. The transformation also preserves layout characteristics such as overall area and number of metal layers for signal routing. In fact besides the use of 3DM cells, the rest of the design is implemented similar to a traditional '2D' IC. Using this transformation, we then propose an Integer Linear Program which routes as many nets as possible on the free 3DM routing tracks, leaving the rest of the nets to be routed via a standard global and detailed router on the metal layers dedicated for signal routing. Our experiments show significant improvement in detailed routability metrics using 3DM cells compared to using 2D standard cells.

Our framework for improving pin access using 3DM cells is explained in Chapter 5.

- **Improving the distribution of congestion during global routing** Shi et al. (2016a)

  We introduce a procedure which takes as input a global routing solution that is already improved for routability based on the traditional overflow metric. It then improves the distribution of congestion while ensuring that the traditional overflow metric is not degraded. Our router is able to significantly decrease the number of edges in undesirable ranges of congestion by optimizing a convex piece-wise linear penalty function. In our experiments, using the already-optimized global routing solutions of the ISPD'11 benchmarks Viswanathan et al. (2011) we show the number of edges which are utilized very close to capacity can be significantly reduced (which implies more flexibility for optimization at the detailed routing stage) while keeping the traditional overflow

metric intact. This work is the first to explicitly target improving the distribution of edge congestion–besides traditional routability metrics–at the global routing stage.

Our procedure to optimize the distribution of congestion at the GR stage is explained in Chapter 6.

It is noteworthy that some other sources of mismatch between GR and DR (introduced in Chapter 1.2) are also covered by the above contributions although they are not specific to a single work introduced above. For example, our proposed via-size aware layer assignment procedure in Chapter 3 not only considers the impacts of via sizes but also includes the consideration of the spacing rules. In addition, we also considered modeling the local nets in Chapter 4 on via-aware track planning. In Chapter 5 on improving pin access, we also considered the local nets in our framework.
Overall, in this dissertation we dived into major sources of mismatch between the GR and DR stages. We showed our work can reduce the gap between global routing and detailed routing by developing DR-level models which can be incorporated at the GR stage.

3 VIA SIZE-AWARE GLOBAL ROUTING

A typical characteristic of advanced technology nodes is significantly-high variation, up to a factor of two, in wire sizes that may exist between *adjacent* metal layers. Routing from a metal layer to its top layer results in using a via which is typically as wide as the wire size on the top layer. The via will then consume wire tracks from the lower metal layer so it acts as a source of local congestion within the (lower layer) g-cell. This source of local congestion is a dynamic one which varies during the global routing stage as a routing solution gets evolved. Moreover, similar to pins in library cells, vias do not scale as well as devices with each technology node so the issue is expected to deteriorate with further technology scaling. For example, in the 32nm technology node, the metal width in the top layer can be up to 20x larger than the one in bottom layer Alpert and Tellez (2010); Alpert et al. (2010).
Figure 3.1 shows an example illustrating the issue. Consider case 2 or 3. They each show a wire that gets connected to a lower metal layer. Here, the used via does not block *additional* routing tracks so from that aspect it can be considered similar to case 1 except for the portion of a routing track that is utilized by the wire. This is because the lower layer always has same or lower track width. However consider case 4 or 5. Here connecting to the top layer with wider track results in the via to actually block available routing tracks in the g-cell. In case 4, the blockage is only corresponding to the left boundary of the g-cell. Therefore unstacked vias can act as local congestion and reduce the number of available tracks that can pass one or more boundaries of the g-cell. Such issue has not been modeled in prior literature.
Some prior work on layer assignment have considered modeling local congestion caused by vias dynamically Lee and Wang (2008, 2009); Liu and Li (2011). Specifically, prior work such as Hsu et al. (2008) intro-

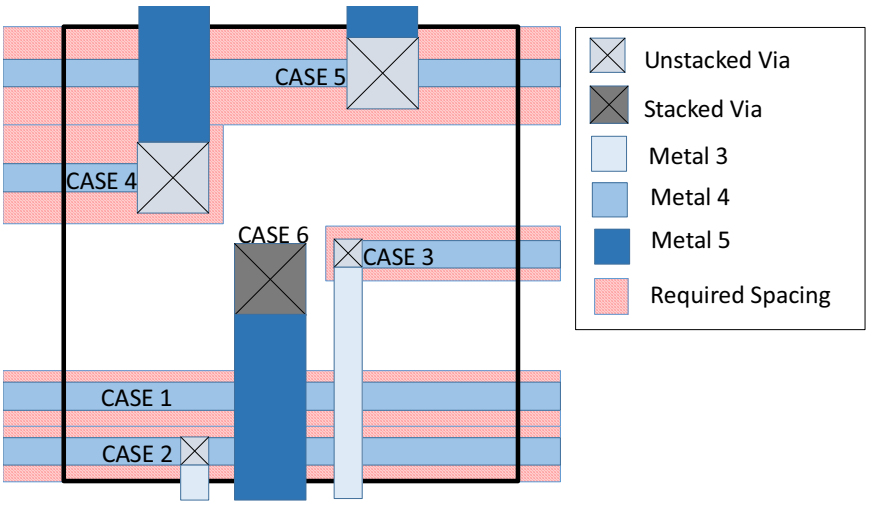


Figure 3.1: Via connecting to the top metal layer blocks additional routing tracks compared to the one connecting to the lower metal layer. See more explanations in Table I.

duced a model for via overflow of a g-cell which took into account the area taken by the used wire tracks, considering the size and spacing requirements, as well as area and spacing requirements for existing vias, and other types of routing blockages inside a g-cell to compute how many new stacked vias could pass through the g-cell. However, prior work typically assumed stacked and unstacked vias in a g-cell have the same size which can be inaccurate in computing the via overflow. Moreover, unstacked vias can impact stacked vias beyond what was considered in prior work. As an example consider Figure 3.1 again. In case 5, a rectangular region as wide as the via size (and spacing requirement) and as long as the g-cell's side is blocked by the unstacked via. This region would have been smaller if the via connection was to a lower layer. As a result, the tile can support fewer stacked vias before having a via overflow and a stacked via such as case 6 cannot be placed in this entire blocked region.

Prior work on layer assignment have done limited study on modeling

local congestion caused by vias Lee and Wang (2008, 2009); Liu and Li (2011). Specifically, prior work such as Hsu et al. (2008) introduced a model for via overflow of a g-cell which took into account the area taken by the used wire tracks, considering the size and spacing requirements, as well as area and spacing requirements for existing vias, and other types of routing blockages inside a g-cell to compute how many new stacked vias could pass through the g-cell. However, prior work typically assumed stacked and unstacked vias in a g-cell have the same size which can be inaccurate in computing the via overflow.
In this work new models are proposed for local congestion caused by stacked and unstacked vias of varying sizes in a g-cell. Specifically, we propose a *via-aware* edge overflow (VA-EOF) metric to model the impact of unstacked vias on the available g-cell boundary (Chapter 3.1). We also extended the existing via overflow model from prior work to an *edge-aware* via overflow (EA-VOF) metric to account for the impact of unstacked vias to better control the stacked ones (Chapter 3.1).

3.1 Via Size-Aware Overflow Models

Before introducing our models we introduce the basic notations used in this work. A global routing instance is defined using a grid-graph $G = (V, E)$. Each vertex $v \in V$ represents a global cell (g-cell) and each edge $e \in E$ represents the common boundary of two adjacent g-cells. For edge $e \in E$, a capacity c_e represents the available length of the g-cell's boundary that can be used for routing after accounting for static routing blockage. Traditionally, when a route t crosses the boundary of a g-cell, it was assumed that it utilizes routing resource equal to the summation of its width w_t and spacing s_t Viswanathan et al. (2011). Utilization of an edge is expressed by $u_e = \sum_{\forall t \ passing \ e}(w_t + s_t)$. The overflow of edge e is given by $o_e = \max(0, u_e - c_e)$. The total edge

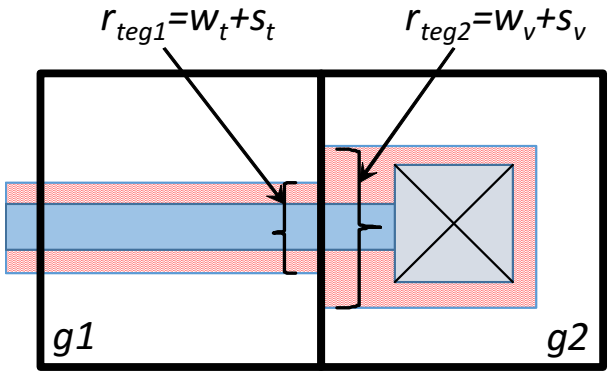


Figure 3.2: The wire needs different amount of routing resource from each side of the common boundary of two g-cells when considering via size and spacing.

overflow metric is given by EOF=$\sum_{\forall e \in E} o_e$.

Via-Aware Edge Overflow Model

Consider a g-cell g and a wire t. In this work we model various cases in which wire t may partially or fully utilize a track inside g which are the cases included in Figure 3.1. A core idea behind both of our overflow metrics is modeling how one wire utilizes routing resources within a g-cell as well as from the boundaries with respect to a single g-cell. First, consider one boundary e of g-cell g. We define $r_{t_{eg}}$ to be the amount of routing resource taken by t from boundary e inside g. Note as shown in Figure 3.2 it is possible that a wire needs different amount of routing resource from the common boundary of two g-cells when vias are considered. Therefore we differentiate with respect to a g-cell boundary. In Table I, column 2, we give the expression for $r_{t_{eg}}$ for the cases shown from the example of Figure 3.1.
Here w_t and s_t are the width and spacing of wire t, and w_v and s_v are the wire width and spacing of the corresponding via. For cases 1 and 2,

Table I: Resource utilization of a g-cell in different cases in Figure 3.1

	r_{t_e}	a_{t_g}
Case 1	L&R: $(w_t + s_t)$	$(w_t + s_t) \times W$
Case 2	L&R: $(w_t + s_t)$	$(w_t + s_t) \times W$
Case 3	R: $(w_t + s_t)$	$(w_t + s_t) \times W/2$
Case 4	L: $(w_v + s_v)$	$(w_v + s_v) \times W/2$
Case 5	L&R: $(w_v + s_v)$	$(w_v + s_v) \times W$
Case 6	-	$(w_v + s_v)^2$

$r_{t_{e_g}}$ is equal to sum of width and spacing of t and edge e could be either of the two boundaries of the g-cell (for passing a horizontal wire). The $r_{t_{e_g}}$ is expressed the same way in case 3 but here edge e corresponds to the right boundary of g as shown in Figure 3.1. For cases 4, the given expression in the table is for the left boundary of the g-cell; for cases 5, the given expression in the table is for both boundaries of the g-cell. They are determined based on the via width and spacing, w_v and s_v, respectively, because the via size is wider than the wire size in these two cases. In this work we do not model the impact of a stacked via on blocking a g-cell boundary (case 6).
We now define the via-aware utilization of an edge e representing the common boundary of two g-cells g_1 and g_2 as follows.

$$u_e^{VA} = \max\Big(\sum_{\forall t \text{ passing } e} r_{t_{e_{g_1}}}, \sum_{\forall t \text{ passing } e} r_{t_{e_{g_2}}}\Big) \tag{3.1}$$

The above equation computes an edge utilization for each side of a common boundary. The via-aware utilization of the edge is then defined as the maximum of the utilization of the two sides. The via-aware overflow of an edge is denoted by o_e^{VA} and expressed as $\max(0, u_e^{VA} - c_e)$. The total via-aware edge overflow is given by VA-EOF=$\sum_{\forall e \in E} o_e^{VA}$. Note that the traditional edge overflow metric (o_e) (EOF) can be considered a special case of via-aware edge overflow (o_e^{VA}). This is because

when via-sizes are not considered, the utilizations of a wire from each side of a common boundary of two g-cells are the same and equal to summation of wire width and spacing.

Edge-Aware Via Overflow Model

We first derive expressions for resource usage of a wire that is routed inside a g-cell which is essentially the area used by the wire. We denote this area by a_{t_g} for wire t and g-cell g. Table I, column 3, gives the expression for a_{t_g} for each of the cases in the example of Figure 3.1. In the table, the new parameter W is the g-cell's width.

The expressions are similar to the boundary expressions (column 2 of the table) expect they require an estimation of the portion of the track that a wire uses inside the g-cell to compute the area. For cases 1, 2, and 5, the expressions are exact because the wires take the entire track. For cases 3 and 4, the expressions are approximate and assume the wire uses half of the track ($W/2$). We used this assumption from prior work Hsu et al. (2008) and find it to be a reasonable assumption if the actual wire lengths inside a g-cell have a uniform distribution. For case 6, the expression is exact and is the area of the stacked via after considering its spacing requirement. Note, it assumes the stacked via has a square shape in order to keep the notation simple, but in case it is not square, the expression can be easily modified to reflect that which is actually considered in our experiments for some benchmarks in which the g-cells are not square.

Next we can derive the edge-aware utilization of a g-cell by a group of wires passing through it as $u_g^{EA} = \sum_{\forall t \text{ passing } g} a_{t_e}$.

We also denote the capacity of a g-cell by c_g which is the available area inside the g-cell after accounting for static routing blockage including an approximation of resources needed for the local nets, for example as described in Wei et al. (2012). The edge-aware overflow of a g-cell is

denoted by o_g^{EA} and expressed as below.

$$o_g^{EA} = \max(0, u_g^{EA} - c_g) \tag{3.2}$$

The total edge-aware via overflow is given by EA-VOF=$\sum_{\forall g} o_g^{EA}$.
The main difference between our edge-aware via overflow model compared to prior work is how a_{t_g} is computed for each wire t that passes g. Specifically, situations that are more accurately handled by our model are cases 4 and 5 in Figure 3.1, when a wire that crosses a g-cell edge connects to a larger unstacked via inside the g-cell, thereby taking more area. This is why the via overflow model is "edge-aware".
Moreover, as an optimization metric, EA-VOF correlates most with controlling the number of stacked vias if minimized in a layer assignment procedure. Therefore EA-VOF can also be thought as also capturing the resource overhead caused by unstacked vias on determining the available area in a g-cell, which in turn influences the allocation of stacked vias in a layer assignment procedure.

3.2 Layer Assignment Framework

Figure 3.3 gives an overview of our layer assignment framework. The input is a projected 2D routing solution. The output is a 3D solution which has the same 2D projection as the input. Our framework has two stages. The first stage is a dynamic programming (DP) algorithm which minimizes a cost function that can be a combination of traditional metrics EOF and via count, as well as our proposed metrics VA-EOF and EA-VOF. It can also be set to remove the traditional EOF metric and only use our proposed metric. We show the impact of both variations when reporting our experimental results.
The DP algorithm operates on a single net and sequentially processes the nets similar to Lee and Wang (2008, 2009); Liu and Li (2011); Liu

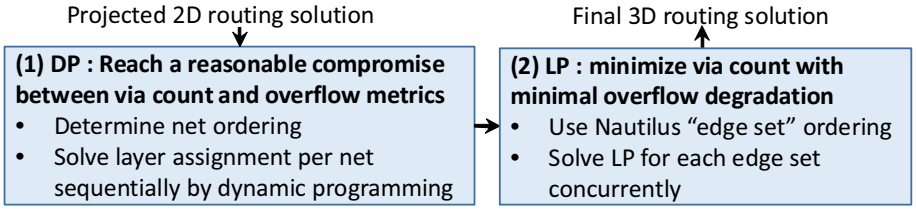


Figure 3.3: Overview of our layer assignment framework.

et al. (2013a). The algorithm results in slight increase in via count because minimizing VA-EOF tends to increase the via count during layer assignment, compared to only minimizing EOF and via count.
The second stage of our framework is solving a simple but elegant linear programming (LP) formulation. It aims to minimize the via count of the solution generated by DP without increase in EOF and with minimal compromise on VA-EOF and implicitly optimize EA-VOF. The LP formulation operates on an "edge-set" which is the set of 3D edges that have the same 2D projection. So it considers all the edges in the same set concurrently. It processes the edge-sets using a novel "Nautilus-shaped" ordering. The LP benefits from the property of totally unimodular polyhedron in optimization theory.

Stage 1: Dynamic Programming

At the first stage, we visit the nets sequentially according to a net ordering. For each individual net, a dynamic programming (DP) formulation is solved which determines the layer assignment for that net. Our DP formulation extends other prior work such as Lee and Wang (2008, 2009); Liu and Li (2011); Liu et al. (2013a) to account for our proposed metrics. We determine our net ordering by evaluating the following "score" for each net n given by $\text{score}(n) = \frac{C_1}{WL(n)} + C_2 \times Deg(n)$, where C_1 and C_2 are two constants, $WL(n)$ is the wirelength of the 2D

projection of net n and $Deg(n)$ is the number of pins of the net. We set $C_1 = 1000$ and $C_2 = 0.4$ in our experiments. Nets with higher score are processed first.
This ordering promotes nets with shorter 2D wire-length and higher pin count to be processed earlier. First, if a net has higher 2D wire-length, it is more flexible during layer assignment because it usually has more flat fragments so it can be processed later. Second, the higher pin count of a net may significantly restrict the number of possible layer assignment solutions. This net ordering has been applied by some other existing layer assignment works Lee and Wang (2008, 2009); Liu and Li (2011); Liu et al. (2013a).
To solve the DP formulation for one net n, we receive as input, the 2D routing tree of the net which we denote by t_n. The output of DP is the layer for each edge $e \in t_n$. We first introduce some notations before discussing the DP algorithm.

- τ_v: subtree rooted at v for a node $v \in t_n$.
- C_v: set of all the children nodes of v. The edge between v and a child node $c_v \in C_v$ is called a "child-edge".
- $CLA(v)$: an array $[l_1, \ldots, l_m]$ representing an assigned layer for each child-edge of v with m children. The i^{th} entry of $CLA(v)$ is the layer assigned to the i^{th} child-edge.

Algorithm 1 shows the procedure called SubTreeLA(v) which determines the layer assignment for all the child-edges of v. It returns a quadruple with four elements which are scores directly related to EOF, via count (denoted by VC), VA-EOF and EA-VOF, respectively.
We start by calling SubTreeLA for the root node of t_n and the procedure recursively calls itself to ensure each node is only processed after its children are processed.

Algorithm 1 DP-based algorithm for a sub-tree τ_v

```
1:  procedure SubTreeLA(v)
2:      if v is leaf then return {0,0,0,0}
3:      end if
4:      bstCLA←unknown; bstScore ← ∞; bstQuad←
5:      {EOF=∞,VC=∞, VA-EOF=∞,EA-VOF=∞ }
6:      for each possible combination of CLA(v) do
7:          s=GetScore(v, CLA(v))
8:          score=s.EOF+s.VC+s.VA-EOF+s.EA-VOF
9:          if score < bestScore then
10:             bestCLA ← CLA(v); bestQuad ← st;
11:             bestScore ← score
12:         end if
13:     end for
14:     assign layers to all the child-edges according to bestCLA
         return bestQuad
15: end procedure
```

When the procedure SubTreeLA starts processing node v, it considers the edge between v and child c_v to be assigned to any layer, and further considers all combinations of layer assignments among all the child-edges of v, $\forall c_v \in C_v$.

For each combination, the function GetScore (Algorithm 2) is then called to compute a corresponding quadruple and score, based on the already-computed $CLA(c_v)$ of the child, and the layer for edge (v,c_v), for all $c_v \in C_v$. Note, the computed quadruple of one combination represents the EOF, VC, VA-EOF, and EA-VOF of τ_{c_v}.

The actual score is computed as shown on line 8 of Algorithm 1. We like to clarify that the 4 metrics of EOF, VC, VA-EOF, EA-VOF are weighted so the normalized values are added to each other on line 8. These weights are 1000, 200, 1, 1, for the above metrics respectively. We assign higher weights to EOF and VC to show the benefits of our new metrics only after these traditional metrics have been fully optimized.

Algorithm 2 DP score of sub-problem

```
1:  procedure GetScore(v, CL(v))
2:      score ← {0,0,0,0}
3:      for each child node c_v of v do
4:          add elements in SubTreeLA(c_v) to score
5:      end for
6:      for each layer l_i assigned to the i^th child-edge do
7:          add EOF of edge (c_v,v) to score.EOF
8:          update score.VC so subtrees of c_v and v connect
9:          update score.VA-EOF for edge (c_v,v)
10:     end for
11:     for each g-cell with same x,y coordinate as v do
12:         update score.EA-VOF for vertex v
13:     end for
        return score
14: end procedure
```

Once a score is determined, among the combinations, the one with the smallest score will be selected as the best one which determines the layer assignment for all the child-edges. In case the best score is tied among multiple assignments, the tie is broken by selecting the assignment with lower EOF, else lower VC, else lower VA-EOF, and else lower EA-VOF.

Stage 2: Linear Programming

As mentioned before, considering VA-EOF makes the DP stage to slightly increase via count because a g-cell experiences overflow earlier when via sizes are considered. However minimizing via count is also a very important objective of layer assignment. Therefore, at the second stage, LP tries to minimize the via count as its only objective with minimal or no degradation in already-optimized VA-EOF and EOF based on the solution generated by DP.

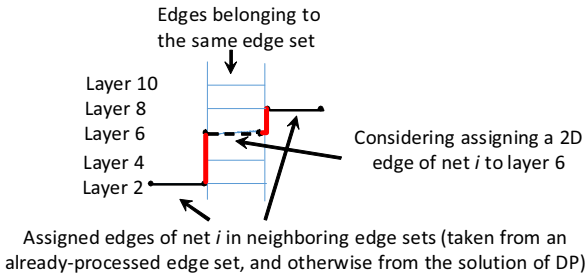


Figure 3.4: Assigning an edge of net i to layer 6 in the considered edge-set results in 2 vias with left neighbor and 1 via with right neighbor. Our LP formulation makes this assignment for all nets in the same edge-set to minimize via count subject to keeping EOF intact and minimal increase in VA-EOF.

Figure 3.4 shows an overview to explain the LP with a simple example. We first group all the 3D edges with the same 2D projection in the same edge-set. Let $G_{2D} = (V_{2D}, E_{2D})$ be the 2D projection of the 3D global routing grid-graph. We denote the edge-set defined by a 2D edge $e \in V_{2D}$ as S_e. The edge-sets are then sorted according to a novel Nautilus-shaped sorting procedure which we discuss later. The edge-sets are processed sequentially using this sorting.

To process an edge-set S_e, we further identify, from the provided 2D routing solution, all the nets whose route contain $e \in V_{2D}$. We refer to this related set of nets for edge-set S_e as N_e. Next an LP is formed which considers all the nets in N_e concurrently. For each net $n \in N_e$, it generates the layer assignment for its 2D edge.

The LP minimizes via count as its objective. When evaluating the assignment of a 2D edge of a net to a specific layer, the number of vias are estimated by looking at the assigned layer for the continuation of the net in neighboring edge-sets. Each individual LP also has a constraint to guarantee that EOF will not increase compared to DP. At the same time, it is effective in controlling any increase in VA-EOF in that edge-set.

Formulation

Consider net $n \in N_e$ and edge-set S_e. Based on our prior definitions, this means the 2D projected route of net n passes from edge $e \in V_{2D}$ and the edge-set S_e is made of all the 3D edges that map to e. During LP, we decide if we assign (this edge of) net n to layer ℓ. So we define variable $x_{n\ell}$ which falls between 0 and 1, and is assigned to 1 if and only if net n is assigned to layer ℓ in this edge-set. Here ℓ refers to any edge (or alternatively corresponding layer) that belongs to the edge-set.
Our formulation is given below.

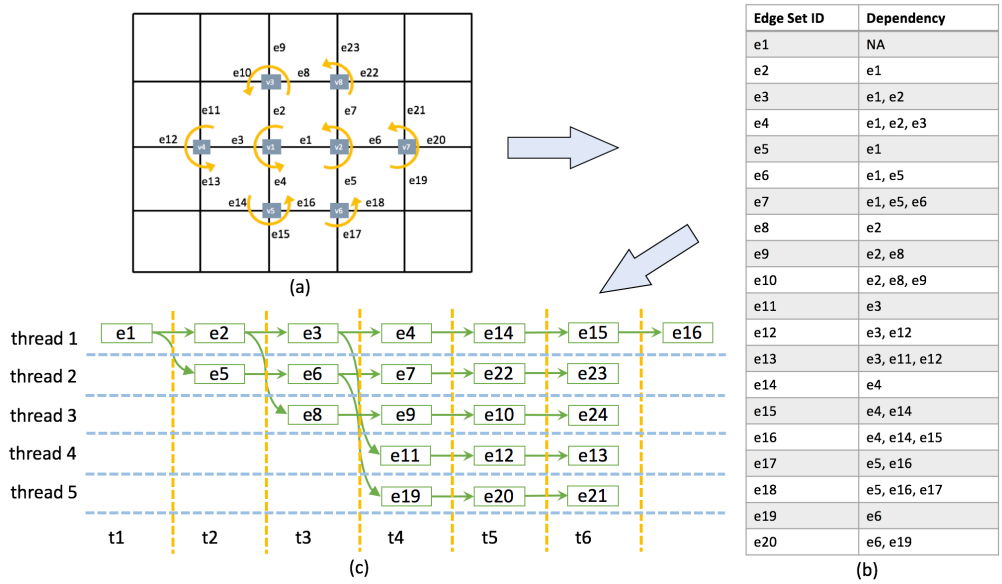
$$\min_{x} \sum_{n \in N_e, \ell \in S_e} w_\ell x_{n\ell}$$

$$\sum_{\ell \in S_e} x_{n\ell} = 1, \qquad \forall n \in N_e \tag{3.3}$$

$$\sum_{n \in N_e} x_{n\ell} \leqslant u_\ell, \qquad \forall \ell \in S_e \tag{3.4}$$

$$0 \leqslant x_{n\ell} \leqslant 1, \qquad \forall n \in N_e, \forall \ell \in S_e \tag{3.5}$$

First we note that this problem is *totally uni-modular* because there exists a bi-coloring row partition Kelly (1985) for every submatrix of the constraint matrix. It implies that each feasible solution of this LP problem is integral by Hoffman-Kruskal theorem Hoffman and Kruskal (2010). With the desirable property of this specific LP, we can feel free to solve this LP to obtain binary solutions.
Inequality 3.4 ensures the number of nets assigned to (a 3D edge in) layer ℓ of edge-set S_e is bounded by quantity u_ℓ. Here u_ℓ is a constant parameter. It is equal to the number of routed nets passing that 3D edge from the layer assignment solution generated by DP. Inequality 3.4 explicitly guarantees that the utilization of each edge does not increase



Edge Set ID	Dependency
e1	NA
e2	e1
e3	e1, e2
e4	e1, e2, e3
e5	e1
e6	e1, e5
e7	e1, e5, e6
e8	e2
e9	e2, e8
e10	e2, e8, e9
e11	e3
e12	e3, e12
e13	e3, e11, e12
e14	e4
e15	e4, e14
e16	e4, e14, e15
e17	e5, e16
e18	e5, e16, e17
e19	e6
e20	e6, e19

Figure 3.5: Illustration of (a) Nautilus-based edge-set ordering; and (b-c) Creating dependency list and edge-set scheduling in the Multi-Threaded Multi-Pass LP (MT-MPLP).

compared to DP. Thus it ensures that traditional EOF will not increase and is effective in limiting any increase in VA-EOF.

The objective of our formulation is minimizing via count for the nets in the edge-set. For each variable $x_{n\ell}$, define weight $w_{n\ell}$ indicating the number of vias if net n is assigned to layer ℓ in the edge-set as it connects to its other fragments in neighboring edge-sets. For the example in Figure 4, assigning net n to layer 6 results in 2 vias with the left and 1 via with the right edge-set so the value of x_{n6} is equal to 3. We assume a solution for the neighboring edge-set as follows. If the edge-set is already processed, we take the solution generated by LP. Otherwise, the solution is taken from the DP stage.

Nautilus Edge-Set Ordering

Edge-sets are processed using an ordering procedure which we call Nautilus ordering, as illustrated in Figure 3.5-(a). The figure shows the 2D projected grid-graph so each 2D edge represents an edge-set. The main idea behind the Nautilus ordering is as follows. First, note that each edge-set has 6 adjacent edge-sets. For example the edge labeled e1 in the figure, has neighboring edge-sets labeled e_2 to e_7. Second, recall in the LP that in order to count the number of vias when assigning a net n to layer ℓ we need to know how the net continues in the adjacent edge-sets. In other words, in the LP for an edge-set we assume the solution for the neighboring edge-sets is already fixed.
This assumption is initially inaccurate when we just start processing the very first edge-set. But as more edge-sets get processed, it becomes more likely that the neighboring edge-sets have been processed by LP to a stable solution. Therefore, the goal of our ordering is to solve neighboring edge-sets consecutively, as much as possible, which we propose to do in a Nautilus-shaped order as shown in Figure 3.5-(a).
We start by selecting the first edge-set (indexed e_1) in the figure which is the edge with the highest via count based on the DP solution generated. The via count for an edge-set is computed similar to the example shown in Figure 3.4.
We store the two vertices corresponding to the selected edge-set (vertices v_1 and v_2 in the figure) in a queue. The ordering proceeds as follows. We select the vertex from the head of the queue, and process the edge-sets connected to it (if they are not already processed) in a counter-clockwise fashion. We then enqueue the neighboring vertices of this vertex, again in a counter-clockwise fashion. The process continues until the queue does not contain anymore vertices.
In the example of Figure 3.5-(a), (after processing edge-set `e1`) we first enqueue v_1 and v_2. We dequeue v_1 and process edge-sets e_2 to e_4. We

Algorithm 3 Multi-pass LP

```
1: procedure MPLP
2:     while true do
3:         Compute new edge-set ordering based on via count
4:         Solve LP for all the edge-sets in order
5:         Exit if the total via count did not decrease
6:     end while
7: end procedure
```

then enqueue the vertices adjacent to v_1 in counter-clockwise if they have not been enqueued before, which are v_3, v_4, v_5. We then dequeue v_2 and the process continues. In the figure, the ordering of edge-sets and the order to enqueue the vertices are shown by edge and vertex labels.

Improvement Using a Multi-Pass Strategy

While the Nautilus-based ordering helps with better estimating the via connecting an edge-set to its neighbors, it is still subject to some error. For example the very first edge-set estimates its via count based on the solution of its neighboring edge-sets from the DP stage, which are then potentially altered when the LP is solved for them later on.
This estimation error is the highest for the first few edge-sets but the Nautilus-based ordering results in less estimation in the subsequent edge-sets because the Nautilus shape makes it more likely that the neighbors of an edge-set are processed by the time the LP is solved for them.
To further improve the Nautilus-based ordering, here we simply repeat the ordering procedure over multiple passes. At the beginning of each pass, first a new ordering is generated (using the same procedure) by sorting based on via count and starting with the edge-set which has the highest via count. Then the LP is solved for each edge-set according to the order.
The Multiple Pass LP (MPLP) procedure terminates when no reduction

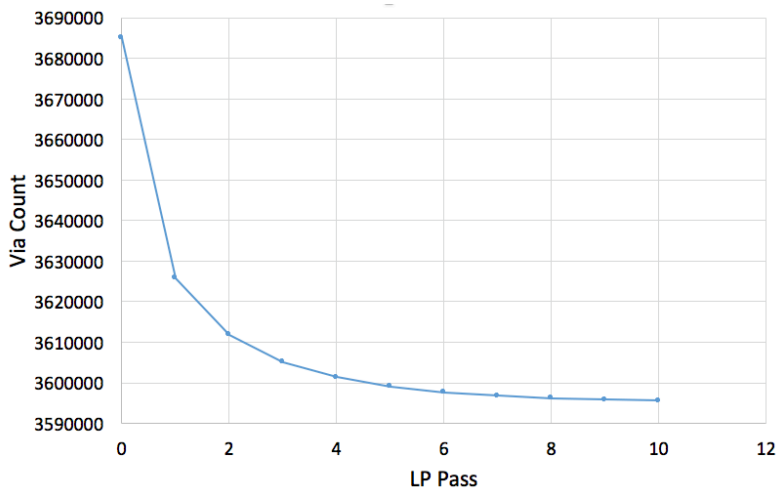


Figure 3.6: Reduction in via count over 10 passes in `s18`.

in via count is obtained in two consecutive passes. Algorithm 3 gives a high-level pseudo-code. Figure 3.6 shows the reduction of via count at each pass for benchmark `s18`. This simple MPLP procedure results in less estimation error and gradual convergence to a final solution over 10 passes but it increases the runtime.
Next, we show a multi-threaded procedure to speed up MPLP without any loss in its solution quality.

Improving the Multi-Pass Strategy via Multi-Threading

Figure 3.5-(a) shows Nautilus-based edge-set ordering in a single pass of MPLP. For example assume it shows pass k in which the edge-set ordering is decided based on the via count of the edge-sets from the solution from pass $k-1$.
We first define a dependency list between the ordered edge-sets based on the following observation. We define edge-set i to depend on any of

edge-sets indexed from 1 to $i-1$, if and only if they are neighbors of i. For example, for the edge-set ordering in Figure 3.5-(a), e_5 only depends on e_1 because e_2 to e_4 are not neighbors of e_5. In other words, the layer assignment results in e_2 to e_4 won't affect the LP formulation built for e_5. Therefore, in a multi-core environment, it is possible to schedule e_5 as soon as e_1 terminates, i.e., schedule e_5 to run simultaneously with e_2. Note this multi-threaded variation keeps the solution of the MPLP intact and only results in speedup. We denote the multi-threaded variation of MPLP by MT-MPLP.

Figure 3.5-(b) shows the dependency list for the top 20 edge-sets given in Figure 3.5-(a). The dependency constraint requires each edge-set to wait for all the edge-sets in its dependency list to finish before getting processed to ensure the solution of MPLP remains intact after multi-threading. Figure 3.5-(c) shows how edge-sets can be assigned to threads to ensure the dependency constraints given in Figure 3.5-(b) are honored.

Algorithm 4 shows the MT-MPLP procedure. At the beginning, all the edge-sets are scheduled for multiple threads according to a scheduling procedure (which will be shortly given in Algorithm 5) to ensure dependency constraints are met. Once the scheduling finishes, we start all the threads together. At each step, all the active threads solve the LP for their corresponding edge-sets and wait for other active threads to finish before doing the next step.

Algorithm 5 illustrates the scheduling algorithm for all the edge-sets. To schedule edge-set i, we first check its dependency list. Suppose $d(i) < i$ is the highest index in the list and edge-set $d(i)$ has been scheduled at time $t_{d(i)}$. Therefore i can be scheduled no earlier than $t_{d_i} + 1$. Any thread free at $t_{d_i} + 1$ can be selected to schedule i. Otherwise, we can either create a new thread if allowed or delay the time slot by 1 and then repeat the similar process for time $t_{d_i} + 2$.

Algorithm 4 Multi-threaded multi-pass LP

```
1: procedure MT-MPLP
2:     Build the dependency list for each edge-set
3:     Schedule all the edge-sets using the dependencies
4:     & determine M (the number of threads)
5:     for t = t_0, t_1, ..., t_M do
6:         Let each active thread solve a scheduled edge-set
7:     end for
8: end procedure
```

The maximum allowed number of threads in Algorithm 5 is limited by the number of threads supported in the system or can be set to a smaller number by the user. The terminal command `lscpu` returns the maximum allowed number of threads on a Linux machine. For our machine, we had 4 cores and 2 thread per core. Therefore we used 8 threads for the MT-MPLP algorithm in our experiments.
Since all the active threads are solving a similar-sized LP (given that they all correspond to same-sized edge-sets), we expect that it takes similar CPU time for each active thread to finish its job in one step. So all the active threads execute their jobs simultaneously and wait for all the other threads to finish before executing the next scheduled job.
We note in the MT-MPLP procedure, all the working threads are created in the beginning. They are not killed until the MT-MPLP finishes. So the overhead of creating / deleting threads is negligible in our case.

3.3 Experimental Results

We implemented our two-stage layer assignment algorithm in C++. We refer to our program as VALA (for Via size-Aware Layer Assignment). We used CPLEX 12.6 to solve the linear programs. All experiments ran on a Linux machine with a 3.4GHz Intel 4-core CPU and 12GB of mem-

Algorithm 5 Scheduling algorithm for MT-MPLP

```
1: procedure SCHEDULING-MT-MPLP
2:     Set number of threads M = 0
3:     for i = 1, ... do
4:         d(i): highest index in the dependency list of i
5:         Set target time to schedule i to t_i = t_{d(i)} + 1
6:         while Edge-set i has not been scheduled do
7:             if Free thread k exists at time t_i then
8:                 Schedule edge-set i at time t_i to thread k
9:             else
10:                if New thread can be created then
11:                    Create a new thread
12:                    Schedule edge-set i at time t_i
13:                    Increment M by 1
14:                else
15:                    Increment t_i by 1
16:                end if
17:            end if
18:        end while
19:    end for
20: end procedure
```

ory. The input solution to VALA is a 2D projected global routing solution. We first used the NCTU-GR 2.0 Liu et al. (2013a) router to generate a 3D global routing solution and then created the 2D-projected version of that solution to feed as input to VALA. We used the ISPD'11 Viswanathan et al. (2011) benchmarks. Specifically, for each benchmark we chose the best placement solution among the contestants (which are posted on the 2011 contest website).

Table II shows technology information used in this work. Information about wire size and spacing were provided in the ISPD'11 benchmarks. As can be seen if a via is between metal k and metal $k+1$, we let the via size be w_{k+1}, which is the wire size on metal $k+1$. We set the minimum via spacing equal to the via size, which is a common rule used in other

Table II: Information on wire size and spacing, and via enclosure rules in our experiments

Layer	Wire width	Wire spacing	Down-via size	Up-via size
M1	1	1	N/A	1
M2	1	1	1	1
M3	1	1	1	1
M4	1	1	1	2
M5	2	2	2	2
M6	2	2	2	2
M7	2	2	2	4
M8	4	4	4	4
M9	4	4	4	N/A

benchmarks (for example, ISPD'15 benchmarks Bustany et al. (2015)). There is no knowledge about the technology node for the ISPD'11 benchmarks. However to do detailed routing we need to specify the `DATABASE MICRONS` in Olympus SoC. We set the minimum wire size in bottom layers to 200nm which is the same value used in the ISPD'15 benchmarks Bustany et al. (2015).
The dimensions of the global routing grid-graph is given for each benchmark in column 2 of Table III. The size of g-cells were either 32x32 or 32x40 in a benchmark.
In our first experiment we made comparison between the following cases.

- **Base**: We ran a variation of the first stage of our framework based on dynamic programming in which all considerations for the new metrics (i.e., VA-EOF and VA-EOF) were eliminated. This was done by eliminating VA-EOF and VA-EOF when computing the score of each solution of a subproblem during dynamic programming. The only considered metrics were via count and EOF.

- **DP**: We ran the first stage of our framework (DP) and considered all metrics including VA-EOF and EA-VOF.
- **DP+LP**: We ran the two-stage VALA framework completely by applying DP followed by LP.

We verified the global routing results of Base were similar to the 3D global routing results generated by NCTU-GR Liu et al. (2013a) in EOF and via count. The Base case allows us to make a fair comparison to evaluate VALA because it is a variation of our existing implementation and during dynamic programming, we like to keep a controllable behavior to precisely measure the effects of enabling or disabling EOF, via count, VA-EOF, and EA-VOF which are primary goals of this work.
We note in our dynamic programming (DP) implementation of all the above variations (including Base) we used the exact same cost function with the same weights for the metrics (EOF, VA-EOF, EA-VOF, VC) across all benchmarks, and there was no benchmark-specific tuning. This is except for the case when a metric was disabled when its corresponding weight in the cost function was set to 0.

Global Routing Comparisons

Here we made comparison in traditional total edge overflow (EOF), via count, total via-aware edge overflow (VA-EOF), and total edge-aware via-overflow (EA-VOF) at the global routing stage. These metrics were defined in Section 3.1.
Table III shows the results. For Base, we report the actual values of these quantities. For DP and DP+LP we report the percentage increase compared to Base. We also report the runtime of each approach in seconds. (For DP(No-EOF+LP), we do not report any runtime because it is very similar to DP+LP.) The runtime of DP+LP includes the runtime of DP. We also only report the EOF for Base because the EOF of all 3 variations

Table III: Comparison at the global routing stage

Design	Base Case					VALA(DP)				VALA(DP+LP)				VALA (DP(No-EOF)+LP)		
	EOF	#Vias	VA-EOF	EA-VOF	Runtime(seconds)	#Vias	VA-EOF	EA-VOF	Runtime(seconds)	#Vias	VA-EOF	EA-VOF	Runtime(seconds)	#Vias	VA-EOF	EA-VOF
s1	0	4613288	40160	480516	76	1.03%	-80.81%	-32.97%	87	0.61%	-77.55%	-37.31%	150	0.08%	-74.69%	-13.99%
s2	1686	6441030	311558	1702887	1022	3.91%	-64.34%	-26.64%	1007	2.39%	-59.50%	-38.99%	1226	0.90%	-57.21%	-32.05%
s4	272	3332687	141114	1424344	62	2.05%	-61.26%	-14.86%	61	0.72%	-58.04%	-27.49%	105	0.34%	-56.13%	-20.83%
s5	0	4674230	136574	1168377	95	2.12%	-67.40%	-20.23%	107	1.12%	-63.04%	-31.40%	196	0.22%	-61.64%	-24.78%
s10	0	6334716	284108	2271959	155	1.66%	-50.14%	-26.86%	153	1.66%	-50.14%	-26.86%	270	0.29%	-44.54%	-21.63%
s12	0	9047463	425732	4827756	105	3.13%	-59.67%	-5.34%	121	0.87%	-53.19%	-23.53%	194	0.25%	-52.08%	-20.58%
s15	0	6204090	220290	1900414	89	2.33%	-66.50%	-23.89%	100	0.98%	-61.48%	-37.34%	163	0.40%	-60.38%	-20.41%
s18	0	3684947	128384	2129915	46	1.87%	-61.17%	-10.99%	52	-0.23%	-55.51%	-30.39%	87	-0.61%	-53.53%	-28.96%
AVERAGE					206	2.26%	-63.92%	-20.22%	211	1.01%	-58.60%	-31.06%	299	0.23%	-57.52%	-22.90%

of VALA are identical to Base except for benchmark `s2` in which all 3 variations of VALA have slightly lower EOF compared to Base. So, overall, EOF remains same or unchanged in all VALA variations.
As can be seen DP allows decrease in VA-EOF and EA-VOF by on-average 63.92% and 20.22% respectively. But it increases the via count by on-average 2.26%. However after applying LP, the increase in via count reduces to on-average 1.01% of the base case. This is expected because the objective of LP is to explicitly minimize via count of the LP's solution. This still results in substantial improvement in VA-EOF to 58.60%. The EA-VOF improves to 31.06% because minimizing vias also help improve EA-VOF.
For DP(No-EOF)+LP, the improvements in VA-EOF and EA-VOF are 57.52% and 22.9%, respectively. Note it has the most reduction in the number of vias (better than DP+LP), and on-average only 0.23% of Base. The average runtimes of Base, DP, and LP stages were 206, 211, and 298 seconds, respectively. As expected the runtimes of Base and DP are both very fast and similar to each other. The runtime of LP is even faster even though it operates on many edge-sets because each LP has a very small size. The runtimes of DP(No- EOF)+LP almost identical to DP+LP.

Evaluation of the Multi-Threaded Multi-Pass Strategy

Next, we made another comparison between LP, the multi-pass LP (MPLP) and the multi-threaded version (MT-MPLP). The comparisons are made in via count, total via-aware edge overflow (VA-EOF), total edge-aware via-overflow (EA-VOF) and total elapsed times at the global routing stage. In all experiments we used the same dynamic programming solution which was DP(No-EOF).
Table IV shows the results. For MPLP (and MT-MPLP), the number of LP passes are reported as well which is on-average about 26 passes per benchmark.

Table IV: Comparing LP with MPLP and MT-MPLP

Design	DP(No-EOF)+LP					DP(No-EOF)+(MPLP)					DP(No-EOF)+(MT-MPLP)				
	#Vias	VA-EOF	EA-VOF	#Pass	Runtime(s)	#Vias	VA-EOF	EA-VOF	#Pass	Runtime (s)	#Vias	VA-EOF	EA-VOF	#Pass	*Runtime (s)
s1	0.08%	-74.69%	-13.99%	1	150	-0.17%	-74.95%	-14.02%	5	387	-0.17%	-74.95%	-14.02%	5	124
s2	0.90%	-57.21%	-32.05%	1	1226	-0.10%	-57.76%	-34.18%	14	3186	-0.10%	-57.76%	-34.18%	14	1279
s4	0.34%	-56.13%	-20.83%	1	105	-0.44%	-56.76%	-24.05%	27	1246	-0.44%	-56.76%	-24.05%	27	209
s5	0.22%	-61.64%	-24.78%	1	196	-0.35%	-61.83%	-26.99%	37	3335	-0.35%	-61.83%	-26.99%	37	510
s10	0.29%	-44.54%	-21.63%	1	270	-0.94%	-45.70%	-26.14%	53	6372	-0.94%	-45.70%	-26.14%	53	930
s12	0.25%	-52.08%	-20.58%	1	194	-0.94%	-52.74%	-26.25%	30	2283	-0.94%	-52.74%	-26.25%	30	391
s15	0.40%	-60.38%	-20.41%	1	163	-0.40%	-61.09%	-23.25%	25	1649	-0.40%	-61.09%	-23.25%	25	293
s18	-0.61%	-53.53%	-28.96%	1	87	-1.75%	-52.90%	-33.73%	18	677	-1.75%	-52.90%	-33.73%	18	130
AVERAGE	0.23%	-57.52%	-22.90%	1	299	-0.63%	-57.96%	-26.08%	26	2392	-0.63%	-57.96%	-26.08%	26	483

As can be seen by applying multiple passes of LP runs, all the metrics of via count, VA-EOF and EA-VOF are improved in MPLP compared to LP. Most notably, the via count is actually decreased for each benchmark (compared to the Base case). MT-MPLP has the exactly same results as MPLP except for runtime because MT-MPLP is a multi-threading variation of MPLP which guarantees no change in the results of MPLP. Due to the license restriction of CPLEX in our machine, we were not able to run multiple instances of CPLEX in parallel. So the runtimes in the right most column of Table IV are estimated run times based on the analysis of scheduling procedure given in Algorithm 5 (but using the actual runtimes taken by CPLEX to solve each LP sequentially). The analysis was made on our target machine with 4 cores and 2 threads per core.

Evaluation of the Nautilus Ordering

We made comparison with a random ordering in which edge-sets were randomly shuffled. (The LP corresponding to each edge-set was then solved in the random order.) We compared this ordering with Nautilus ordering in terms of runtime and global routing metrics which were measured in terms of number of vias, VA-EOF, and EA-VOF. Specifically we compare the multi-pass variation (MPLP) with Nautilus ordering and with ordering determined by random shuffling. In random shuffling the ordering changed at each new pass. The stopping criteria in both orderings was the same, i.e., do not initiate a new pass if no reduction in via count can be seen in the past two consecutive passes. Table V shows the results. As can be seen the global routing metrics are only slightly better with Nautilus ordering because of more reduction in via count, VA-EOF, and EA-VOF metrics. (The reported numbers are percentages compared to the same Base case given in the previous experiment.) However the runtime of MPLP with Nautilus ordering is significantly faster. This is because Nautilus ordering resulted in a sig-

nificantly smaller number of passes to converge. The number of passes was on-average 26.1 with Nautilus ordering as opposed to on-average 39.1 with random shuffling.

Detailed Routing Comparisons

In this experiment we made comparison between the Base case and VALA (DP+LP) at the detailed routing stage. We created a setup to feed in our global routing solutions to the Olympus-SoC detailed router Graphics. Specifically, global routes were loaded into Olympus using a custom tcl script. The script parses a routing file generated by us which was in the standard format used in the ISPD 2011 contest. Horizontal and vertical wire segments on metal layers are added using the `create_wire` command and vias are inserted one at a time using the `create_via` command. We imitate the Olympus global router's operation by placing end points of global wire segments and vias at the center of global cells.

Furthermore, to convert the placement info from the Bookshelf format to LEF/DEF, we wrote a perl script similar to the "convert" function in Liu et al. (2013b). This differs from the NCTU-GR converter in a few ways. First, it adds rectilinear-shaped cells to the OVERLAP layer, to properly represent their shapes in Olympus and remove placement violations. Second, the NCTU-GR converter scales via size on a metal layer based on the minimum width of only that layer. Our converter allowed vias to be additionally scaled when changing layers with differing minimum widths. Third, our script outputs a Verilog file to specify each net's pin connections. Finally, pins on cells with blockage are at their default/actual locations which is metal1 according to the Bookshelf guidelines for such cells that are not declared NI (non-interfering). This causes issues with pin access, so blockage is removed in order to allow these pins to be routed. The router still is forced to route around these

Table V: Comparison of nautilus ordering with random shuffling during MPLP

Design	Nautilus ordering					Random shuffling				
	#Vias	VA-EOF	EA-VOF	#Pass	Runtime (s)	#Vias	VA-EOF	EA-VOF	#Pass	Runtime (s)
s1	-0.17%	-74.95%	-14.02%	5	387	-0.17%	-75.18%	-14.05%	32	2020
s2	-0.10%	-57.76%	-34.18%	14	3186	-0.10%	-57.87%	-34.22%	62	10371
s4	-0.44%	-56.36%	-24.05%	27	1246	-0.43%	-56.22%	-23.98%	43	1948
s5	-0.35%	-61.83%	-26.99%	37	3335	-0.34%	-61.77%	-26.97%	32	2899
s10	-0.94%	-45.70%	-26.14%	53	6372	-0.91%	-45.58%	-25.93%	55	6607
s12	-0.94%	-52.74%	-26.25%	30	2283	-0.90%	-52.62%	-25.92%	29	2211
s15	-0.40%	-61.09%	-23.25%	25	1649	-0.38%	-60.97%	-23.27%	36	2330
s18	-1.75%	-52.90%	-33.73%	18	677	-1.73%	-52.90%	-33.53%	24	885
average	-0.64%	-57.92%	-26.08%	26	2392	-0.62%	-57.89%	-25.98%	39	3659

blockages as the track router strictly follows the global routing solution, and only violates blockages in order to access these pins.
The ISPD 2011 benchmarks provide neither pin shape information nor the cell library information. In our experiments, in order to incorporate such information for detailed routing we did the following. The ".nets" files of benchmarks tell the (x, y, z) position of each pin. We defined a net pin as the smallest square (1 unit per side) in the layout at the locations specified by the ".nets" file. Each unit was set to 200nm as discussed before.
After importing our global routing solution into Olympus, we then ran the `track_route` command which creates an initial detailed routing solution based on our global routing solution. For our runs, in order to respect routing blockages, we instructed the track_route to strictly follow the global routing solution during track assignment. Track routing ran in congestion mode (as opposed to timing mode). To save time, the track router ran with "medium" effort, which runs two passes. The first minimizes metal and cut space violations, and the second minimizes polygon-based violations.
These experiments ran on an Intel(R) Core(TM) i7-2600 CPU. The results are shown in Table VI. For each benchmark we report the number of DRC violations. For Base we report the actual number of violations. For the two VALA variations we report a percentage improvement compared to Base. As can be seen both VALA variations improve the number of DRC violations. The improvements for DP+LP and DP(No-EOF)+LP are on-average 3.6% and 9.1%, respectively. Note, these improvements are made only by changing the layer assignment of the same 2D global routing solution. In DP(No-EOF)+LP, when we completely replace optimizing EOF with our proposed metrics, the reduction in the number of DRC violations is higher which suggests our proposed overflow metrics may act as a good replacement for traditional EOF during global rout-

Table VI: Comparison at the detailed routing stage

Design	Base Case	VALA(DP+LP)	VALA(DP(No-EOF)+LP)
s1	69471	4.8%	-2.7%
s2	351969	-4.5%	-11.3%
s4	158892	-5.2%	-6.2%
s5	141145	-5.6%	-12.9%
s12	624414	-0.9%	-4.2%
s15	295567	-2.8%	-10.6%
s18	186862	-10.6%	-16.1%
AVERAGE		-3.6%	-9.1%

ing.

For benchmark `s1` we observe an overall less improvement compared to the other benchmarks. If we look back at Table III, we can observe from the 'Base Case' that the VA-EOF and EA-VOF metrics are significantly smaller in `s1` compared to the other benchmarks. Also the other ISPD'11 benchmarks are similar in terms of number of g-cells, number of nets and number of cells. Lower VA-EOF and EA-VOF in `s1` implies that the local congestion issues caused by varying-size vias are not as significant compared to other benchmarks.

In summary, as tested by detailed router in Olympus SoC tool, this set of solutions shows the least DRC violations. It implies that our metrics of VA-EOF and EA-VOF are reasonable and feasible in practice.

Impact of Varying the G-cell Size

In this experiment our goal is to study the impact of the EA-VOF model with change in the g-cell size. Recall Table I showed how much routing resource is consumed by the routing segments and vias under different cases. If the cell width W changes, the parameter α_{t_g} (reflecting the routing resource usage inside the g-cell) will also change. In Table I, for cases 1 to 5, α_{t_g} has a factor of W in its expression. In cases 1, 2 and 5, in

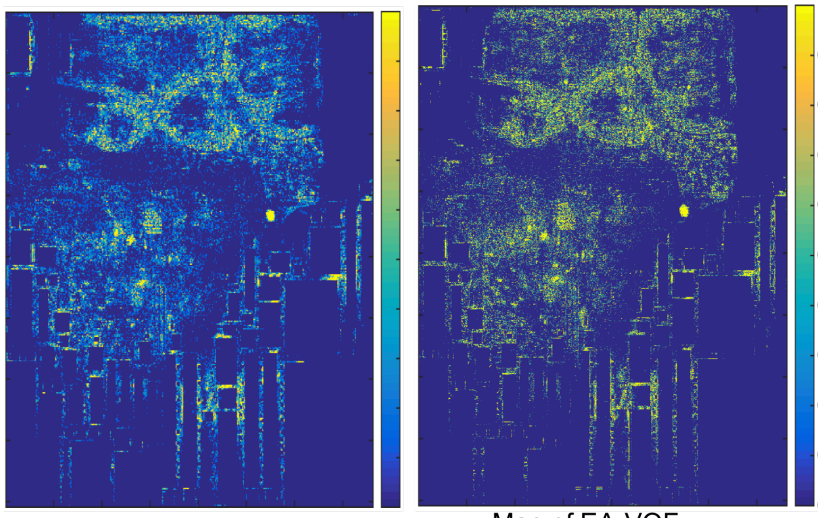


Figure 3.7: Impact of varying the g-cell size on EA-VOF metric.

which the routing segment goes through the g-cell completely, the consumed routing resource is exactly given by a_{t_g}. So the routing resource won't be over-estimated if W increases. In cases 3 and 4, the factor $W/2$ is an approximation about how far the routing segment enters the g-cell from one side. However whether this factor will over- or under- estimate the routing resources is independent of W.

To see the impact of changing the g-cell size, we manually modified the ".route" file of one benchmark (`s18`) and made the g-cell half its size and then applied global routing in each case. Figures 3.7 shows the maps of EA-VOF for regular g-cell size and half g-cell size on metal layer 4. This is the layer in which the via sizes are different when connecting to layer 3 versus connecting to layer 5 as specified in Table II. For the EA-VOF maps, a normalized number between 0 and 1 was generated for each g-cell. This was by dividing the EA-VOF of the g-cell to the g-cell area.

When comparing the two EA-VOF maps with each other, the one with

half g-cell size has higher EA-VOF estimates which can be seen by the brighter colors in its map. This could be because, if a route fragment partially passes a regular-sized g-cell under case 3 or 4, then the same fragment will break into two pieces when using half-sized g-cells: one piece completely passes through a g-cell (under case 1, 2, or 5) and the other piece passes the second g-cell under case 3 or 4. So overall there will be a higher number of g-cells falling under cases 1, 2, and 5 which naturally results in higher normalized EA-VOF values because of more resource usage in cases 1, 2, and 5 compared to cases 3 and 4. This means that for the whole layout there will be less approximation error with half-sized g-cells because of fewer percentages of g-cells falling under cases 3 and 4.

4 VIA-AWARE TRACK PLANNING FOR GLOBAL ROUTING

In this work in order to reduce the gap between GR and DR, we propose a framework to estimate local congestion seen at the DR stage within the GR stage. Our framework can be used as an analyzer to estimate the routing resource usage seen inside each g-cell. While our framework considers local nets inside the g-cells, its main feature is determining the locations of vias on the routing tracks. Via locations in turn allow estimating partial track utilization for better estimation of local track congestion (both utilization and overlap) inside each g-cell. This information can potentially be used as feedback to improve the global router, for example by adjusting the modeling of routing resources in the global routing grid-graph using vertex or edge capacities Viswanathan et al. (2011); Wong et al. (2016), or it can be fed to a routability-driven placer for further optimization. Note, the scope of this work is limited to the analysis part; there is just not enough space to discuss global routing and placement -based optimizations.

There are two ways that our analyzer can be used. First, it can analyze an existing track assignment of a GR solution to estimate via locations and partial track utilization inside the g-cells. Recently a number of fast track assignment techniques have been proposed but none of them estimate via locations. We show in our experiments that our analyzer can improve the prediction accuracy of two recent track assignment techniques Zhang and Chu (2013); Wong et al. (2016) with respect to the congestion seen at the DR stage obtained by a commercial tool. Second, our analyzer can be directly used to analyze an existing GR solution. This is because we also integrated our model of local track congestion with a newly-proposed track assignment algorithm to more effectively reduce track overlaps-when the tracks are fully and partially utilized. For example in Figure 4.1, estimating via location and partial track uti-

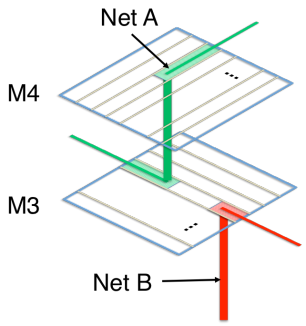


Figure 4.1: Finding via locations from the (potential) track assignment of the segments of a global net allows better estimation of track utilization inside each g-cell.

lization of the segment of net A (on layer M3) allows better planning of track assignment for the segment of net B, thus avoiding track overlap inside the g-cell. In this case, our analyzer is also compared with recent track assignment techniques. We show better prediction of track overlaps using the congestion seen at the DR stage as reference.

Our algorithms are runtime-efficient and our analyzer runs fast as shown for the ISPD'15 detailed routability-driven placement benchmarks Bustany et al. (2015).

Before we discuss our techniques, we give an overview of two related works. These two recent works aim at reducing track overlap which is similar to the objective used by us. Consideration of other objectives such as yield, timing, coupling capacitance Cho et al. (2008); Hu et al. (2005) is beyond the scope of this dissertation.

First, RegularRoute Zhang and Chu (2013) performs fast detailed routing by doing track assignment on a panel-by-panel basis. As shown in Figure 4.2, each panel corresponds to a row of g-cells on the same layer. Based on the GR solution, for each panel, a set of segments of the global nets are identified. These segments compete for the routing tracks inside the panel.

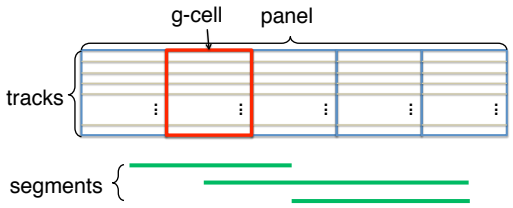


Figure 4.2: Definition of segments and panel from Zhang and Chu (2013).

To determine the track assignment for each segment in a panel, RegularRoute solves a maximum-weight independent set formulation which greedily assigns segments to tracks by prioritizing them according to factors such as segment length, via/pin connection, and density at g-cell boundaries. The assignment aims to maximize the number of non-overlapping segments. RegularRoute then performs additional optimizations to finalize the track assignment. A DR solution can be constructed from the track assignment in the end by using vias to connect consecutive segments of the same net on their assigned tracks. In this case vias may be assumed to be located at the middle of the tracks in the g-cells. The panel-by-panel strategy of RegularRoute is effective in minimizing track violations by packing as many non-overlapping segments on the tracks of the same panel.

A recent work NTA Wong et al. (2016) provides an alternative and fast track assignment algorithm. After an initial assignment which includes consideration for local nets, NTA proceeds to a negotiation-based overlap reduction stage. Specifically this stage is done by visiting the segments in a panel-by-panel manner; for each panel, segments with overlap are ripped up and re-assigned to reduce a cost function which depends on metrics such as degree of overlap, wire-length (based on the pin locations), blockage, and history. The rip-up and reroute in NTA allows explicit reduction of degree of overlap between the segments (as opposed to maximizing number of overlap-free segments such as in

Chang et al. (2008); Batterywala et al. (2002); Zhang and Chu (2013) in order to provide routability feedback to the global router.
Both NTA and RegularRoute have the limitation that they only consider overlap reduction without accurately accounting for partial track overlap inside the g-cells when a via is used. This is inherently because segments are viewed independently during overlap reduction. However to estimate the via locations, it is essential to simultaneously view segments of the same net belonging to more than one panel.
In this work, we propose a track assignment algorithm which utilizes a net-based processing however the net ordering is guided by a panel-based processing. Similar to prior works, we consider groups of segments on a panel-by-panel basis however, whenever a segment is considered, then all other segments belonging to the same net (on different panels) will be also be considered and assigned to tracks simultaneously. This hybrid approach allows using the benefits of panel-by-panel assignment similar to RegularRoute and NTA but also incorporates a net-by-net processing to estimate the via locations inside the g-cells. Similar to NTA, our work aims to minimize the degree of segment overlap. However we consider both fully and partially utilized tracks to compute and reduce the track overlaps. Also unlike NTA and RR, we actually evaluate our prediction accuracy with the intra-gcell congestion as seen at the DR stage, and generated by a commercial tool.
In the remainder of this Chapter, we explain our model to estimate via locations and partial track assignment in Chapter 4.1. Our track assignment algorithm is explained in Chapter 4.2. Experimental results are given in Chapter 4.3.

4.1 Modeling the Via Locations

To find the location of a via we assume we are given the track assignments of two consecutive segments (in the same global net (g-net)) which are connected by the via. In case the two segments are apart by more than one layer, we determine the locations all the vias connecting the intermediate layers which may fall under stacked and unstacked scenarios.

For a given track assignment of a global routing solution, our model can be repeatedly applied to estimate the via locations of every pair of segments connected by a via. It can also be applied within our proposed track assignment algorithm which will be explained Chapter 4.2.

We first introduce some notations. Let capacity c_{tg} be the portion of track t falling inside g-cell g. Let utilization u_{tg} be the total length of segment(s) of g-nets that fall on track t inside g-cell g. Using these quantities, we define a 'total track utilization' seen inside g-cell g as below.

$$\mathtt{TrU}_g = \sum_{\forall \text{track t in g}} u_{tg} \tag{4.1}$$

We also define a 'total track overlap' metric for a g-cell g denoted by $\mathtt{TrOV}_g$ as below.

$$\mathtt{TrOV}_g = \sum_{\forall \text{track t in g}} \max(u_{tg} - c_{tg}, 0) \tag{4.2}$$

The above computes the sum of the track overlaps for all tracks in g from the utilization and capacity of each track.

Next we discuss how c_{tg} and u_{tg} are computed in detail in order to compute the $\mathtt{TrOV}_g$ and $\mathtt{TrU}_g$ metrics.

Track Capacity of A G-cell

To compute a specific c_{tg}, we initialize it to be equal to the length of that boundary of g which runs parallel to the track as shown in Figure 4.3-(a). Next, the lengths of pre-routed nets and obstacles on t are deducted from this initial value. In this work, pre-routed nets and obstacles include power/ground mesh, standard cells, macro obstacles, and the detailed routes of the clock nets which are the inputs to our framework. It also includes a detailed routing estimate for each local net which we discuss in Chapter 4.2.
In this work, pre-routed nets and obstacles include power/ground mesh, macro cell macro obstacles, and the detailed routes of the clock nets which are the inputs to our framework. Our framework ensures that the routing tracks used by the pre-reouted nets and blocked by the blockages are not used as it runs to predict the violations inside each g-cell. It also includes a detailed routing estimate for each local net which we discuss in Chapter 4.2.

Via Location and Track Utilization of A G-cell

Assume we are given a g-cell g that includes some segments of g-nets and track t which passes g. The quantity u_{tg} represents how much length is consumed by the segments on track t inside g. (In our track assignment framework, this quantity is initially set to zero and then updated every time a new segment is assigned to t inside g.)
We estimate track utilization for 3 distinct cases. These cases depend on whether a segment connects to a subsequent segment in another layer, in which case a via location is estimated inside each of the two g-cells that include the via. The cases also depend on the number of layers that two subsequent segments are apart from each other which requires estimating (potentially-different) locations for each via in the intermediate

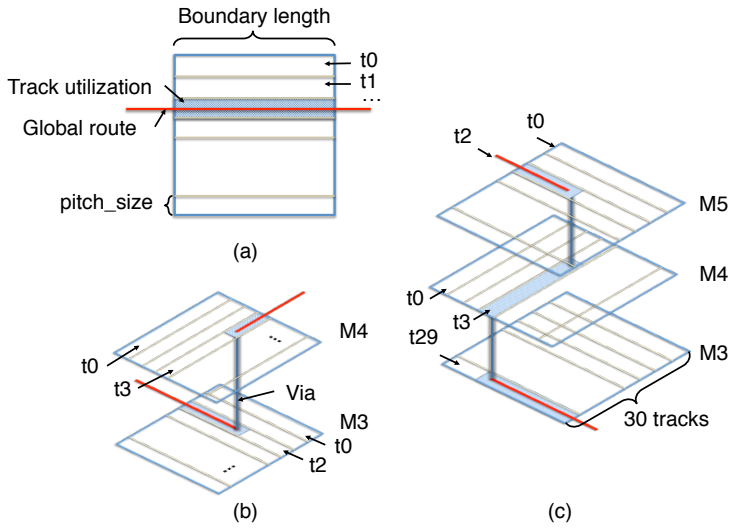


Figure 4.3: Cases for estimating via locations inside a g-cell.

layers. For each case, we first discuss how vias are located and then we assume track utilization is the corresponding length obtained from the via locations. Also we assume track utilization includes the area used by the vias on the track.

First, which is the simplest case, a segment completely passes through a g-cell. In this case no via is used and the track utilization taken by the segment equals to the g-cell's boundary length as shown in Figure 4.3-(a).

Second, if the two consecutive segments of a g-net are located on adjacent layers, they are connected by a single via. For a considered track assignment for these two segments, the via location is easily computed from the track indexes (track locations) of the two segments in the two g-cells. Specifically, the via location lies on the intersection area of the two tracks in 2D space [1].

[1]Note, the analysis relies on the assumption that in practice, the direction of routing tracks alternate in adjacent layers.

For example, as shown in Figure 4.3-(b), one segment is assigned to track t2 on M3 and the other one is assigned to t3 on M4. The via location is determined by computing the overlap area of t2 and t3. Next, the track utilizations are updated using the via locations: for the above example 4 $\times$ `pitch_size` units are added to the utilization on t2 on M3 and 3 $\times$ `pitch_size` units are added to the utilization on track t3 on M4 inside the corresponding g-cells.

Third, and the most complex case is when two segments of a g-net are apart by more than one layer. In this case, the two segments need to be connected by multiple vias. Here the vias may be stacked or unstacked. In the case of unstacked vias which is the more generic case, we consider introducing additional, local, segment(s) in the intermediate layer(s) to connect the two segments to each other. For example, as shown in Figure 4.3-(c), the assigned routing tracks of the two segments of the g-net are t29 on M3 and t2 on M5. In our approach, we introduce a local segment on M4 and estimate a track for it. The local segment on M4 falls completely inside the g-cell and connects the two global segments on M3 and M5.

To estimate the track(s) for the local segment(s) in the intermediate layer(s), we evaluate all track options for the local segment(s). Each evaluation considers a temporary track assignment to the local segment(s). For each track assignment, we compute the via locations; this is done similar to the second case, i.e., by looking at the overlap point of assigned tracks of local and global segments in adjacent layers. Using the via locations, we update the track utilization inside the affected g-cells for each option. In the end, among all track assignment options for the local segment(s) we pick the one which results in minimum $\texttt{TrOV}_g$ metric when added over all the affected g-cells.

In the example in Figure 4.3-(c), if track t3 is selected on M4, we update the track utilizations as follows: 27 $\times$ `pitch_size` units are added to

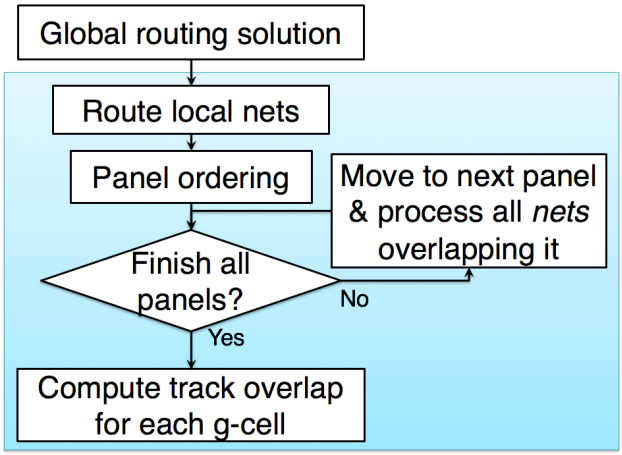


Figure 4.4: Overview of our framework.

the utilization of track t29 on M3, 28 $\times$ `pitch_size` units are added to the utilization of track t3 on M4, and 4 $\times$ `pitch_size` are added to the utilization of track t2 on M5. These are determined by looking at the overlapping points of the assigned tracks at the adjacent layers (falling under case 2) to determine the via locations.
Finally, in a special case when the via locations are found to be the same point in different g-cells, the length of the local segment in the intermediate layer(s) is 1. It indicates that the segments of the g-net will be connected by a set of stacked vias to each other.

4.2 Track Assignment Framework

Figure 4.4 shows an overview of our framework. We first do fast detailed routing for all the local nets. A local net has all its pins located inside the same g-cell. A detailed route for a local net is composed of local segments, each of which falls on a specific track inside a g-cell. The local segments are also connected by vias which we determine in

our framework. The impact of the detailed routes of the local nets is reflected in our framework by updating the track utilizations inside the affected g-cells.

To route the local nets, we observe detailed routings of the local nets are typically on M2 and M3, the lowest two routable layers which allow routing in vertical and horizontal directions. For example, in the detailed routing results generated by Olympus SoC tool Graphics, 46% of all the detailed wirings of the local nets are vertical routes in M2 and 44% are horizontal routes in M3, accounting for over 90% of all detailed wirings. With this observation, our framework routes each local net by only using vertical routes in M2 and horizontal routes in M3 inside the g-cells located above the pins of the local net.

Specifically, we first create one vertical trunk (V-trunk) on a specific track on M2. The range of the V-trunk covers the span of the local net based on the locations of its pins.

After creating the V-trunk, all the pins of a local net are connected to it by adding horizontal branches (H-branches) on M3. For each local net, to decide the track to be used as its V-trunk, we pick the track which minimizes the $\texttt{TrOV}_g$ metric. This is done based on the local nets processed so far and based on other existing static blockages which were given as part of the input to our framework.

After all the local nets are routed, we sort the panels based on the number of segments of global nets (g-nets) which fall on each panel. The panels are then visited in descending order. For each visited panel p, we create a detailed route for all the g-nets which have at least one global segment on p. This is assuming the g-net has not been processed before as part of another panel. Algorithm 6 shows the details.

First, each time the procedure RoutePanel is called for a panel, it sorts all segments that fall on it by their lengths. The segments are then visited in descending order. For each segment s, the algorithm calls DRouteNet

Algorithm 6 Track routing for one panel

```
1: procedure RrocessPanel(panel p)
2:     sort all the global segments on p by length
3:     for each global segment s do
4:         let gnet(s) be the global net containing s
5:         if gnet(s) is NOT 'processed' then
6:             DRoute(gnet(s))
7:             mark gnet(s) 'processed'
8:         end if
9:     end for
10: end procedure
```

which creates a detailed route for the entire g-net which includes s across multiple panels. This is assuming the net has not been processed/detailed routed before by an already-visited panel.
The detailed route of each g-net is created in order to minimize the sum of the $\texttt{TrOV}_g$ metric over the affected g-cells. (The $\texttt{TrOV}_g$ metric is updated every time a new g-net is processed over the g-cells included in the g-net). Next we discuss the details of the DRouteNet procedure. Our framework continues until all panels are processed.

Detailed Routing of A Single Global Net

The DRouteNet procedure takes as input a g-net (specified by its pin locations and set of global segments), and outputs a detailed route for it. Creating a detailed route includes making a track assignment for each segment. The track assignment should in part ensure the pins of a g-net which fall on specific tracks connect to its segments. It also needs to find the via locations inside the g-cells. The detailed route of a g-net is done with the goal to minimize the sum of $\texttt{TrOV}_g$ over the affected g-cells which in part depends on the track utilizations of the g-nets that have already been detailed routed according to Algorithm 6 and the static

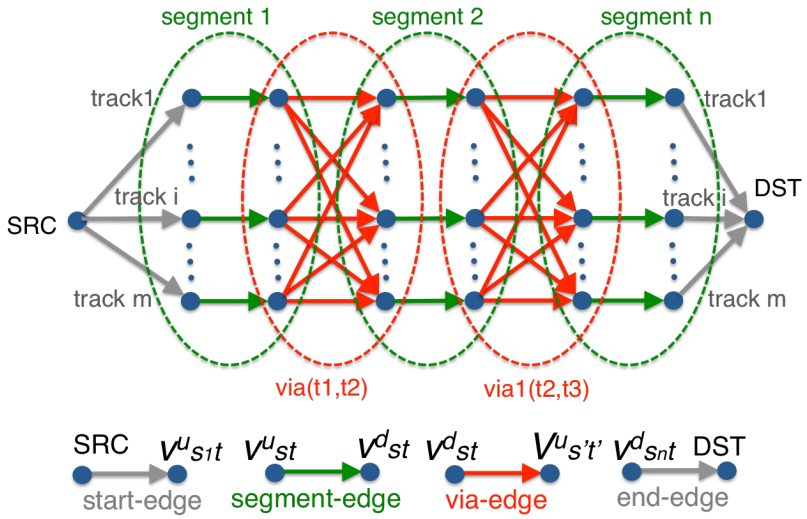


Figure 4.5: Graph model for detailed routing of one g-net.

obstacles and local nets that were considered initially by our framework. In DRouteNet we propose a procedure which simultaneously solves the problem of track assignment and finding via locations. It optimally solves the detailed routing problem for one net, given our track utilization and overlap models.

First, a weighted graph is constructed to represent all track assignment options for each segment of a g-net, and all corresponding vias options in any two consecutive segments. Figure 4.5 shows how this weighted graph is constructed for a two-pin g-net.

(Multi-pin nets are decomposed and processed as union of two-pin subnets.) First, two vertices namely `SRC` and `DST` representing the two pins are created. Next, for every global segment s of a g-net and track option t, we add an upstream and a downstream vertex for the two ends of s. We denote these by v^{u}_{st} and v^{d}_{st}, respectively. A directed edge is then added from v^{u}_{st} to v^{d}_{st}. Figure 4.5 shows the set of edges corresponding to the track options for each segment as a group in green. We refer to

this type of edge in the graph as a 'segment-edge'.
Next, for each via connecting two consecutive segments such as s and s', we add directed edges for every possible combination of their track assignments. Figure 4.5 shows the set of edges corresponding to the via options between two consecutive segments as a group in red. Specifically, an edge is added from v^{d}_{st} to $v^{u}_{s't'}$ as shown in Figure 4.5. We refer to this type of edge in the graph as a 'via-edge'.
Finally, a set of 'start-edges' are added from the SRC vertex to v^{u}_{st} if s is the first segment and for every feasible track t. Similarly, a set of 'end-edges' are added from v^{d}_{st} to the DST vertex, if s is the last segment of the global net, for every feasible track t connecting to the DST pin. Feasible tracks are the ones that the SRC or DST pins fall on. Figure 4.5 shows a simplistic example when all tracks are considered feasible.
Next, the edge weights in the graph are computed for each category of edges. First, the weights of start-edges and end-edges are set to 0. For each segment-edge between v^{u}_{st} and v^{d}_{st}, assuming s is not the first or the last segment of the g-net, the edge weight is equal to sum of $\mathtt{TrOV}_g$ over the affected g-cells. The affected g-cells are the ones that are included in s except the two end g-cells of a segment. (This is because the track overlaps in the first and last g-cells of a segment will be reflected in the corresponding via-edges.) If s is the first segment of a g-net, the edge weight will additionally include the track overlap of the g-cell containing the SRC pin. Similarly, if s is the last segment, the edge weight will additionally include the track overlap of the g-cell connecting to the DST pin.
Finally, for each via-edge connecting v^{d}_{st} to $v^{u}_{s't'}$ for consecutive tracks s and s' on tracks t and t', respectively, the edge weight is defined as follows. We compute the via location and subsequently compute the track overlap in the two g-cells that are connected by the via in s as well as all the g-cells in the layers between s and s'. This is according to the

procedure given in Chapter 4.1. The weight of a via-edge is the sum of the track overlaps in these g-cells.
Once the graph is constructed, we find the min-cost path from `SRC` to `DST`. The path identifies one segment-edge for each segment and one via-edge for each via. It produces the track assignment of all the segments while accurately considering the impact of via locations. The procedure finds the optimal track assignment for the given edge weights. It is also run-time efficient because the graph size only depends on the track count in a g-cell and number of segments in a g-net which are both small in practice.

4.3 Experimental Results

We implemented our analyzer in C++ on a Linux machine with a 2.8GHz Intel CPU and 12GB memory. We built an evaluation platform using the Olympus SoC tool of Mentor Graphics Graphics. Our evaluation infrastructure is shown in Figure 4.6. We first read the input files in LEF/DEF format for the 45nm ISPD'15 detailed routability-driven placement benchmarks Bustany et al. (2015). Then we executed the built-in commands of `place_global` and `place_detail` to place each design. Next we ran global and detailed routing for the clock net and then executed the built-in `route_global` for all the signal nets to generate global segments for each net. We refer to this solution as *Olympus-GR* in our experiments.
We then proceeded with detailed routing. Specifically we used the `track_route` command which according to the Olympus manual (v2015.2) generates an initial detailed routing solution and includes information on via location and partial track utilization and track overlap inside the g-cells. This command took longer to run per design as we report. We refer to this solution as *Olympus-DR*.
Moreover, after generating the global routing solution by Olympus-GR,

Table VII: Comparison of different techniques with respect to Olympus-DR as reference.

Design	Error in Track Overlap				Error in Track Utilization					Runtime (sec)			
	RR	NTA	TraPL	%g-cells	Olympus-GR	RR/NTA	RR+ViA	NTA+ViA	TraPL	RR	NTA	TraPL	Olympus-DR
des_perf_1	19.0%	14.2%	9.4%	98.5	20.7%	21.3%	18.4%	17.7%	11.5%	0.3	3.3	46.1	7200
des_perf_a	8.6%	6.8%	5.5%	46.0	11.4%	11.4%	10.0%	11.3%	8.3%	0.4	11.6	55.9	3000
des_perf_b	7.1%	5.8 %	5.2%	92.0	11.7%	12.0%	10.4%	12.0%	8.1%	0.3	5.2	48.4	1800
edit_dist_a	18.6%	13.6%	10.4%	71.0	15.1%	15.9%	14.0%	14.7%	11.9%	0.8	20.0	115.4	14400
fft_1	15.1%	11.1%	9.0%	99.7	18.4%	19.4%	17.0%	16.1%	12.7%	0.1	1.1	11.6	2400
fft_2	9.6%	6.9%	6.9%	78.9	13.2%	14.5%	13.5%	14.9%	9.3%	0.1	1.8	10.2	1200
fft_a	8.4%	6.8%	7.3%	34.1	12.0%	11.6%	11.1%	10.0%	10.9%	0.2	12.0	16.1	1500
fft_b	7.9%	6.3%	6.9%	31.9	10.1%	9.8%	9.3%	9.3%	10.7%	0.2	4.7	17.7	1200
matrix_mult_1	18.6%	13.8%	9.3%	99.9	19.0%	20.0%	11.2%	17.6%	17.7%	0.5	12.0	65.1	10800
matrix_mult_a	8.0%	6.4%	5.7%	25.0	11.3%	11.1%	9.9%	10.5%	7.9%	0.6	29.3	68.5	1800
matrix_mult_b	5.7%	4.7%	4.8%	30.3	8.7%	8.6%	7.9%	9.5%	7.4%	0.7	24.4	73.6	1800
pci_bridge32_a	5.9%	4.7%	4.7%	69.1	11.2%	9.8%	8.9%	9.9%	7.6%	0.1	1.5	9.1	240
pci_bridge32_b	2.4%	2.1%	2.3%	34.7	6.7%	5.5%	5.2%	6.4%	5.4%	0.1	2.1	9.7	120
AVERAGE	10.4%	7.9%	6.7%	62.4%	13.0%	13.0%	11.8%	12.3%	9.4%	0.33	9.92	42.11	3650

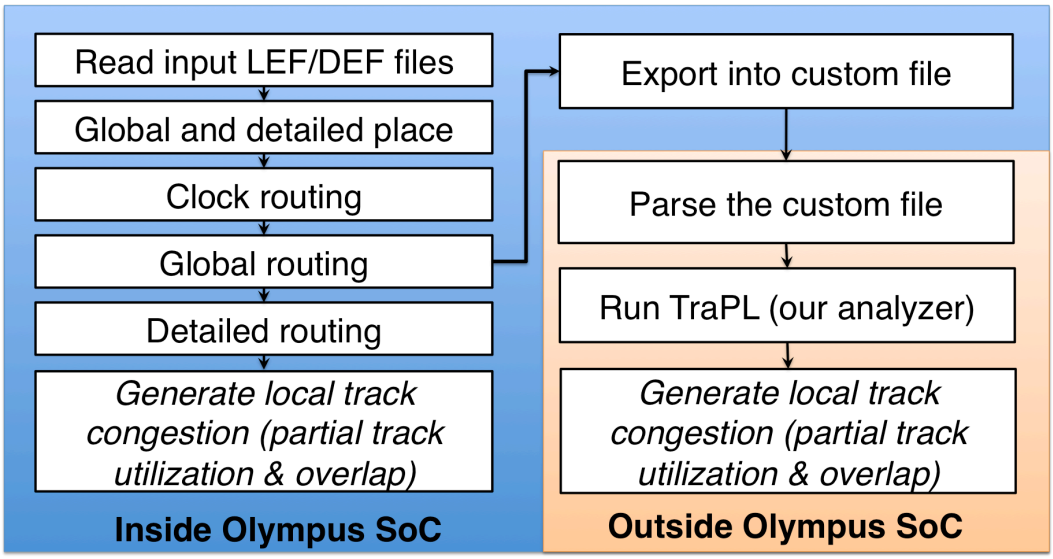


Figure 4.6: Overview of our evaluation infrastructure.

we fed the global routing solution as input to our analyzer. Specifically, we wrote and sourced a Tcl script in Olympus tool to export the information of built-in DB such as layout dimensions, netlist, placement of cells, exact physical information of various types of blockages and the global routing segments for each global net. The above information was then stored in a custom file and provided as input to the C++ code implementing our analyzer. Our framework computed a total track overlap ($\texttt{TrOV}_g$) and utilization ($\texttt{TrU}_g$) in individual g-cells g which we compare with other approaches and Olympus-GR in terms of 'mismatch' errors computed with respect to Olympus-DR as reference.
We made comparison with our implementation of RegluarRoute Zhang and Chu (2013) (denoted by RR) and with NTA Wong et al. (2016) techniques. These techniques both did track routing on the Olympus-GR solution. The RR and NTA techniques were discussed in the beginning

of this Chapter.
For fair comparison, both RR and NTA used the same procedure for routing the local nets as the one in our analyzer which was explained in Chapter 4.2.
We use our analyzer in two ways. First we use it to determine the via locations and consequently partial track utilization and overlaps inside the g-cell for existing track assignment solutions generated by RR and NTA. We refer to this variation of our framework as *ViA* (for Via Analyzer). In the second way we directly apply our analyzer to the Olympus-GR solution. As explained in Chapter 4.2, this variation does track assignment while minimizing track overlaps, accounting for partial overlaps inside the g-cells. We refer to this variation of our analyzer as *TraPL* (for Planning Local Track congestion).

Comparison of Track Overlap

In our first experiment we compare the $\texttt{TrOV}_g$ metric of different approaches with respect to Olympus-DR as reference. We only considered those g-cells g which contained at least one via; this is because this work is about estimating via locations inside the g-cells. Also the number of such g-cells was quite high in our benchmarks; the percentage of g-cells containing at least one g-cell compared to total number of g-cells is given in Table VII column 5, and was on-average 62.4% across the benchmarks.
Recall the $\texttt{TrOV}_g$ metric reflects the degree of track overlap inside each g-cell and is computed using Equation 4.2. Since RR and NTA do not explicitly consider via locations, we assumed all vias are located at the middle of the corresponding track inside the g-cell which we used in order to compute the track overlaps. However in TraPL we used our model of via location to compute the track overlap.

We compute the $\texttt{TrOV}_g$ metric for each approach. We also compute what we consider as the 'actual' value of this metric. This is done by measuring this metric using the via locations inside each g-cell as generated in Olympus-DR.
Next, for RR, NTA, and TraPL, we compute the absolute value of difference in $\texttt{TrOV}_g$ between each approach and Olympus-DR for all g-cells. We add this error over all g-cells, divide it by the total number of g-cells, and report it as a percentage per benchmark in Table VII. Each entry can be viewed as a 'mismatch' error in estimating track overlaps due to via locations inside the g-cells that contain vias.
In Table VII we report the error for RR, NTA, and TraPL. As can be seen from the table, TraPL has the smallest error over all approaches, on-average 6.7% across all benchmarks. NTA and RR consider minimizing track overlaps and number of overlaps respectively but they do not consider via locations and partial track utilization in computing the overlaps.
Overall we show that integrating track assignment with our model of via location to minimize track overlap results in the smallest mismatch with respect to Olympus-DR.

Comparison of Track Utilization

Table VII also shows comparison for total track utilization inside a g-cell denoted by $\texttt{U}_g$ and computed by Equation 4.1. T his metric can be very different from track overlap. For example two g-cells may have all their tracks fully utilized but one may have 0 overlaps while the other may have a very high overlap because all segments are assigned to the same track.
For this approach we additionally compare the solution of Olympus-GR and measure track utilization. Since the Olympus-GR solution does not provide any via locations, we assume all vias are located at the cen-

ter of the g-cells. For NTA and RR, we note that the track utilizations are equal to each other so the same column (column 7) in the table represents both techniques; in fact when considering track utilization in a g-cell, NTA and RR are only different from Olympus-GR because local nets are considered in them.
The reason NTA and RR have the same track utilization per g-cell is because they do not consider via locations so all utilizations are computed with respect to the middle of a track whenever a via is used. This utilization is independent of how track assignment is done. In fact different track assignment procedures only result in different track overlaps as we showed in the previous experiment.
Finally, in this experiment we also report results for two cases of NTA+ViA and RR+ViA. In this case, we use our 'via analyzer' to analyze the track assignment solution generated by NTA and RR and estimate the via locations inside the g-cells for each case.
When considering the track utilization metric ($\texttt{TrU}_g$), we similarly measure the absolute value of difference in this metric between each approach and the Olympus-DR. We report this as a percentage over all g-cells containing vias, just like the previous experiment. As can be seen our approach (TraPL) drops the average error from about 13% (in Olympus-GR / NTA / RR) to 9.4%.
We can also observe that our analyzer reduces the error in NTA and RR. For example for RR, the error drops from on-average 13% (in RR) to 11.3% (in RR+ViA).
So overall we show in this experiment that TraPL can reduce the mismatch error in estimating the track utilization inside the g-cells. It can also be used as an analyzer to reduce the mismatch of the track assignment solutions generated by RR and NTA by estimating the via locations.

Comparison of Runtime

Table VII reports the runtimes of RR, NTA, TraPL, and Olympus-DR. The runtime of our 'via analyzer' (denoted by ViA) was negligible compared to the runtimes of the other approaches so they are not reported in the table. As can be seen, RR, NTA, and TraPL all have significantly faster runtimes than Olympus-DR. Their runtimes are all feasible to be used at the GR stage in order to analyze a GR solution. Our TraPL algorithm supports some degree of parallelism by allowing independent groups of panels to be processed simultaneously. We have observed this opportunity in our preliminary analysis of the benchmarks and plan to pursue this extension in our future work.

5 IMPROVING DETAILED ROUTABILITY AND PIN ACCESS WITH 3D MONOLITHIC STANDARD CELLS

A main issue complicating the detailed routability effort is the pin access challenge which at a fundamental level occurs in dense layouts with devices implemented with a very fine pitch size. An effort from academia showed that a new device structure called VeSFET can significantly improve pin access by allowing two-sided routing Qiu and Sadowska (2013). Recently, industry is exploring a new technology, namely 3D monolithic (or 3D VLSI) Acharya et al. (2016); Arabi et al. (2015); Billoint et al. (2015); Panth et al. (2015) which among its various features allows creation of '3D' logic cells. These cells can be implemented on two separate tiers as opposed to their '2D' single-tier counterparts. The tier can then be accessed separately.

To implement a 3D cell, one way is to fold a 2D cell by placing the NMOS portion on the top tier and PMOS portion on the bottom tier Arabi et al. (2015); Billoint et al. (2015); Lee et al. (2013). Here by a 2D cell we mean a standard logic cell that is implemented on a single tier. Figure 5.1 from Lee et al. (2013) shows an example of how the folding is done for an inverter. The left side shows the 2D cell. The right side shows its 3D monolithic counter-part which we denote by a '3DM' cell in this work. The figure shows how pin Z can be made accessible in both the top and bottom tiers by creating separate pins on each tier and internally connecting them using monolithic inter-*tier* vias (MIVs).

In this work we utilize the above model of transforming 2D cells into 3DM cells to highlight the substantial benefits to improve detailed routability and pin access. We note while the above folding technique for a 2D cell was shown before, it's usage to improve routability has not been studied so far. Specifically, first we propose a design flow to translate

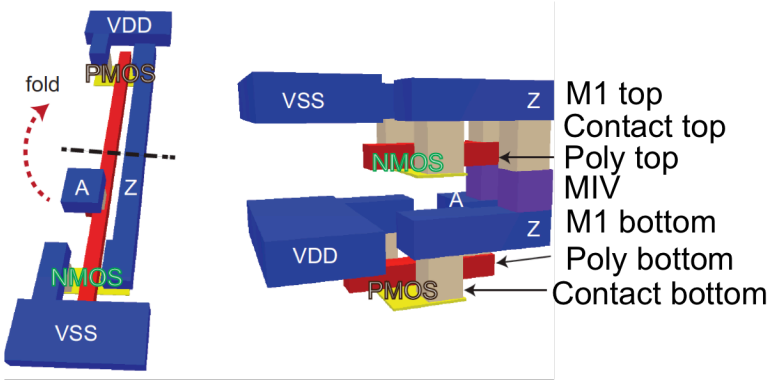


Figure 5.1: Two layouts of an inverter from Lee et al. (2013) (a) a single-tier standard cell, and (b) it's folded two-tier 3D monolithic one.

a placed design with 2D cells into one which has 3DM cells with redundant pins. The translation creates an extra tier to implement the 3DM cells but it ensures the layout area and number of layers for signal routing (i.e., M2 and above) remain the same, while creating new opportunities to improve routability and pin access. Figure 5.2 shows overview of our design flow. Compared to a traditional flow, we first apply a transformation to translate a pair of back-to-back 2D standard *rows* (with shared VDD or VSS in the middle) into a pair of 3DM standard *rows* with free routing tracks in the middle, moving the VDD/VSS lines to the sides and individual tiers. This transformation is explained in Chapter 5.1.

Then we try to route as many nets as possible on the top tier of metal 1 (denoted by M1T) and the bottom tier (denoted by M1B). We do this in two steps: First, we apply cell-to-cell track routing to directly connect adjacent cells on the same tier, by also taking advantage of redundant pins. This step is explained in Chapter 5.2.

Next, we discuss an Integer Linear Program which routes as many remaining nets as possible on the free tracks running between a pair of

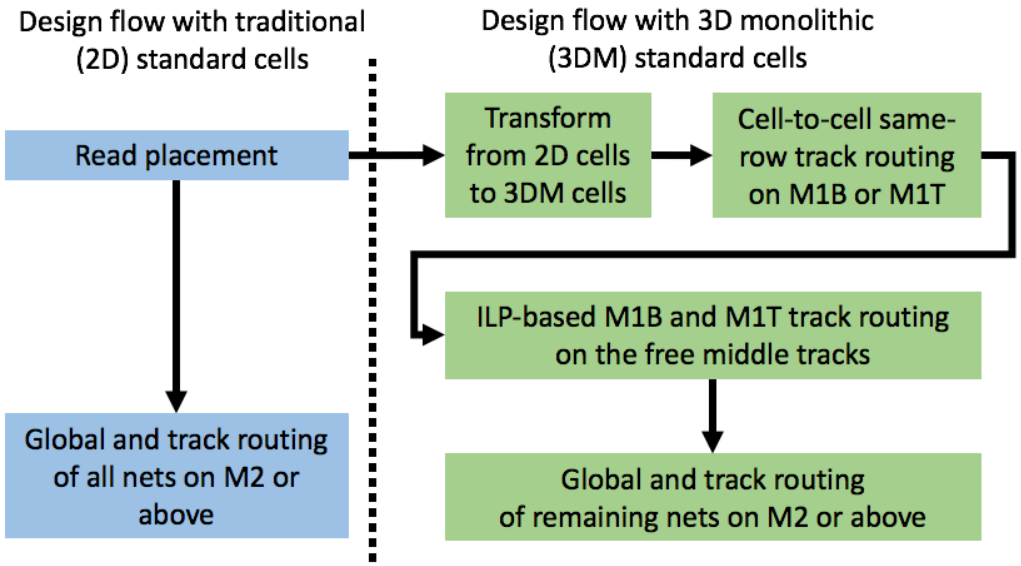


Figure 5.2: Design flow with traditional (2D) standard cells versus 3D monolithic (3DM) standard cells.

3DM rows on M1T or M1B. We discuss this process in Chapter 5.3. Finally, all remaining unrouted nets are global and detailed routed on M2 and above layers using a commercial global and detailed router.
Our experiments show that the number of segment violations and DRC errors are reduced on-average by 29.8% and 20.7%, respectively. This work is the first to highlight the promise of 3D monolithic cells to improve detailed routability via better pin access.

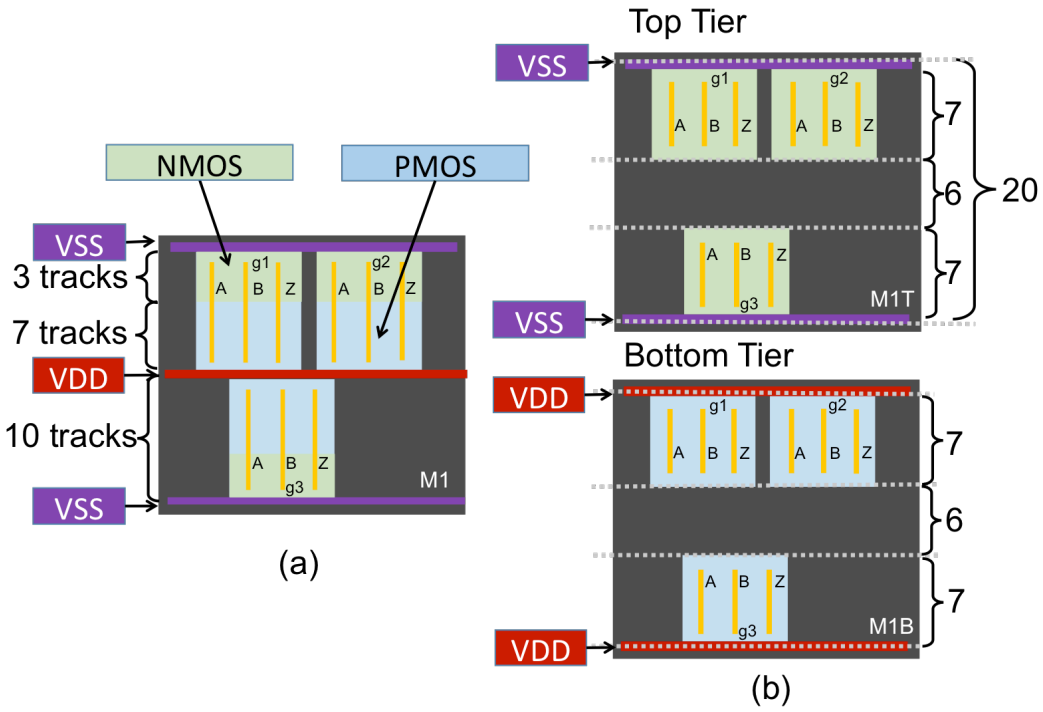


Figure 5.3: Example transforming 2D to 3DM standard rows.

5.1 Transforming 2D Standard Rows to 3DM Standard Rows

Figure 5.3 shows the transformation from 2D to 3DM standard rows using an example. In Figure 5.3-(a) two standard rows are shown with a height of 10 M1 pitches per row which is the case for the ISPD'15 benchmark suite Bustany et al. (2015). The rows are placed back-to-back so a pair of rows either share VDD (as shown in Figure 5.3-(a)) or share VSS. For each 2D cell, the area occupied by NMOS and PMOS transistors are shown. For a standard CMOS layout, the PMOS transistors will be placed closer to VDD and NMOS transistors are placed closer to VSS.

Due to the difference in carrier mobility, the PMOS to NMOS channel lengths have a ratio >1 and for example a ratio of 7:3 in Figure 5.3-(a). Therefore if we assume a standard cell height of 10 pitches, the PMOS and NMOS take 7 and 3 tracks respectively as shown in Figure 5.3-(a). We use the same folding technique described in Arabi et al. (2015); Lee et al. (2013) with the feature of having redundant pins inside the 3DM cells.

As shown in Figure 5.3-(b), the '2D' standard rows will be transformed into 3D monolithic (3DM) standard cells folded over two tiers, i.e., NMOS on the top tier and PMOS on the bottom tier. Here two rows of NMOS are enclosed by VSS lines running in parallel, and similarly two rows of PMOS are enclosed by VDD lines after the transformation. To ensure creating standard 3DM rows, i.e., with the same height, we keep the height of the PMOS and NMOS portions on the two tiers equal to each other which are 7 tracks in this example, as shown in Figure 5.3-(b). This results in unused area in the NMOS portion. Note the width of each 3DM cell remains the same as its 2D counterpart.

From a routing perspective, instead of metal layer M1 in 2D, two layers of M1T (top) and M1B (bottom) are added.

The two M1T and M1B layers connect with monolithic inter-*tier* vias (denoted by MIVs), which are placed only inside each 3DM cell. Specifically, for each pin on M1 in the 2D case, we create two pins in the 3DM cell, one located on M1T and the other located on M1B which connect to each other by MIVs as also shown in Figure 5.1. These *redundant* pins allow improving pin access because they are connected to each other within the standard cell and a route can connect to either one of them on either tier. For example in Figure 5.3-(b), notice for cell g1 that each of pins A, B, and Z are paired with a redundant one on the other tier. In the figure the pins are drawn to have equal height for simplicity. Moreover, we assume the transformation preserves the placement of

each cell. Given the same width for each cell but shorter height, the transformation results in additional tracks running between the two 3DM rows (per tier) which is 6 ($= 20 - 7 - 7$) free tracks in the example shown in Figure 5.3-(b). This assumes the shared VDD line in the center is ignored but it should be added as an additional track otherwise so in that case there will be 7 free tracks in the center. Overall, the above transformation results in the same layout area. However, it is also possible to trade off (reduce) the number of these free tracks for smaller layout area as we show in our experiments.

5.2 Cell-to-Cell Intra-Row Track Routing

After the 2D to 3DM standrd row transformation, we try to route as many nets which connect adjacent cells within the same 3DM row on either M1B or M1T. Note this is intra-row routing and does not use the free tracks in the middle of two standard rows. Figure 5.4 shows an example of two 2-pin nets. One net connects g1.Z to g2.A, and the other one connects g3.Z to g4.A. In the case when the two pins of a net that need to connect are in adjacent standard cells, we can connect them directly in either M1B or M1T (because of redundant pins), assuming there exists a common and free track which connects the two pins to each other.

Specifically, in our framework, after the 3DM cell transformation, we visit all the nets. For each visited 2-pin net, we then check if it can be routed on the same row between two neighboring cells. As we show in our experiments, in practice only a very small fraction of nets can be routed using this technique so next we apply inter-row track routing.

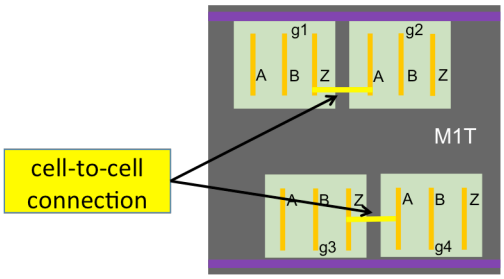


Figure 5.4: Cell-to-cell intra-row track routing on M1T.

5.3 Inter-Row Track Routing

In this section we present an ILP-based track routing technique which maximizes the number of nets that *may* be routed on the free tracks running between a pair of standard 3DM rows on either M1T or M1B. It also takes advantage of the redundant pins in 3DM cells.
The ILP is written for a double-row 'panels' with each panel containing two standard rows with free tracks in between, as shown in Figure 5.3-(b). One panel is located in the top tier and its corresponding panel is located in the bottom pier. The ILP considers routing all the nets with pins completely inside the double-row panels (which were *not* routed using the intra-row cell-to-cell technique.)
Recall during the transformation stage, each pin of a 2D cell translates into a pair of redundant pins located on M1B and M1T which are internally connected by MIVs. In order to simplify the discussions in this section, we only refer to the pins inside one of the double-row panels but the discussions hold for the corresponding pins in the other panel as well.
If a net has all its pins within a double-row panel, each pin belongs to either the higher or the lower standard row. We denote all the pins of net i belonging to the higher row as its north pins and all the pins of net i belonging to the lower row as the south pins of net i, as shown in

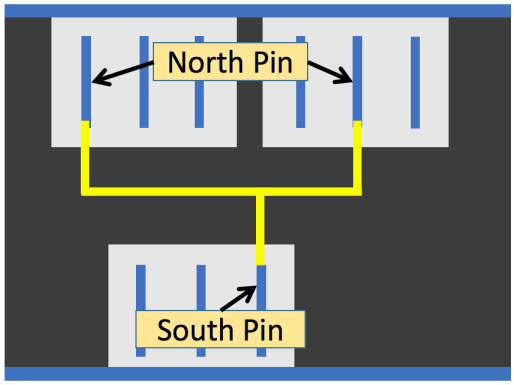


Figure 5.5: The single-trunk routing structure used for inter-row track routing with branches located on the same tier.

Figure 5.5.
When considering routing a net on M1B or M1T, the ILP assumes candidate routes with the simple structure made of a single trunk spanning the interval defined by the pins of the net and a set of branches connecting the pins to the trunk. The trunks may be located on either M1B or M1T and should be one of the available free tracks in the middle of the double-row panel. The branches are also located on M1B or M1T so they directly connect to the trunk (without need for vias). This is because M1B and M1T can support both horizontal and vertical routing. Figure 5.5 shows one candidate route of a net on one tier and there is another one identical to this on the other tier for the same net.
Finally, depending on the locations of north and south pins, we can sometimes make assertion that the candidate routes of two nets running on the same tier will definitely create a conflict (overlap) if selected simultaneously. Therefore the two nets may never be routed on the same tier. These conflicts are discussed in detail in Chapter 5.3, and are accounted for by the ILP.

Integer Linear Programming Formulation

Given a double-row panel P, all the nets whose pins are falling in these panels are first denoted by the set N_P.
For each net $i \in N_P$ we define three binary variables x_B^i, x_T^i and x_A^i expressing whether net i can be routed in M1B, M1T or above (referring to routing on M2 or higher layers). The ILP is formulated to maximize the number of nets which may be routed on either M1T or M1B and is given below:

$$\max_{x_B, x_T, x_A} \sum_{i \in N_P} x_B^i + x_T^i$$

s.t.

$$x_B^i + x_T^i + x_A^i = 1 \qquad \forall i \in N_P \tag{5.1}$$

$$x_B^i + x_B^j < 1, x_T^i + x_T^j < 1 \qquad \forall \text{conflicted } i, j \in N_P \tag{5.2}$$

$$x_B^i, x_T^i, x_A^i \in \{0, 1\} \qquad \forall i \in N_P \tag{5.3}$$

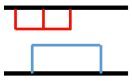
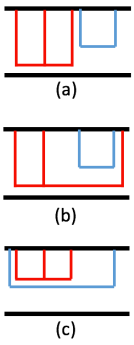
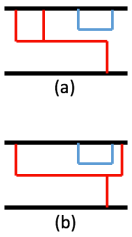
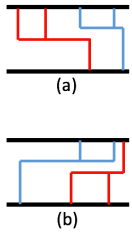
The first set of constraints ensures that for each net i only one route option is selected which indicates the net may be routed on M1B or M1T or above layers.
The second set of constraints checks each pair of nets i and j which have a routing conflict and ensures they cannot get simultaneously routed on M1B or on M1T. We discuss checking for conflict for a pair of nets in Chapter 5.3.
Once the integer program is solved, all the nets with $x_B^i = 1$ may be routed in M1B and all the nets with $x_T^i = 1$ may be routed in M1T. Those nets with $x_A^i = 1$ will have to be routed in M2 or above using commercial router, since routing them in M1B or M1T will definitely cause conflict.
Please note that the objective is to maximize the number of conflict-free nets which *may* be routed on M1B or on M1T. (Alternatively it can be

set to minimize the number of nets that are routed on M2 or above, i.e., sum of the x_A variables). In fact it is possible that the ILP generates a solution in which the number of nets that may be routed on a double-row panel exceeds the capacity of the panel. By capacity we mean the number of tracks located in the middle of the double-row panel, e.g., 7 tracks in the example of Figure 5.3-(b). Therefore, after ILP, we apply a final step to determine the nets that *can* be routed up to the panel capacity per tier. We discuss this procedure in Chapter 5.3.

Conflict Checking for A Pair of Nets

For two nets a and b that belong to the same double-row panel P, existence of a conflict means they cannot be routed simultaneously on M1T or M1B because their routes will overlap at some point. This is assuming the (single-trunk) routing structure as shown in Figure 5.5.
Table VIII is a look-up table (LUT) that we use to check if two nets a and b *do not* have a conflict based on their sets $\mathcal{N}$ and $\mathcal{S}$. In case of a conflict, a pair of constraints are created for them in the ILP as we discussed before.
Based on the north and south pins of the two net, we first look up columns 2 to 5 to identify one of the rows (9 cases). For each case, the conflict-free condition is listed in column 6 and a graphical example of that is listed in column 7. Note when multiple conditions are listed per case, only one of them needs to be satisfied. For example consider case 3 which is identified when all pins of nets a and b are north pins. Here one of the 3 listed conditions must be satisfied which are shown in the figure to ensure they are conflict-free. If none are satisfied, then the two nets have a conflict.

Table VIII: Look-up table of non-conflict conditions between nets, assuming all the elements in the set $\mathcal{N}$ or $\mathcal{S}$ are sorted in ascending order.

Case	Net a		Net b		Conditions	Examples
	$\mathcal{N}(a)$	$\mathcal{S}(a)$	$\mathcal{N}(b)$	$\mathcal{S}(b)$		
1	$n_1^a, \ldots, n_{N_a}^a$	$\emptyset$	$\emptyset$	$s_1^b, \ldots, s_{S_b}^b$	• No conflict between n_a and n_b in any case	
2	$\emptyset$	$s_1^a, \ldots, s_{S_a}^a$	$n_1^b, \ldots, n_{N_b}^b$	$\emptyset$		
3	$n_1^a, \ldots, n_{N_a}^a$	$\emptyset$	$n_1^b, \ldots, n_{N_b}^b$	$\emptyset$	a) $\mathcal{N}(a)$ and $\mathcal{N}(b)$ do no overlap: $n_{N_a}^a < n_1^b$ OR $n_{N_b}^b < n_1^a$ b) all north pins of net a are between two neighboring pins in $\mathcal{N}(b)$: $\exists k$ s.t. $n_k^b < n_1^a$ AND $n_{N_a}^a < n_{k+1}^b$ c) all north pins of net b are between two neighboring pins in $\mathcal{N}(a)$: $\exists k$ s.t. $n_k^a < n_1^b$ AND $n_{N_b}^b < n_{k+1}^a$	
4	$\emptyset$	$s_1^a, \ldots, s_{S_a}^a$	$\emptyset$	$s_1^b, \ldots, s_{S_b}^b$	Similar to case 3.	
5	$n_1^a, \ldots, n_{N_a}^a$	$\emptyset$	$n_1^b, \ldots, n_{N_b}^b$	$s_1^b, \ldots, s_{S_b}^b$	a) $\mathcal{N}(a)$ and $\mathcal{N}(b)$ do not overlap: $n_{N_a}^a < n_1^b$ OR $n_{N_b}^b < n_1^a$ b) all north pins of net a are between two neighboring pins in $\mathcal{N}(b)$: $\exists k$ s.t. $n_k^b < n_1^a$ AND $n_{N_a}^a < n_{k+1}^b$	
6	$\emptyset$	$s_1^a, \ldots, s_{S_a}^a$	$n_1^b, \ldots, n_{N_b}^b$	$s_1^b, \ldots, s_{S_b}^b$	Similar to case 5.	
7	$n_1^a, \ldots, n_{N_a}^a$	$s_1^a, \ldots, s_{S_a}^a$	$\emptyset$	$s_1^b, \ldots, s_{S_b}^b$	Similar to case 5.	
8	$n_1^a, \ldots, n_{N_a}^a$	$s_1^a, \ldots, s_{S_a}^a$	$n_1^b, \ldots, n_{N_b}^b$	$\emptyset$	Similar to case 5.	
9	$n_1^a, \ldots, n_{N_a}^a$	$s_1^a, \ldots, s_{S_a}^a$	$n_1^b, \ldots, n_{N_b}^b$	$s_1^b, \ldots, s_{S_b}^b$	a) $\mathcal{N}(a)$ is to the left of $\mathcal{N}(b)$ AND $\mathcal{S}(a)$ is to the left of $\mathcal{S}(b)$: $n_{N_a}^a < n_1^b$ AND $s_{S_a}^a < s_1^b$ b) $\mathcal{N}(a)$ is to the right of $\mathcal{N}(b)$ AND $\mathcal{S}(a)$ is to the right of $\mathcal{S}(b)$: $n_{N_b}^b < n_1^a$ AND $s_{S_b}^b < s_1^a$	

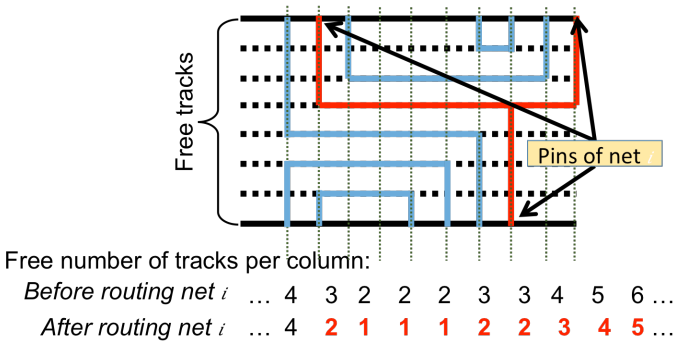


Figure 5.6: Available tracks per column before/after routing net i. Assume the blue nets have already been routed.

Finalizing the Inter-Track Routing Solution

Once the lists of conflict-free nets that *may* be routed on M1B and M1T are found from the ILP solution, we try to route as many as possible up to the panel's capacity. Our procedure works on each tier independently. For example for M1T, all nets with $x_T = 1$ are first identified for the top tier. Next, these nets are sorted in ascending order, by the length of the interval defined by their left-most and right-most pins.
Next we attempt to route each net in the sorted order using the simple structure (one trunk connected by branches), if there exists free tracks in the middle of the panel. Specifically, we keep track of the number of free tracks per 'column' (corresponding to pin coordinates) as each net gets routed. Initially all the tracks are available per column. Once a net gets routed, the number of tracks in each column covered in the net's range is deducted by 1. Figure 5.6 illustrates.

5.4 Experimental Results

We implemented our design flow in C++ on the 45nm ISPD'15 Bustany et al. (2015) benchmarks. All simulations ran on a Linux machine with a 2.8GHz Intel CPU and 12GB memory.

To apply the transformation from 2D to 3DM standard rows, we assumed a 2D standard row height of 10 tracks which resulted in a 3DM row height with 7 tracks and 6 free tracks in between a pair of 3DM rows, as shown in Figure 5.3.

Next our framework routed as many nets as possible on M1T and M1B. Specifically it first identified and routed all nets in adjacent cells which were eligible for cell-to-cell intra-row track routing. Then, for each double-row panel, we used our ILP formulation to maximize the number of nets which *may* be routed using inter-row tracking using and selected a group of eligible nets. After the ILP, we used the technique given in Chapter 5.3 to route as many as possible from the selected nets on the free tracks in the middle of each double-row panel. We used CPLEX 12.6 IBM to solve the ILPs. We then removed the nets which were routed on M1B and M1T from the benchmarks and used the Olympus SoC global and track router Graphics for the remaining nets. The detailed routing solution generated by Olympus was finally evaluated by running the '`check_drc`' command.

In our experiments we make comparison between the above 3DM-based implementation and a standard implementation using 2D cells. Specifically in the 2D case, we started with the same placed design and only ran it through the Olympus SoC global and detailed router and then evaluated the generated solution for detailed routability.

Next we discuss our simulation results at different stages of our framework (i.e., M1B/M1T routing as well as M2 and above routing). We also do an experiment in which we vary the number of free tracks in the middle of a double-row panel to show the existing trade offs between

area reduction and improvement in routability.

Routing on M1B/M1T

Table IX shows the results of M1B/M1T routing. Columns 3 and 4 show the numbers and percentages of nets routed at the cell-to-cell intra-row routing stage. As can be seen only a small percentage (on-average 1.39%) of nets were routed at this step. This shows intra-row routing in practice is quite limited for making improvement.
Columns 5 and 6 report the numbers and percentages of nets selected by ILP. The nets which are eventually routed in M1B/M1T using the ILP solution are reported in columns 7 and 8. As can be seen, on-average 27.76% were routed on M1B/M1T across the benchmarks which is a relatively high percentage and we show results in significant improvement in detailed routability.
The last three columns (9 to 11) report information about ILPs, which includes the number of ILP problems (i.e. the number of double-row panels), the average number of constraints per ILP and the total runtime to solve *all* the ILPs. As can be seen on-average about 194 ILPs are solved in 5.3 seconds and each include on-average 617 constraints.

Routing on M2 and Above

After routing nets in M1B/M1T, we performed global routing and track routing on the remaining nets. These nets are routed on M2 or above (i.e. M2 to M5 in the ISPD 2015 benchmarks). We used the Olympus commands '`route_global`' and '`route_track`' to perform global and track routing. Table X shows the comparison of detailed routing results between the 2D and 3DM cases. Recall in the 2D case, the original benchmarks containing all the nets are global and detailed routed.

Table IX: Results of M1B/M1T routing

Design	# nets	cell-to-cell intra-row		ILP-based inter-row				ILP information		
		routed		selected		routed		# ILPs(panels)	Ave. # constraints	Runtime (seconds)
des_perf_1	112877	1534	1.4%	37450	33.2%	37289	33.0%	112	1655.6	10.2
des_perf_a	110280	1206	1.1%	28521	25.9%	28506	25.8%	226	560.2	6.8
des_perf_b	112877	1445	1.3%	30126	26.7%	30113	26.7%	151	724.1	7.0
edit_dist_a	131133	1513	1.1%	28537	21.8%	28274	21.6%	200	611.5	6.0
fft_1	33306	237	0.7%	11008	33.1%	10880	32.7%	67	698.0	2.0
fft_2	33306	313	0.9%	10595	31.8%	10594	31.8%	86	381.0	1.6
fft_a	32087	206	0.6%	7182	22.4%	7021	21.9%	200	93.5	1.3
fft_b	32087	225	0.7%	7606	23.7%	7482	23.3%	200	108.3	1.0
matrix_mult_1	158526	1921	1.2%	51612	32.6%	51203	32.3%	137	1793.3	12.0
matrix_mult_a	154283	2217	1.4%	44784	29.0%	44761	29.0%	376	418.4	8.2
matrix_mult_b	151611	2177	1.4%	35078	23.1%	35068	23.1%	376	254.1	5.8
pci_bridge32_b	29416	1407	4.8%	8840	30.1%	8833	30.0%	200	103.7	1.5
AVERAGE	90982.4	1200.0	1.39%	25111.5	27.76%	25002.0	27.60%	194.2	616.8	5.3

We ran the '`check_drc`' command on the detailed routing DB in Olympus SoC which reported the total number of violated segments and total number of DRC errors. The total number of violated segments included errors on shorts, spacing, port shorts, cross shorts, twist shorts, diffnet shorts, diffnet spacings, samenet spacings, cut projection, cut spacing, end of line, min-area bottom, min-area, loop over port, minstep, and vias overlap.
As can be seen from the table the total number of violated segments and DRC errors were reduced on-average by 29.8% and 20.7% in the 3DM case, compared to the 2D case.

Table X: Results of routing on M2 and above

Design	# violated segments		# DRC errors	
	2D	3DM	2D	3DM
des_perf_1	2768	75.4%	1987	65.2%
des_perf_a	7062	30.1%	5262	22.2%
des_perf_b	307	61.6%	550	50.4%
edit_dist_a	83671	46.8%	33581	40.4%
fft_1	205	25.4%	388	-11.3%
fft_2	2584	13.1%	2577	9.8%
fft_a	1474	11.2%	1418	11.6%
fft_b	3024	15.2%	4218	1.9%
matrix_mult_1	10906	46.4%	7984	25.1%
matrix_mult_a	5141	18.0%	3348	8.7%
matrix_mult_b	9416	11.4%	7561	9.2%
pci_bridge32_b	730	2.9%	712	14.9%
AVERAGE	10607.3	5968.8 (29.8%)	5418.0	3896.6 (20.7%)

Impacts of Number of Free Tracks on the M1B/M1T Routing

Recall the transformation process of 2D to 3DM standard row results in free tracks in the middle of a double-row panel as shown in Figure 5.3. In this experiment we reduce the number of free tracks which translates into direct area saving in terms of reducing the height of the layout. We show the tradeoff of this reduction with fewer number of nets that were routed by our framework on M1B and M1T. Note that this step only impacts the inter-row track routing step of our framework.
Table XI reports the result when the number of free tracks vary from 6 (original) to 4, 2, and 1 tracks in the middle. As can be seen, using 6 free tracks our framework routed 29.00% of all the nets (which can also be obtained by adding the nets routed at the intra-row and inter-row stages of our framework in Table IX). Using 4 and 2 free tracks our framework routed 28.69% and 24.69%, respectively. With just 1 free track, we were still able to route 17.01% of the nets on M1B/M1T. We note 1 free track translates into area saving of 5 ($= 6-1$) tracks per panel for per each pair of standard rows. This is a significant reduction in the height of the layout. These results are fundamentally because many nets connect pins which are placed close to each other and located on adjacent standard rows.

Table XI: Impact of varying number of free tracks in the middle on the number of nets routed in M1B and M1T

Design	# free tracks			
	6	4	2	1
des_perf_1	34.4%	33.4%	25.8%	16.4%
des_perf_a	26.9%	26.6%	22.0%	14.7%
des_perf_b	28.0%	27.6%	23.1%	15.6%
edit_dist_a	22.7%	22.4%	19.5%	14.0%
fft_1	33.4%	33.0%	26.7%	17.1%
fft_2	32.7%	32.6%	28.6%	19.3%
fft_a	22.5%	22.5%	21.1%	15.8%
fft_b	24.0%	24.0%	22.4%	16.5%
matrix_mult_1	33.5%	32.9%	26.0%	16.5%
matrix_mult_a	30.4%	30.3%	26.7%	18.2%
matrix_mult_b	24.6%	24.5%	22.3%	16.0%
pci_bridge32_b	34.8%	34.7%	32.1%	23.9%
AVERAGE	29.00%	28.69%	24.69%	17.01%

6 IMPROVING THE DISTRIBUTION OF CONGESTION IN GLOBAL ROUTING

To improve routability, typically global routing procedures minimize a total edge-overflow metric (denoted by EOF) Chen et al. (2009); Cho et al. (2009); Xu and Chu (2011); Chang et al. (2010); Pan and Chu (2007); Zhang et al. (2008); Xu et al. (2009); Pan et al. (2012). Total overflow is sum of edge overflows in the global routing (GR) grid-graph. However, minimizing EOF may no longer be sufficient for routability.
Recently new metrics are proposed to evaluate routability based on edge congestion ratio, which is the ratio of edge utilization to edge capacity. The authors in Wei et al. (2012) propose two rank-based metrics denoted by $ACE(x)$ and $WCI(y)$, where $ACE(x)$ measures the average congestion ratio of the top x% most-congested edges, and $WCI(y)$ measures the percentage of nets with congestion ratio higher than y%. A fast global router is then proposed in Liu et al. (2013a) with the main purpose to quickly build a congestion map to interact with a routability-driven placer (and not to create an improved global routing solution). However the impact on the traditional EOF metric is unclear and not shown in the experiments.
In this work we propose a procedure to shape the congestion ratio distribution of a GR solution while guaranteeing that EOF is not increased. Our framework at high-level is based on defining non-overlapping 3D routing subregions covering about 25% of the layout. Our procedure is then applied to the subregions independently without perturbing the routing outside the subregions. This allows independent and parallel processing of all the subregions. Shaping of the distribution is done in a controlled manner, using a convex penalty function; the user is allowed to associate a higher penalty to minimize the edges which will fall within undesirable ranges of congestion. In our experiments we

show that a significant improvement can be made to the distribution of the congested edges in an already-optimized global routing solution on realistic modern design instances while guaranteeing that the total overflow is not increased.

6.1 Problem Statement

Before discussing our techniques we give a review of basic notations in this work and provide a problem statement. We are given a grid-graph $G = (V, E)$ for global routing of dimension $V = X \times Y \times L$, where L represents the number of metal layers. We are also given a set of nets $\mathcal{N}$ to route. Each net is identified by some terminals which are a subset of the vertices in V. In modern designs, some of these terminals may be *virtual* indicating that they may be located at the higher metal layers. Each edge $e \in E$ has a constant capacity c_e and is associated with an individual (sum of) wire size and spacing denoted by s_e. It may also have a constant blockage amount of b_e for example representing routing resources taken by pre-routed power delivery or clock networks. Intuitively, edge e represents the boundary of two adjacent global cells and its capacity represents the available routing resources on the boundary when ignoring its routing blockage.

We are also given an initial solution where for each net $i \in \mathcal{N}$, we have a Steiner tree $\hat{t}_i$, a subset of the edges in E that connects the terminals of net i. Each edge $e \in E$ has a utilization $\hat{u}_e$, which is the amount of routing resource consumed from edge e by the initial solution. The overflow of an edge in the initial solution is given by $\max(\hat{u}_e - (c_e - b_e), 0)$ and the total overflow denoted by $\hat{TOF}$ is the sum of all the edge overflows. We wish to improve the distribution of this initial solution while ensuring the total overflow of the global routing solution does not exceed its current value $\hat{TOF}$.

We also define a congestion ratio r_e for each edge e which is given by $\hat{r}_e = \frac{\hat{u}_e + b_e}{c_e}$. A congestion ratio less than or equal to 1 corresponds to an edge with 0 overflow. Our routing framework models all the factors captured in the ISPD'11 benchmarks Viswanathan et al. (2011).

6.2 Global Routing Framework

Our router works on non-overlapping routing subregions which are identified around the congested edges in the input solution. In this section, we first describe how our routing subregions are created. We then present a mathematical formulation to be solved inside each subregion followed by our techniques to improve the runtime efficiency of our routing framework.

Forming the Routing Subregions

We start by visiting the edges in decreasing order of their initial congestion ratios ($\hat{r}_e$ for edge e). When visiting an edge, we first create a 3D subregion of fixed size centered around the edge. The shape of the subregion is a hyper-cube which extends over all the metal layers. (In our experiments we use subregion size of 20x20x9 which we found appropriate based on typical global grid dimension and number of metal layers for ISPD'11 benchmarks Viswanathan et al. (2011).)
If the subregion of an edge overlaps with a previous subregion, then the subregion will be dropped so that subregions will be non-overlapping. The process stops as soon as the sum of the volume of the subregions corresponding to the edges visited so far becomes at least 25% of the overall routing space. The remaining edges (which will have a lower congestion ratio) won't be visited.
For each routing subregion $\mathcal{R} = (V_R, E_R)$, we then identify a subset of the nets that are fully or partially routed inside $\mathcal{R}$. If a route crosses the

boundary of $\mathcal{R}$, the crossing point will be added as a new terminal of the corresponding net. If a net terminal falls outside $\mathcal{R}$, it will be eliminated from the set of the terminals of the net. In the end, for subregion $\mathcal{R}$, the new set of nets to be routed is denoted by $\mathcal{N}_{\mathcal{R}}$ and includes those nets that have one or more terminals inside $\mathcal{R}$, and/or have a "pseudo-terminal" on the boundary.
The above procedure creates independent subregions that will be processed in parallel. The routing of each net will be completely inside the subregion and will more effectively utilize the 3D-routing resources. Note that the above procedure ensures the routing solution outside the re-routed subregions remains intact which is useful to ensure minimal change to the already-optimized initial solution.

Mixed-Integer Linear Programming Formulation

Here we present a mathematical formulation to precisely describe the routing problem that we propose to be solved inside each subregion. We first describe the objective optimized by the formulation, and then describe the formulation as a mixed-integer linear program.
The objective of our formulation is to minimize a convex, piecewise-linear penalty function of the edge congestion ratios $r_e = \frac{u_e + b_e}{c_e}$ with u_e, c_e and b_e representing the utilization, capacity and blockage of edge e respectively. The piecewise-linear function is given as a set of slope and y-intercept pairs $J_e = \{(m_{je}, d_{je}) \mid j = 1, 2 \ldots, |J_e|\}$, so that the penalty for edge $e \in E$ is $p_e = m_{je} r_e + d_{je}$ for some $j \in J_e$.
Figure 6.1-(a) depicts a typical example of the penalty function. The x-axis is the congestion ratio r_e, and the y-axis is the penalty p_e. The intervals on the x-axis represent thresholds to specify ranges of interest for congestion ratios which are denoted by (TCRs). Figure 6.1-(a) shows a realization of the penalty function with three line segments. Assume $TCR_1 = 1$ and $TCR_2 = 1.1$. The edges with $r_e \leqslant 1$ do not have overflow,

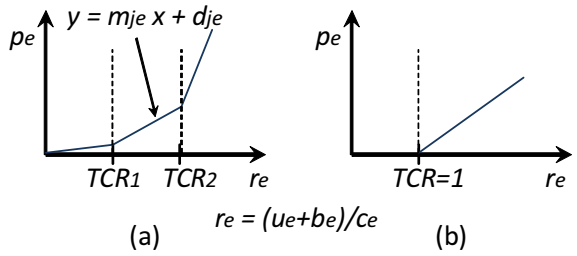


Figure 6.1: (a) Mapping edge congestion ratio to edge penalty based on a convex piece-wise linear function; (b) Variation when edge penalty is its overflow.

therefore the associated line segment has a very small slope representing a very small penalty. For the edges with congestion ratio between 1 and 1.1, the line segment has a higher slope indicating a higher penalty. For the remaining range of $r_e > 1.1$, the penalty is incurred at a much higher rate. The objective is to minimize the sum of these edge penalties, which we denote by *PCR* (for Penalized Congestion Ratios).
The piecewise-linear penalty function *PCR* is extremely flexible. For example as shown in Figure 6.1-(b), by creating only the two slope-intercept pairs $J_e = \{(0,0),(1,-1)\}\,\forall e \in E$, the *PCR* metric becomes equivalent to the total edge-overflow metric EOF (when there is no blockage and if all edge capacities are equal).
Given a routing region $\mathcal{R} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ and nets $\mathcal{N}_{\mathcal{R}}$, we propose a mixed-integer linear programming (MILP) formulation that identifies nets that will improve the PCR metric. We first give the following definitions.

- $\mathcal{T}_i$: set of candidate routes for net $i \in \mathcal{N}_{\mathcal{R}}$.
- Parameter $a_{te} = 1$ if route $t \in \mathcal{T}_i$ contains edge $e \in E_{\mathcal{R}}$.
- Piecewise linear functions given as a set of slope-intercept pairs $J_e = \{(m_{je}, d_{je}) \mid j = 1, 2 \ldots |J_e|\}\,\forall e \in E_{\mathcal{R}}$.

- Binary variable x_{it} set to 1 if route $t \in \mathcal{T}_i$ is selected for net i and 0 otherwise.
- Real variables $u_e \geqslant 0$, $o_e \geqslant 0$. and $p_e \geqslant 0$ representing the utilization, overflow, and penalty for each edge $e \in E_{\mathcal{R}}$.

Also recall that parameters c_e, b_e, and s_e represent the capacity, blockage, and sum of wire size and spacing corresponding to edge e as defined in Chapter 6.1. With the above definitions, the MILP is expressed as follows:

$$\min_{x,p,u,r,o} PCR = \sum_{e \in E_{\mathcal{R}}} p_e$$

$$\sum_{t \in \mathcal{T}_i} x_{it} = 1 \quad \forall i \in \mathcal{N}_{\mathcal{R}} \tag{6.1}$$

$$s_e \sum_{i \in \mathcal{N}} \sum_{t \in \mathcal{T}_i} a_{te} x_{it} = u_e \quad \forall e \in E_{\mathcal{R}} \tag{6.2}$$

$$\frac{u_e + b_e}{c_e} = r_e \quad \forall e \in E_{\mathcal{R}} \tag{6.3}$$

$$p_e \geqslant m_{je} r_e + d_{je} \quad \forall e \in E_{\mathcal{R}} \; \forall j \in J_e \tag{6.4}$$

$$u_e - c_e + b_e \leqslant o_e \quad \forall e \in E_{\mathcal{R}} \tag{6.5}$$

$$\sum_{\forall e \in E_{\mathcal{R}}} o_e \leqslant \hat{TOF} \tag{6.6}$$

In this formulation, the first set of equations states that for each net in $\mathcal{R}$, only one route should be selected from the set of its routing trees. The second set of equations computes the utilization for each edge in region $\mathcal{R}$. (Here if a route t includes edge e, it contributes s_e units to the utilization of the edge where s_e is a constant parameter representing the sum of wire size and spacing on the layer corresponding to edge e.) The third set of equations computes the congestion ratio r_e for edge e using its utilization u_e and the constant parameters c_e and b_e corresponding to the edge capacity and blockage. The fourth set of inequalities

computes the edge penalties p_e using the specified parameters m_{je} and d_{je}. The objective PCR minimizes the sum of the edge penalties. Finally, the fifth and sixth set of inequalities compute the edge overflow o_e and bound the sum of all edge overflows to be at most the total overflow of the initial solution which was denoted by $\hat{TOF}$ (in Chapter 6.1).
The presented formulation selects a new route for each net from all its possible routing trees inside the subregion such that in the resulting routing solution, the sum of the penalized congestion ratios (*PCR*) is minimized. The objective of the formulation is designed to "shape" the distribution of the congestion away from highly-congested edges.
Our global routing procedure solves the above MILP formulation as a sequence of linear programs (LPs) in which only a subset of "critical" net variables are considered. This allows to develop a fast procedure. Furthermore, each LP updates the candidate routes defined by the set $\mathcal{N}_R$ by *adding* new candidate routes to it. We solve a sequence of 20 LPs to create a final set of candidate routes for each net. Finally, the route of a net is found by solving the MILP (mixed-integer) formulation, by setting $\mathcal{N}_R$ to be the set of candidate routes obtained from the last LP.

Techniques for Improving Runtime Efficiency

Here we present our techniques to improve the runtime efficiency of our routing framework. After trying various strategies, we follow the following procedure to consider a small number of nets for each LP in order to improve the runtime spent on solving it.

1. For subregion $\mathcal{R}$, sort all the edges $e \in \mathcal{R}$ in descending order of their congestion ratio. (The congestion ratio r_e is directly taken from the initial solution in the first LP, and from the corresponding variable assignment from the previous LP.)

2. For a visited edge e, select net $i \in \mathcal{N}_{\mathcal{R}}$ if its largest candidate route variable (i.e., highest x_{it} value from the previous LP solution) includes edge e. For the first iteration, use the route provided by the initial solution.

3. Go to the next edge and repeat step 2. Stop as soon as a fixed number of nets (=200) are selected. We determined this value to be suitable in our experiments for all the ISPD'11 benchmarks and our discussed subregion size.

Note, each LP includes all the constraints given in the MILP formulation so all edges in the subregions are included in it. All binary variables are relaxed to be in the interval [0,1]. Overall, the above procedure ensures that a small number of nets are selected in each subregion. The nets are deemed critical because the edges used to identify them have higher congestion as determined by the most-recent LP solution. The LP-solver can in turn more effectively reduce the *PCR* objective in consecutive LPs because congested edges have a higher degree of contribution to the *PCR*.

Once the set of critical nets are selected, each LP adds *at most* two candidate routes per critical net.

We first explain how the first candidate route is found for a critical net. This is done by applying decomposition for a multi-terminal net to break it into two-terminal subnets which is done similar to prior works such as Shojaei et al. (2011). We then use the shortest path (i.e., min-cost path in a weighted graph) to connect each sub-net.

Specifically, we start by assigning a weight w_e for each edge $e \in \mathcal{R}$ in the considered (3D) routing subregion. The weight of each edge is determined based on its utilization which is computed from the most-recent LP solution (or the initial routing solution in the first LP). We then apply a transformation to simplify this 3D graph to a weighted 2D graph

in order to reduce its size and accelerate the runtime of the min-cost path problem.
Specifically, for the 3D grid-graph G of size $X \times Y \times L$, the 2D grid-graph G_{2D} will be of size $X \times Y$. To determine the weight of an edge in the G_{2D}, for each edge $e_{2D} = ((i_1, j_1), (i_2, j_2))$ we consider all edges in G specified by $e = ((i_1, j_1, \ell), (i_2, j_2, \ell)), \forall \ell \in L$. The weight of e_{2D} denoted by w_{2D} is given by $w_{e_{2D}} := \min_\ell(w_e)$. A net terminal located at (i,j,ℓ) in G will also map to (i,j) in G_{2D}. For each edge e_{2D} in G_{2D} we also remember the layer ℓ of the corresponding 3D edge in G which had the minimum weight.
Figure 6.2 shows an example on how this transformation is done together with the layer remembered for each 2D edge.
Once a shortest path is found in G_{2D}, a 3D path is immediately created by assigning each edge to its corresponding layer and then connecting the edges (which have a shared vertex in G_{2D}) with vias.

Theorem 6.1. *Our transformation guarantees that the shortest-weight path in* G_{2D} *creates the shortest-weight path on* G.

Proof. Assume there exists a shortest path P in G which includes at least one edge $e = ((i_1, j_1, \ell), (i_2, j_2, \ell))$ in which w_e is not the smallest compared to $e' = ((i_1, j_1, \ell'), (i_2, j_2, \ell'))$ which is above or below it on a different layer. Replace e with e' in P and create a connected path by adding vias from the two end points of e'. The new path P' will have a smaller total weight than P because $0 < w_{e'} < w_e$ and the added vias have a weight of 0. This contradicts the initial assumption that P is the shortest path. □

We also consider adding a second candidate route for each selected net as follows. We first identify the "best" candidate route of net i which corresponds to tree t with the highest x_{it} value from the solution of the previous LP iteration. After updating the edge weights at the beginning

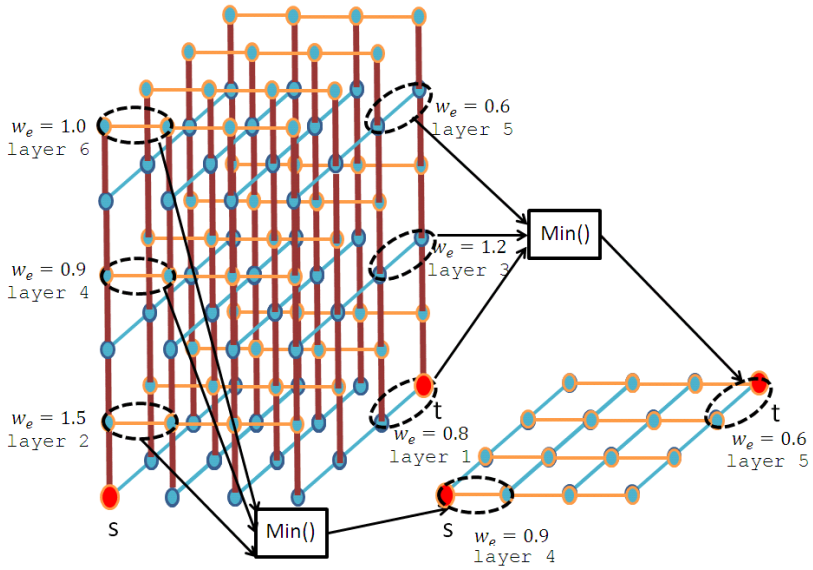


Figure 6.2: Creating a 2D grid graph for solving the 3D shortest path problem.

of the new iteration, we consider moving each flat segment of t to the other layers. If the sum of the edge weights for a segment decreases, then it will be moved. This approach founds a second candidate route for a net if there is at least one such segment that has a decreased cost. Determining if a second candidate route can be found for a net is done extremely fast and in a runtime smaller than solving the shortest path problem.

6.3 Experimental Results

We implemented our global routing procedure, denoted by IGR, in C++ and ran experiments on a 2.8GHz Intel CPU with 12GB of memory. Experiments were ran on the ISPD'11 routability-driven placement benchmark Viswanathan et al. (2011). Table XII lists the benchmark information including the global routing X and Y grid-dimension (with 9 metal

layers for all benchmarks), and number of nets. We used a placed version for each benchmark by downloading the best placement solution generated for that benchmark from the ISPD'11 contest website.
To create an initial solution for IGR, we used the NCTU-GR 2.0 Liu et al. (2013a) global router which ran in regular mode. As a result our initial global routing solutions were already optimized and had a total overflow of 0 in most benchmarks.
To implement IGR, 3D non-overlapping subregions were formed which covered about 25% of the layout. The global routing problem inside each subregion was defined to optimize the *PCR* objective according to the following penalty function.

$$p_e = \begin{cases} 0 & \text{if } 0 \leqslant r_e < 0.5 \\ 10r_e - 5 & \text{if } 0.5 \leqslant r_e < 0.7 \\ 1000r_e - 698 & \text{if } r_e \geqslant 0.7 \end{cases} \tag{6.7}$$

This function assigns a mild penalty to edges with congestion ratio of $0.5 \leqslant r_e < 0.7$. However it assigns a much higher (X10) penalty slope for congestion ratio of 0.7 or above. This makes all the edges with congestion ratio greater than 0.7 to have a higher contribution to the *PCR* objective, thus be more likely to be reduced by IGR. This includes edges with overflow (i.e., congestion ratio above 1) as well as edges with congestion ratio close to 1; as noted in Alpert and Tellez (2010), this latter group of edges can easily get over-capacity if rerouting becomes necessary at the detailed-routing stage when additional factors need to be considered which were ignored during global routing. All LPs and the MILPs of subregions were solved using CPLEX V12.6.0 IBM.

Table XII: Benchmark information and placement tools

Design	$G_X \times G_Y$	#Nets	Placer
s1	704x516	822744	SimPLR
s2	770x1114	990899	Ripple
s4	467x415	567607	Ripple
s5	774x713	786999	Ripple
s10	638x968	1085737	RADIANT
s12	444x518	1293436	SimPLR
s15	399x495	1080409	Ripple
s18	381x404	468918	mPL11

Comparison of the Distribution of the Congestion Ratios

Table XIII shows the impact of IGR on the distribution of edge congestion ratios. Columns 4-10 correspond to edges with different ranges of congestion ratio. Different rows show the distribution for the initial solution (expressed in terms of number of edges in each range) as well as of IGR (expressed in terms of % reduction in number of edges in each range compared to the initial). For example for s1, there were 327201 edges with congestion ratio in the range of (0.8,1] in the initial solution, and IGR reduced the *number* of these edges by 18.09%.
Within the (0.8,1] range, we were able to further reduce the number of edges within the (0.9,1] sub-range, on-average by 12.83% and at most by 23.80% over all the benchmark.
Columns 11 and 12 report the total wire-length (TWL) and total edge-overflow (EOF). We verified that our calculation of these quantities matched with the ones generated by the ISPD'11 contest evaluation script for each benchmark. All EOFs remained unchanged after applying IGR. TWL increased slightly, however TWL is not considered in our formulation and to control long detouring, we ensure the initial routes are decomposed on the subregion boundaries and rerouted only inside each subregion.

Table XIII: Comparison of the distribution of edge congestion ratio

Design	GR		Distribution of Edge Congestion Ratio										
			0	(0.0,0.2]	(0.2,0.4]	(0.4,0.6]	(0.6,0.8]	(0.8,1]	>1	TWL	EOF	#sub	T_{sub}
s1	INIT	#edges	529110	485072	380205	328254	312823	327201	0	14735	0		
	IGR	%reduction	1.85%	4.96%	4.84%	-18.21%	-16.53%	18.09%	0.00%	15253	0	227	39
s2	INIT	#edges	1122420	757910	655549	662891	777935	1216494	239	31385	1732		
	IGR	%reduction	0.61%	3.21%	4.47%	-10.56%	-15.04%	10.40%	17.57%	32181	1732	536	83
s4	INIT	#edges	129022	198833	196199	199062	214162	366314	52	10781	282		
	IGR	%reduction	2.65%	4.46%	2.22%	-11.48%	-12.76%	9.16%	25.00%	11009	282	121	43
s5	INIT	#edges	982709	691280	524226	422988	373759	520927	0	17549	0		
	IGR	%reduction	0.77%	4.19%	4.51%	-17.30%	-20.51%	17.20%	0.00%	18235	0	344	64
s10	INIT	#edges	779358	591978	533919	506525	533167	915525	0	25094	0		
	IGR	%reduction	1.51%	3.66%	1.18%	-13.38%	-12.04%	10.06%	0.00%	25697	0	385	113
s12	INIT	#edges	250428	148342	125669	148520	227682	794792	0	22450	0		
	IGR	%reduction	1.49%	5.83%	5.93%	-10.76%	-11.62%	2.85%	0.00%	22716	0	143	37
s15	INIT	#edges	45970	98203	144774	196549	289995	654558	0	17824	0		
	IGR	%reduction	5.45%	7.79%	6.70%	-8.19%	-11.25%	4.41%	0.00%	18087	0	123	41
s18	INIT	#edges	279609	129569	118442	124688	140766	279642	0	9561	0		
	IGR	%reduction	1.53%	6.47%	8.89%	-15.20%	-17.03%	7.05%	0.00%	9813	0	96	38
AVERAGE		%reduction	1.98%	5.07%	4.84%	-13.13%	-14.59%	9.90%	5.32%	-	-	-	-

The last row shows the average %reduction by IGR for each range over all the benchmarks. On average IGR was able to reduce the number of congested edges which fall in the range (0.8,1] by 9.9%. Only two benchmarks (s2 and s4) had edges with congestion ratio >1. These are few edges having overflow and IGR associates a higher penalty with them so IGR reduced their count by a higher percentage of 17.57% and 25.00% for s2 and s4 respectively. In return, IGR increased the edge count falling in the ranges with lower congestion ratio (0.4,0.8]. Column 13 reports the number of subregions created by IGR for each benchmark. Note these are independent and non-overlapping subregions of fixed size 20x20x9 so they can be parallel-processed. Column 14 reports the maximum runtime of a subregion in each benchmark in seconds. The average runtime is less than a minute and the maximum runtime of subregion over all benchmarks was 113 seconds. This runtime includes all steps of the global routing procedure for a subregion. We note, we consider IGR to be a fast procedure in the presence of parallel cores/machines which is common these days. So the wall runtime of our procedure will also be very fast and comparable to the the maximum runtime of a subregion. Through IGR we are able to modify the congestion ratio distribution of a given GR solution.

Comparison of Runtime

Table XIV reports information related to the runtime of IGR. The number of subregions created by IGR is reported in column 2 for each benchmark. Note these are independent and non-overlapping subregions of fixed size 20x20x9 which cover about 25% of the layout and can be parallel-processed. Column 3 reports the maximum runtime of a subregion in each benchmark in minutes. The average runtime is less than a minute and the maximum runtime of subregion over all benchmarks was 1.28 minutes. This per-subregion runtime includes all steps of the

global routing procedure for a subregion including solving iterations of column generation using linear programming and the final MILP. We note, in the presence of enough number of cores (which should not be an issue for EDA companies), the estimate of the wall clock time will be close to (slightly higher than) the maximum runtime of a subregion per benchmark because the subregions are completely independent of each other. The only additional overhead is to send information about each subregion (which is a small-sized problem relative to the entire layout) to each core and then collect back the solutions and assemble them together. The overall CPU time when all subregions are processed sequentially (which is based on our current implementation) is reported in minutes per benchmark and is on average 69.98 minutes.
Columns SBR, SHP, LP, ILP show the breakdown of the overall sequential runtime as a percentage. Specifically SBR represents the percentage of the time spent on creating the subregion, SHP represents the percentage spent on solving all the shortest path problems, LP represents the percentage spent on solving the linear programs which is done at each iteration of column generation, and ILP represents the percentage spent on one-time solving the ILP at each subregion. We note these percentages are computed by adding the corresponding times over all the subregions and then dividing each by the overall sequential runtime. As can be seen, most of the runtime is spent on solving LP and ILP using CPLEX. Interesting solving the shortest path problem to generate new candidate routes takes a small percentage of on-average only 0.08% over all the benchmarks which is due to our graph transformation procedure to quickly solve many instances of the 3D minimum-weight path problem using our proposed 2D variation.

Table XIV: Analysis of runtime and number of subregions in IGR

Design	Runtime (minutes)			Breakdown(%)			
	#regions	T_{region}^{max}	T_{seq}	SBR	SHP	LP	ILP
s1	227	0.72	64.47	5.8	0.09	63.7	30.4
s2	536	1.14	112.00	2.3	0.09	50.9	46.7
s4	121	0.48	32.27	1.8	0.09	51.0	47.1
s5	344	1.42	111.88	3.1	0.08	81.5	15.3
s10	385	1.28	125.78	11.4	0.08	60.4	28.1
s12	143	0.77	50.12	4.0	0.07	53.7	42.2
s15	123	0.51	37.35	3.5	0.08	69.9	26.5
s18	96	0.41	26.01	3.6	0.08	62.3	33.9
Average	265.9	0.84	69.98	4.45%	0.08%	61.69%	33.78%

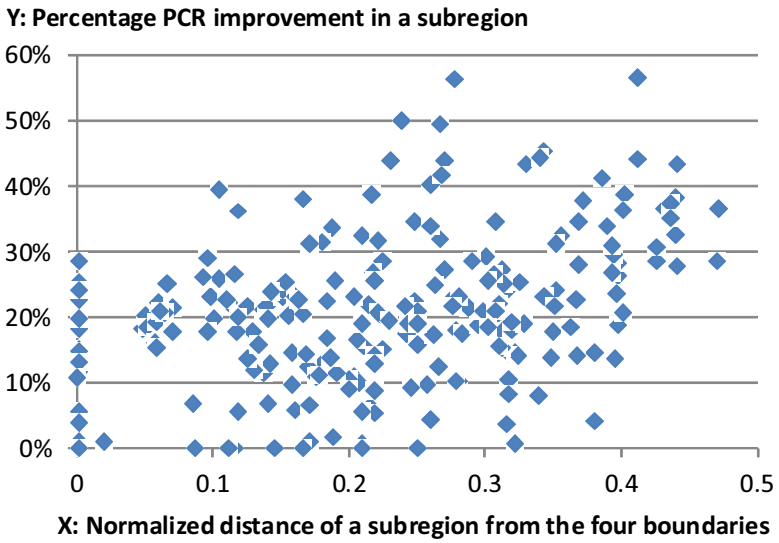


Figure 6.3: Improvement in solution quality as a function of subregion distance from the closest boundary shown for the 227 subregions in `s1`.

Comparison Among the Subregions

Figure 6.3 shows the percentage improvement in solution quality (PCR) that is obtained per subregion as a function of its distance from the closest boundary. This is for benchmark `s1` when running IGR using

the same setup as in the previous experiments. The closest distance is normalized to the corresponding grid size. (So if the closest distance is measured horizontally it is normalized to the size of the X-grid, and similarly if it is vertical it is normalized to the size of the Y-grid.) The Y-axis measure the improvement in PCR (locally) within each subregion. As can be seen from the figure, most of the improvements in PCR are in subregions which are not very close to the boundary. Overall about 8% of the subregions were close to a boundary (with normalized distance of at most 0.05). The average of these local PCR improvements over the subregions is about 20% which is quite high. We note these are higher than the average PCR reported in Table XIII which is measured globally over all the subregions. We think this is due to the combination of two factors: (1) our subregions only cover about 25% of the layout, and (2) these designs are highly congested. Therefore, more improvement is likely for these benchmarks if a higher percentage of the layout was covered.

Analysis of One Subregion

Table XV shows the details when solving one subregion in benchmark `s1`. Iteration 0 is the initial solution. Rows 1 to 7 correspond to the iterations of column generation. (Here the stopping criteria forces termination before reaching the 10 maximum iterations of column generation.) Column 2 shows the actual PCR in iteration 0, and the reduction in PCR (compared to iteration 0) as a percentage in iterations 1 to 7. As expected improvement in PCR increases (from 3.4% to 12.08%) because the total number of candidate routes increase at each iteration. The number of new candidate routes added per iteration is reported in column 3. For example 270 new candidate routes are added at iteration 7. The runtimes of each iteration is also reported in seconds. As can be seen, with increase in the number of candidate routes it takes longer to

Table XV: Analysis for one subregion

Iteration	PCR	#NewCandidateRoutes	Runtime (sec)
0	423328.1	360	1.19
1	3.40%	358	1.49
2	5.61%	354	1.21
3	7.55%	353	1.45
4	8.80%	342	1.72
5	10.06%	300	1.70
6	11.50%	295	2.05
7	12.08%	270	2.36

solve the LP at each iteration.

REFERENCES

Lef def exchange format versions 5.7 reference documentation plus reader.
`https://www.si2.org/openeda.si2.org/projects/lefdefnew/`.

Acharya, Kartik, Kyungwook Chang, Bon Woong Ku, Shreepad Panth, Saurabh Sinha, Brian Cline, Greg Yeric, and Sung Kyu Lim. 2016. Monolithic 3D IC design: Power, performance, and area impact at 7nm. In *International symp. on physical design*, 41–48.

Alpert, Charles J, Zhuo Li, Michael D Moffitt, Gi-Joon Nam, Jarrod A Roy, and Gustavo E Tellez. 2010. What makes a design difficult to route. In *International symp. on physical design*, 7–12.

Alpert, Charles J, Zhuo Li, Cliff N Sze, and Yaoguang Wei. 2013. Patent: Consideration of local routing and pin access during VLSI global routing.

Alpert, Charles J, and G E Tellez. 2010. The importance of routing congestion analysis. In *Dac knowledge center*.

Arabi, Karim, Kambiz Samadi, and Yang Du. 2015. 3D VLSI: A scalable integration beyond 2D. In *International symp. on physical design*, 1–7.

Batterywala, Shabbir H, Narendra V Shenoy, William Nicholls, and Hai Zhou. 2002. Track assignment: A desirable intermediate step between global routing and detailed routing. In *International conf. on computer-aided design*, 59–66.

Billoint, Olivier, Hossam Sarhan, Iyad Rayane, Maud Vinet, Perrine Batude, Claire Fenouillet Béranger, Olivier Rozeau, Gerald Cibrario,

Fabien Deprat, Ogun Turkyilmaz, Sebastien Thuries, and Fabien Clermidy. 2015. From 2D to monolithic 3D: design possibilities, expectations and challenges. In *International symp. on physical design*, 127.

Bustany, Ismail S, David Chinnery, Joseph R Shinnerl, and Vladimir Yutsis. 2015. ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement. In *International symp. on physical design*, 157–164.

Cadence. SoC encounter RTL-to-GDSII system full-chip implementation in a single system.
`http://www.cadence.com/products/di/edi_system/`.

Chang, Yen-Jung, Yu-Ting Lee, Jhih-Rong Gao, Pei-Ci Wu, and Ting-Chi Wang. 2010. NTHU-Route 2.0: A robust global router for modern designs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 29(12):1931–1944.

Chang, Yu-Ning, Yih-Lang Li, Wei-Tin Lin, and Wen-Nai Cheng. 2008. Non-slicing floorplanning-based crosstalk reduction on gridless track assignment for a gridless routing system with fast pseudo-tile extraction. In *International symp. on physical design*, 134–141.

Chen, Huang-Yu, Chin-Hsiung Hsu, and Yao-Wen Chang. 2009. High-performance global routing with fast overflow reduction. In *Asia and south pacific design automation conf.*

Cho, Minsik, Katrina Lu, Kun Yuan, and David Z Pan. 2009. BoxRouter 2.0: A hybrid and robust global router with layer assignment for routability. *ACM Trans. on Design Automation of Electronic Systems* 14(2).

Cho, Minsik, Hua Xiang, Ruchir Puri, and David Z Pan. 2008. Track routing and optimization for yield. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 27(5):872–882.

Graphics, Mentor. Olympus-SoC tool : Place and route for advanced node designs.
`https://www.mentor.com/products/ic_nanometer_design/place-route/olympus-soc/`.

Han, Kwangsoo, Andrew B Kahng, and Hyein Lee. 2015. Evaluation of BEOL design rule impacts using an optimal ILP-based detailed router. In *Design automation conf.*, 68–66.

Hoffman, Alan J, and Joseph B Kruskal. 2010. Integral boundary points of convex polyhedra. *50 Years of Integer Programming* (Chapter 3):49–76.

Hsu, Chin-Hsiung, Huang-Yu Chen, and Yao-Wen Chang. 2008. Multi-layer global routing considering via and wire capacities. In *International conf. on computer-aided design*, 350–355.

Hsu, Meng-Kai, Nitesh Katta, Homer Yen-Hung Lin, Keny Tzu-Hen Lin, King Ho Tam, and Ken Chung-Hsing Wang. 2014. Design and manufacturing process co-optimization in nano-technology. In *International conf. on computer-aided design*, 574–581.

Hu, Jiang, Min Zhao, Rabi N Mahapatra, and Di Wu. 2005. Timing driven track routing considering coupling capacitance. In *Asia and south pacific design automation conf.*, 1156–1159.

IBM. CPLEX Optimization Studio CPLEX User's Manual.
`https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.1/ilog.odms.studio.help/pdf/usrcplex.pdf`.

Jia, Xiaotao, Yici Cai, Qiang Zhou, Gang Chen, Zhuoyuan Li, and Zuowei Li. 2014. MCFRoute: A detailed router based on multi-commodity flow method. In *International conf. on computer-aided design*, 397–404.

Kelly, David. 1985. Comparability graphs. In *Graphs and order*. Springer Netherlands.

Lee, Tsung-Hsien, and Ting-Chi Wang. 2008. Congestion-constrained layer assignment for via minimization in global routing. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 27(9):1643–1656.

———. 2009. Robust layer assignment for via optimization in multi-layer global routing. In *International symp. on physical design*, 159–166.

Lee, Young-Joon, Daniel B. Limbrick, and Sung Kyu Lim. 2013. Power benefit study for ultra-high density transistor-level monolithic 3D ICs. In *Design automation conf.*, 104:1–104:10.

Liu, Wen-Hao, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao. 2013a. NCTU-GR 2.0: Multithreaded collision-aware global routing with bounded-length maze routing. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 32(5):709–722.

Liu, Wen-Hao, Cheng-Kok Koh, and Yih-Lang Li. 2013b. Case study for placement solutions in ISPD11 and DAC12 routability-driven placement contests. In *International symp. on physical design*, 114–119.

Liu, Wen-Hao, and Yih-Lang Li. 2011. Negotiation-based layer assignment for via count and via overflow minimization. In *Asia and south pacific design automation conf.*, 539–544.

Liu, Wen-Hao, Yaoguang Wei, Cliff N Sze, Charles J Alpert, Zhuo Li, Yih-Lang Li, and Natarajan Viswanathan. 2013c. Routing congestion estimation with real design constraints. In *Design automation conf.*, 92.

Pan, Min, and Chris C N Chu. 2007. FastRoute 2.0: A high-quality and efficient global router. In *Asia and south pacific design automation conf.*, 250–255.

Pan, Min, Yue Xu, Yanheng Zhang, and Chris Chu. 2012. FastRoute: An efficient and high-quality global router. *VLSI Design* 362:1–18.

Panth, Shreepad, Kambiz Samadi, Yang Du, and Sung Kyu Lim. 2015. Placement-driven partitioning for congestion mitigation in monolithic 3D IC designs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 34(4):540–553.

Paper, Mentor Graphics White. Understanding physical design constraints in the 10nm era.
`http://go.mentor.com/4O2zY`.

Qiu, Xiang, and Malgorzata Marek Sadowska. 2013. Routing challenges for designs with super high pin density. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 32(9):1357–1368.

Shi, Daohang, and Azadeh Davoodi. 2017. Improving detailed routability and pin access with 3D monolithic standard cells. In *International symp. on physical design*.

Shi, Daohang, Azadeh Davoodi, and Jeffrey T Linderoth. 2016a. A procedure for improving the distribution of congestion in global routing. In *Design, automation and test in europe*, 249–252.

Shi, Daohang, Edward Tashjian, and Azadeh Davoodi. 2016b. Dynamic planning of local congestion from varying-size vias for global routing layer assignment. In *Asia and south pacific design automation conf.*, 372–377.

Shojaei, Hamid, Azadeh Davoodi, and Jeffrey T Linderoth. 2011. Congestion analysis for global routing via integer programming. In *International conf. on computer-aided design*, 256–262.

———. 2013. Planning for local net congestion in global routing. In *International symp. on physical design*, 85–92.

Synopsys. IC Compiler: Place and Route System. `http://www.synopsys.com/Tools/Implementation/PhysicalImplementation/Documents/iccompiler_ds.pdf`.

Viswanathan, Natarajan, Charles J Alpert, Cliff N Sze, Zhuo Li, Gi-Joon Nam, and Jarrod A Roy. 2011. The ISPD-2011 routability-driven placement contest and benchmark suite. In *International symp. on physical design*, 141–146.

Wei, Yaoguang, Cliff N Sze, Natarajan Viswanathan, Zhuo Li, Charles J Alpert, Lakshmi N Reddy, Andrew D Huber, Gustavo E Tellez, Douglas Keller, and Sachin S Sapatnekar. 2012. GLARE: Global and local wiring aware routability evaluation. In *Design automation conf.*, 768–773.

Wong, Man-Pan, Wen-Hao Liu, and Ting-Chi Wang. 2016. Negotiation-based track assignment considering local nets. In *Asia and south pacific design automation conf.*, 378–383.

Xu, Xiaoqing, Brian Cline, Greg Yeric, Bei Yu, and David Z Pan. 2014. Self-aligned double patterning aware pin access and standard cell layout co-optimization. In *International symp. on physical design*, 101–108.

———. 2015a. Self-aligned double patterning aware pin access and standard cell layout co-optimization. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 34(5):699–712.

Xu, Xiaoqing, Bei Yu, Jhih-Rong Gao, Che-Lun Hsu, and David Z Pan. 2015b. PARR: pin access planning and regular routing for self-aligned double patterning. In *Design automation conf.*, 28–26.

———. 2016. PARR: Pin-access planning and regular routing for self-aligned double patterning. *ACM Trans. on Design Automation of Electronic Systems* 21(3):42:1–42:21.

Xu, Yue, and Chris Chu. 2011. MGR: Multi-level global router. In *International conf. on computer-aided design*, 250–255.

Xu, Yue, Yanheng Zhang, and Chris Chu. 2009. FastRoute 4.0: Global router with efficient via minimization. In *Asia and south pacific design automation conf.*, 576–581.

Zhang, Yanheng, and Chris Chu. 2011. RegularRoute: An efficient detailed router with regular routing patterns. In *International symp. on physical design*, 45–52.

———. 2012. GDRouter: Interleaved global routing and detailed routing for ultimate routability. In *Design automation conf.*, 597–602.

———. 2013. RegularRoute: An efficient detailed router applying regular routing patterns. *IEEE Trans. on Very Large Scale Integration Systems* 21(9):1655–1668.

Zhang, Yanheng, Yue Xu, and Chris Chu. 2008. FastRoute3.0: A fast and high quality global router based on virtual capacity. In *International conf. on computer-aided design*, 344–349.